Linear logic, type assignment systems and implicit computational complexity

Erika de Benedetti

Thèse

en vue de l'obtention du grade de **Docteur de l'Université de Lyon**, délivré par
l'École Normale Supérieure de Lyon
en cotutelle avec *Università degli Studi di Torino*

Discipline: Informatique
Laboratoire de l'Informatique du Parallélisme
École Doctorale InfoMaths

présentée et soutenue publiquement le 10 février 2015
par Erika De Benedetti

Linear logic, type assignment systems and implicit computational complexity

directeur de thèse	Patrick Baillot
co-tutrice de thèse	Simona Ronchi Della Rocca

Devant la commission d'examen formée de :

Patrick Baillot, ENS de Lyon	(directeur)
Ugo Dal Lago, Università di Bologna	(rapporteur)
Delia Kesner, Université Paris Diderot	(examinateur)
Jean-Yves Marion, Loria INPL - École des Mines de Nancy	(rapporteur)
Simona Ronchi Della Rocca, Università di Torino	(co-tutrice)
Aleksy Schubert, Warsaw University	(examinateur)

Contents

List of Figures

List of Tables

Abstract

In this thesis we explore the linear logic approach to implicit computational complexity, through the design of type assignment systems based on light linear logic, or heavily inspired by them, with the purpose of giving a characterization of one or more complexity classes, through variants of λ-calculi which are typable in such systems.
In particular, we consider both a monovalent and a polyvalent perspective with respect to ICC. In the first one the aim is to characterize a hierarchy of complexity classes through an elementary λ-calculus typed in Elementary Linear Logic (ELL), where the complexity depends only on the interface of a term, namely its type. The second approach gives an account of both the functions computable in polynomial time and of strong normalization, through terms of pure λ-calculus which are typed in a system inspired by Soft Linear Logic (SLL); in particular, with respect to the usual logical perspective, in the latter we erase the "!" modality in favour of employing stratified types as a refinement of non-associative intersection types, in order to improve typability and, as a consequence, expressivity.
Finally we explore the use of intersection types, deprived of some of their usual properties, towards a more quantitative approach rather than the usual qualitative one, namely in order to compute a bound on the computation of pure λ-terms, obtaining in addition a characterization of strong normalization.

Riassunto della tesi

Lo studio della complessità implicita (ICC) si pone come obiettivo la caratterizzazione di classi di complessità tramite dei linguaggi di programmazione o delle logiche, senza fare alcun riferimento esplicito a limiti di risorse (ad esempio tempo o spazio di memoria). In questa tesi viene studiato l'approccio della logica lineare alla complessità implicita. L'obiettivo è quello di dare delle caratterizzazioni implicite di classi di complessità, attraverso delle varianti del λ-calcolo che siano tipabili in tali sistemi.

In particolare, consideriamo sia una prospettiva monovalente sia una polivalente rispetto all'ICC. Nel primo caso, l'idea è di dare una caratterizzazione una gerarchia di classi di complessità attraverso un λ-calcolo elementare tipato nella logica lineare elementare (`ELL`), in cui la complessità dipende solamente dall'interfaccia di un programma, ovvero il suo tipo. Il secondo approccio, invece, si propone di caratterizzare sia le funzioni calcolabili in tempo polinomiale sia la normalizzazione forte, attraverso termini del λ-calcolo puro a cui viene assegnato un tipo tramite un sistema ispirato alla logica lineare Soft (`SLL`); in particolare, rispetto all'approccio logico, qui abbandoniamo la modalità "!" a favore dell'utilizzo di tipi stratificati, visti come un raffinamento dei tipi intersezione non associativi, con il fine di aumentare la tipabilità e, come conseguenza, l'espressività.

Infine, esploriamo l'utilizzo dei tipi intersezione, privati di alcune delle loro proprietà abituali, in una direzione quantitativa piuttosto che il classico approccio qualitativo, con il fine di calcolare un limite per il calcolo dei λ-termini puri, ottenendo nello stesso tempo anche una caratterizzazione della normalizzazione forte.

Résumé de thèse

La complexité implicite (ICC) vise à donner des caractérisations de classes de complexité dans des langages de programmation ou des logiques, sans faire référence à des bornes sur les ressources (temps, espace mémoire). Dans cette thèse, nous étudions l'approche de la logique linéaire à la complexité implicite. L'objectif est de donner des caractérisations de classes de complexité, à travers des variantes du λ-calcul qui sont typables dans de tels systèmes.

En particulier, nous considérons à la fois une perspective monovalente et une perspective polyvalente par rapport à l'ICC. Dans le premier cas, le but est de caractériser une hiérarchie de classes de complexité à travers un λ-calcul élémentaire typé dans la logique linéaire élémentaire (ELL), où la complexité ne dépend que de l'interface d'un programme, c'est à dire son type. La deuxième approche rend compte à la fois des fonctions calculables en temps polynomial et de la normalisation forte, à travers des termes du λ-calcul pur qui sont typés dans un système inspiré par la logique linéaire Soft (SLL) ; en particulier, par rapport à l'approche logique ordinaire, ici nous abandonnons la modalité " !" en faveur de l'emploi des types stratifiés, vus comme un raffinement des types intersection non associatifs, afin d'améliorer la typabilité et, en conséquence, l'expressivité.

Enfin, nous explorons l'utilisation des types intersection, privés de certaines de leurs propriétés, vers une direction plus quantitative que l'approche qualitative habituelle, afin d'obtenir une borne sur le calcul de λ-termes purs, en obtenant en plus une caractérisation de la normalisation forte.

Acknownlodgments

I would like to thank my supervisors, Simona and Patrick, for accompanying me through this journey, as well as Ugo and Jean-Yves who kindly accepted to review this thesis.

I would like to thank my family for their continuous support and enthusiasm.

Last, but not last, I would like to thank Alberto for his infinite patience and support through my ramblings and all the months spent abroad, and for never giving up on me.

⋆ This work was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program *Investissements d'Avenir* (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

Chapter 1

Introduction

The careful control of resource usage has gained more and more importance in the field of computer science, due to the growing difficulty of problems that we wish to solve, and for which the *time* and the *storage* needed to compute are often the essence (and money).
Such aim has spurred a continuous growth in the study of computational complexity and on the classification of algorithms, based on the time or space they require in order to produce an output. In particular, only a subclass of the computable problems are of practical interest from a computer science point of view, while the remaining ones are called *intractable*, because of the tremendous amount of resources they require for their computation when supplied with a large enough input; in order to tackle such problems, or at least some recurring cases of them, a range of non-deterministic algorithms has been proposed, such as probabilistic and approximation algorithms, whose underlying idea is to sacrifice the accuracy of the result in exchange for a more careful exploitation of both time and space resources.

The main issue concerning the inherent difficulty of a problem is the fact that it does not depend on the particular language in which the program is written, nor on the machine on which the algorithm itself is run: in fact, by considering larger and larger input's sizes, it is easy to see that intractable problems would require amount of resources sometimes surpassing the very age of the Universe! Since any problem can be classified regardless of its code or the support upon which it is executed, the theory of computational complexity is mainly studied on a given *computational model*, such as the Turing machine [Sip97]. Indeed, by the Church-Turing thesis it is a commonly accepted fact that all Turing-complete models are equally powerful, in the sense that any problem, which is computable on a given computational model, is also computed by a Turing machine: most importantly, the computation can be performed on a Turing machine with only a polynomial overhead with respect to the time of execution required on most computational models. Because Turing machines

are also prone to be easily analyzed in a formal way, such model is usually chosen as the principal model in complexity theory.

Nonetheless, classical complexity theory presents some shortcomings, mostly deriving from the fact that it is based on a particular computational model: this ultimately means that the complexity measure is in a sense "given" from outside, exposing the fact that a proof of complexity is not so easily formalised. Moreover, it is a well known fact that if there is a Turing machine with two tapes representing some algorithm, then a Turing machine with one tape can be built, doing essentially the same job with a negligible increase in computation time [Pap94]. It is clear, based on such premises, that writing a formal proof becomes quite the challenge.

For these reasons, the idea arose of turning to formal methods in order to give a proper account of complexity classes and their properties, in a way which is ideally independent on any machine model: this allows to give an implicit characterization of complexity classes, through the design of systems where the complexity of programs is statically verified, beside their usual correctness criterions.
In the following subsections we introduce the main ingredients of such approach, which can be found in certain refinements of classical and intuitionistic logics, and also in the use of the simplest paradigmatic functional language, that is λ-calculus. We then give a brief review of the state of the art concerning implicit computational complexity. A short history of intersection types and their recent use for proving quantitative properties of programs follows. We conclude with an overview of the content of this thesis.

1.1 Girard's linear logic

Intuitionistic (or constructive) logic was introduced as a restriction of classical logic, where the excluded middle and the involution of the negation are no more provable; as a result, the notion of truth in classical logic is no more applicable: indeed, a propositional formula of intuitionistic logic is considered true only if there is an object demonstrating its existence, that is, a constructive *proof* of the formula itself.
The constructivism of such logic is particularly suited for computer science, moreover so since the Curry-Howard correspondence relates the proofs of intuitionistic logic to programs written in some functional language. In practice, this means that constructive proofs can be rewritten as functional programs, which in turn can be run: indeed computation corresponds to the cut-elimination procedure in the case of sequent calculus, or to reduction of a proof to normal form in the case of natural deduction. It is then possible to study the computational complexity of such proofs-as-programs simply by examining their reduction steps.

Linear logic was introduced by Girard [Gir87] with the aim of combining the inherent duality of classical logic with the constructive nature of intuitionistic logic, the latter allowing to extract programs from proofs, so proving to be of great interest in particular for computer science.
The intuition behind this new logic comes from the need of a more careful handling of resources, something both classical and intuitionistic logic lack. In particular, unrestricted weakening and contraction were the main problems to overcome: indeed, not all resources are infinite in nature, on the contrary most of them are consumed and thus they are not available anymore after usage. In order to achieve this effect, Girard proposes a careful restriction of the so-called structural rules: indeed, these can no more be disregarded as meaningless, but are to be treated explicitly in order to gain a finer control on the number of premises needed to prove a formula.

Such proposal has the immediate consequence of splitting each binary connective of classical logic in two connectives, respectively an additive and a multiplicative one: $\wedge$ becomes & and $\otimes$, while $\vee$ becomes $\oplus$ and $\parr$. However, the additive and multiplicative fragments of linear logic are not very interesting from the point of view of the cut-elimination procedure: indeed, cut-elimination can be carried out in a linear number of steps with respect to the size of the proof, so the programs which we can extract from such proofs are not very interesting from a computational complexity point of view.

In order to overcome this lack of power, one has to consider the structural rules, allowing duplication and erasure in cut-elimination, but also the rules for dereliction and promotion, which allow to control such duplications. To this aim, Girard introduces the *exponentials* ! and ?, as a way to gain expressivity in a controlled manner. This marks a clear distinction between consumable (or linear) and reusable resources, as the former disappear after they are used, while the latter -duly marked- can be thought of as being available in an infinite number of copies. This is clearly a big advantage to be exploited, as exponentials allow for much more expressivity: for this reason, the exponential fragment of linear logic is most useful when linear logic is used as a tool for the study of computational complexity.

1.2 Type assignment systems for λ-calculus

Alonzo Church proposed the formal system of λ-calculus (see [Bar84]) in the 30s, in order to give a formal account of functions and the means to compute them. Such calculus is based on a formal language endowed with a set of rewriting rules; the terms of the language are variables, abstractions (representing functions) and applications (representing applications of functions to their arguments), while the usual reduction rule allows the computation of a function simply through the substitution of the

$$\mathcal{M}, \mathcal{N} ::= x \mid \lambda x.\mathcal{M} \mid \mathcal{M}\mathcal{N} \quad \text{where } x \in \texttt{Var}$$

Figure 1.1: Syntax of pure λ-calculus.

argument of an application into the body of the function, which may be triggered when some condition on the redex is met (see [RP04] for a survey of the parametric λ-calculus).
This calculus, although very basic, is Turing complete, in the sense that it allows to capture all computable functions. Such fact, combined with the elegance and simplicity of the language, has made it so that λ-calculus is commonly accepted as the paradigmatic programming language in the functional setting, on which later functional languages such as LISP, ML and Haskell have been based: the functional core of all these languages share some common features, such as programs representing the nested evaluation of functions and the absence of side effects, in the form of storing and fetching of values into and from memory.

The λ-calculus has gained even more attention after the Curry-Howard isomorphism was proposed, relating it to the natural deduction formulation of intuitionistic logic, and so giving way to *strongly typed languages*: indeed, the logic becomes a *type assignment system* for the language, where hypotheses and formulas correspond respectively to free variables and types, while implication elimination and introduction correspond to application and abstraction respectively. This intuitively means that the shape of a proof in natural deduction can be "read" from the term, so linking the structure of a derivation with the programs typed by it and allowing such programs to inherit the good properties of the logic, like termination and correctness: this allows to capture only a proper subset of the λ-terms displaying some desirable properties of "sensible" programs, that is, those programs which always produce an output and never go wrong during execution.

There are mainly two versions of the typed standard λ-calculus, due to Church and Curry respectively. In Church-style λ-calculus, only well-typed terms (so typing derivations) are assigned a meaning, in the sense that the type of the formal parameter of an abstraction is explicitly annotated on the term, so terms differing only by type annotations can be assigned different meanings. On the other hand, in Curry-style λ-calculus the terms are assigned a meaning independently on their typing, so they are interpreted as they would in an untyped setting. See [BDS13] for an extensive survey of typed λ-calculus and its applications. In this thesis we will focus on Curry-style λ-calculus (see Figure 1.1).

1.3 Implicit computational complexity

The idea of using a (functional) programming approach to complexity theory is explored in [Jon97; Jon01], where complexity classes are the mean to characterize the expressivity of some features of programming languages, such as constructors and control flow structures.

The interest behind an intrinsic definition of complexity classes is twofold:

- from the computational complexity point of view, they offer characterizations of complexity classes which are not tied to machine models;
- from the programming languages point of view, they allow to analyze the impact of various programming features (such as higher-order types, recursive definitions or I/O operations) on complexity.

Such intuitions form the basis for the research line of *implicit computational complexity* (ICC), whose goal is to study complexity classes without relying on explicit bounds on resources, but instead by considering restrictions on programming languages and calculi. A very early work in this direction [Cob65] shows that the class of functions which are definable by bounded recursion on notation and basic operations on binary words is exactly the class of polynomial time computable functions.

A pervasive concept in the field of implicit complexity is that of *stratification*, meant as the distribution of the computation over different strata. This notion is the foundation of several systems: ramified and safe (or tiered) recursion [Lei94; BC92], in which the variables of a function are divided into strata and the results of recursive calls are only allowed to be processed via certain strata, in order to prevent unwanted recursion; stratified comprehension [Lei02], where strata are used for quantification and the classification of first order set-existence by implicational rank yields a natural hierarchy of complexity classes inside the class of elementary functions; various restrictions of linear logic [Gir98], where programs are divided into strata by the modality ! and the complexity of the normalization procedure depends on the depth of such strata. Stratification of variables into different strata is also involved in secure flow information analysis, where the flow of information from lower to higher strata must be forbidden in order to satisfy a non-interference condition [Mar11].

In this line of work, a distinction can be made between *monovalent* characterizations, in which one language corresponds to one given complexity class, and *polyvalent* characterizations, in which the language allows to capture several complexity classes. The first approach is further analyzed in the next section. As for the latter, some examples of polyvalent characterizations are given by [Jon01; Lei02] which capture the family of classes k-`EXP`, for $k \geq 0$, and in [Red13], where a stratified version of

combinatory logic is used in order to prove that the subsystem restricted to the levels less than or equal to $k+1$ characterizes k-EXP, so in particular PTIME when $k = 0$.

1.3.1 Linear logic in ICC

As introduced in the previous sections, the linear logic approach to ICC is based on the proofs-as-programs correspondence of the Curry-Howard isomorphism. Linear logic provides a powerful system to analyze the duplication and sharing of arguments in typed λ-calculus and thus in functional languages: indeed, if a formal parameter occurs many times in a λ-abstraction, then the reduction of a term containing an application of such function can very well cause a complexity explosion. For this reason, the idea is to use weak versions of !, in order to restrict the set of well-typed terms and characterize complexity classes.

The use of exponentials can be restricted so as to limit the set of formulas which can be proved in the system, thus characterizing different classes of functions. Some subsystems of linear logic, called *light logics*, have been studied from the point of view of the complexity of the cut-elimination procedure.

In order to obtain such characterization, one usually considers some encoding of data (for example Church numerals or booleans) and then proceeds to show that the system is sound and complete with respect to the functions in a given complexity class (or a hierarchy of classes), where terms of the language are functional programs representing the desired functions, whose type is in accord with the definition of the chosen datatypes.

Monovalent characterizations Among the so-called *light logics*, Elementary Linear Logic (ELL) [Gir98; DJ03] has a very simple syntax and it aims to characterize (in the proofs-as-programs sense) the class of elementary functions, that is those functions computable in time bounded by a tower of exponentials of fixed height. Both Light Linear Logic (LLL) [Gir98] and Bounded Linear Logic (BLL) [GSS92] characterize the complexity class of polynomial time functions, but the syntax of both logics is slightly more complicated due to the presence of a second modality in LLL and of explicit polynomials in BLL. Another proposal in this sense came from Lafont [Laf04] and his Soft Linear Logic System, where only the ! modality is needed and the control of both contraction and weakening is maintained by the multiplexor rule.

Such logical systems can be adapted to act as type assignment systems for (variants of) λ-calculus or other functional languages, transferring the time complexity bounds from the logical derivation to well-typed programs.
For some subsystems of linear logic, a decoration with λ-calculus has been given, with the aim of relating the number of reduction steps of the logical derivations

with the complexity of the normalization procedure of a term, that is, the resources needed in order to completely evaluate a term. In particular there are already two proposals in this direction: the system DLAL [BT04], whose types are a proper subset of formulae of LAL [AR02], a simplified affine version of LLL, and the system STA [GR07], whose types are a proper subset of SLL formulae. On a similar note, in [MA11] the authors design an elementary λ-calculus typed in ELL, endowed with both the ! modality and a let construction, which is then extended to a setting of concurrent parallel threads with references.

Polyvalent characterizations The polyvalent characterization from which we take most inspiration is given in [Bai11], in which the author offers a sound and complete characterization of k-EXP for $k \geq 0$ through ELL. Here *proof-nets* are the programming language of choice: such construct is essentially a representation of proofs as graphs, where nodes represent rules and edges represent formulae, which are useful in order to reason about cut-elimination.

1.3.2 Extensional completeness of logic-based systems

When speaking about *extensional completeness* of a language with respect to some complexity class, we mean that for each function in such class there is at least one program in the language representing it. However, all languages which are typed in a logical system (such as the ones mentioned above) suffer from a common lack of expressivity, due to the constraints imposed by the underlying type assignment system, in the sense that they are able to capture only a subset of all the algorithms representing a function in the desired complexity class.

Ideally, we are interested in identifying some criteria which allow to type more and more programs, as long as the complexity bound is met, in order to obtain more *intensional expressivity*.
In this direction, one possible approach is to enrich the features offered by the programming language, such as higher-order types, polymorphism and the handling of exceptions. This idea is explored in [Hof99], where the author proposes a typed λ-calculus with unrestricted recursion operators, based on the observation that some usually rejected programs actually do not increase the size of their input, since their repeated iteration does not break the polynomial time bound.
In [CS12] an extension of STA is given, where λ-calculus is enriched with some features like ML-polymorphism. In [BGM10] the authors propose a typed extension of DLAL, resulting in a more user-friendly and modular language called LPL which features pattern-matching and recursive definitions, while ensuring the polynomial bound through the rejection of exponential recursive definitions.

1.4 Intersection type systems for complexity

If one wants to maintain pure λ-calculus as programming language, one possible approach, which is extensively explored in this thesis, is to enrich the typing system in order to expand the class of terms which can be typed in it, and thus the expressive power of the language itself: a natural tool to this aim can be found in *intersection types* [CD80].

The formulae-as-types correspondence can be trivially extended with the logical conjunction, by endowing the pure λ-calculus accordingly with a pair constructor and two pair selectors. In this case, the associated reduction rules of introduction and elimination of a pair correspond to the typing rules for the newly introduced constructs.
The idea of intersection types stems from the observation that the conjunction can also be interpreted in a way which is not related to the Curry-Howard correspondence, in the sense that a term might be assigned many types at the same time, that is, an *intersection* of types:

$$\frac{\Gamma \vdash \mathcal{M} : \sigma \quad \Gamma \vdash \mathcal{M} : \tau}{\Gamma \vdash \mathcal{M} : \sigma \wedge \tau}\ (\wedge I) \qquad \frac{\Gamma \vdash \mathcal{M} : \sigma \wedge \tau}{\Gamma \vdash \mathcal{M} : \sigma}\ (\wedge E_l) \qquad \frac{\Gamma \vdash \mathcal{M} : \sigma \wedge \tau}{\Gamma \vdash \mathcal{M} : \tau}\ (\wedge E_r)$$

Figure 1.2: Derivation rules for $\wedge$.

Clearly, such type constructor does not correspond to any logical connective in the formulae-as-types correspondence: if a term is assigned an intersection type $\sigma \wedge \tau$, then the same term must be typable both by σ and τ; this means that the subderivations typing the term must have exactly the same structure, a meta-condition which would no more be expressed if the type assignment system were stripped off the calculus.

The basic type assignment system is obtained by endowing the simply typed λ-calculus with the rules of Figure 1.2, and has been proven to be sound and complete with respect to the class of strongly normalizing λ-terms, that is, the terms whose normalization procedure terminates regardless of which reduction strategy is used. Moreover, with the introduction of a universal type ω and of a partial order relation, the resulting typing system characterizes the class of normalizing terms, that is, the terms for which there exists a reduction sequence to normal form. See [DCGL98] for a review of intersection types.

Historically, intersection types are considered modulo the three equations given in Figure 1.3. Indeed, intersection types were born with the aim of studying qualitative properties of λ-terms, such as solvability, normalization and strong normalization. Lately, however, the interest has shifted towards less standard formulations of the intersection types discipline, that is, intersection types enjoying various combinations

$$\sigma \wedge \sigma = \sigma \quad \text{(idempotence)} \tag{1.1}$$

$$\sigma \wedge \tau = \tau \wedge \sigma \quad \text{(commutativity)} \tag{1.2}$$

$$(\sigma \wedge \tau) \wedge \rho = \sigma \wedge (\tau \wedge \rho) \quad \text{(associativity)} \tag{1.3}$$

Figure 1.3: Properties of intersection types.

of the three properties shown in Figure 1.3. In fact, by considering non idempotent or non associative intersection types, it is possible to reason also about *quantative properties* of terms, such as bounds on the number of normalization steps or on the size of the normal form with respect to the initial term or derivation.

In particular, non idempotent intersection types (see [CDCV80]) seem to be suited to this purpose, since they retain a precise information about the number of copies of a term which will be needed during reduction. In [Kfo00] non idempotent intersection types are related to linear reduction through a linearization of λ-calculus, where all copies of the argument of a function are explicitly written in the term, with the aim of showing the characterization of strongly normalizing λ-terms by a type assignment system of intersection types. In [KW04], the authors employ non idempotent intersection types in order to design a unification-based type inference algorithm, which finds the principal typing of a term when a finite-rank restriction of the typing system is considered. Moreover, a quantitative approach is taken in [KV14], where the authors consider a typing system of non-idempotent intersection types in order to characterize linear-head, head, weak and strong normalization through linear substitution calculus, namely a typed calculus with explicit substitutions.

Recently, non idempotent intersection types have been used [PR10] to characterize the solvability of λ-terms in a variant of the calculus which allows to model resource consumption, where the arguments of a function are represented as finite multisets of (either linear or reusable) resources.

Both [dCa09] and [BL11] offer some insights about complexity of reduction through non idempotent intersection types: the former relates the size of a type derivation to the number of steps required to execute a term, using Krivine's machine as computational model, while the latter gives a bound on the size of the longest reduction sequence, which can be read from explicit labels written over type derivations.

Following the direction suggested by these works, in this thesis we propose several uses of intersection types for quantitative purposes, one of which marks the first use of intersection types in the ICC setting.

1.5 Content of this thesis

The present manuscript focuses on the characterization of complexity classes through type assignment systems, based both on light linear logics and intersection types; moreover, we use intersection types with the aim of studying both quantitative and qualitative properties of λ-terms. In the following, we offer a brief overview of the content of each chapter.

Chapter 2: A polyvalent characterization of the k-EXP hierarchy through an elementary λ-calculus typed in ELL. We consider a version of pure λ-calculus extended with the ! modality. For the normalization of the application of a program to a binary word up to a given stratum, we prove a complexity result in terms of a bound on both the number of normalization steps and the size of the resulting term. In order to obtain a characterization of the desired hierarchy for both predicates and functions, we introduce a type assignment system assigning formulae of ELL to well-formed terms of the calculus. We show a correspondence between the complexity class of k-EXP and all such programs whose output type's depth depends on k, thus effectively giving a characterization of the whole hierarchy of classes k-EXP, for both predicates and functions. Moreover, we supply another characterization of functions based on a representation of binary words through pairs, which allows to capture some new properties of complexity classes such as the closure of FPTIME with respect to composition.

Chapter 3: A monovalent characterization of FPTIME through stratified types. Starting from the STA system, we design a type assignment system for pure λ-calculus where the ! modality is substituted by *stratification* of types, that is, by a non associative version of intersection types. The distinctive trait of such a construction is the fact that the nesting of a type maintains a trace of its stratification history, in order to prevent idempotence from erasing any quantitative information from types. Such type system, named STR, guarantees both the usual correctness and the complexity bound for the calculus typed in it; indeed, both STA and STR characterize the class FPTIME, but the latter is much more expressive in that (as usual intersection types systems) all strongly normalizing terms are typable, while in STA only a strict subset of them is accepted. This result marks the first work in the ICC field making use of intersection types; moreover, here we extend the result given in [BPS03], where the authors prove the equivalence of simple types and intersection types with respect to the class of function represented by typed terms: in particular, we show that soft types and stratified types allow to characterize the same class of functions, yet stratified types entail more expressivity in the sense that they allow more programs to be typed.

Chapter 4: Bounding normalization of λ-terms through non idempotent intersection types. It is well known that leaving out the idempotence property is enough in order to recover some quantitative properties of λ-terms: indeed, since the number of copies of an argument is written in its type, the derivation contains all necessary copies of it and so the derivation shrinks at each reduction step. The system we consider is similar to the previous one, where the non associativity is dropped in favor of a flat, non stratified construction of types. By adapting the usual notion of strata to this setting, we bound the number of normalization steps and the size of the normal form of a term with respect to the size and the depth of the initial term; in order to prove this result, we consider a definition of weight of a derivation depending on the number of premises contracted by each application of the multiplexor rule, so achieving a more interesting bound than the naive one resulting from the standard definition of size.

Chapter 2

Characterizing exponential complexity classes through elementary λ-calculus

Elementary linear logic (ELL) is a light linear logic which was introduced in order to give a monovalent characterization of elementary complexity, namely computation whose time (or space) is bounded by a tower of exponentials of fixed height. In [Bai11], the author shows that it is also possible to use ELL, extended with a type fixpoint, in order to give a polyvalent characterization of the complexity classes of problems k-EXP, for $k \geq 0$, and in particular to capture the class PTIME when $k = 0$; this result is achieved by proving that each complexity class corresponds to proofs whose conclusion depends on the parameter k. Such results are proved on proof-nets and rely on a precise analysis of their normalization procedure, thus their understanding requires a certain background knowledge in this field; furthermore, the characterization is limited to the class of predicates k-EXP, since the author uses a semantic argument which cannot be trivially adapted to the case of functions.

In this chapter we recast the methodology of !-stratification in the more standard and widely known setting of λ-calculus, in order to show that the resulting simple language is sufficient to obtain a polyvalent characterization not only of the hierarchy of problems k-EXP, but also of the hierarchy of functions k-FEXP, for $k \geq 0$.
Observe that the results of [Bai11] cannot be trivially extended to λ-calculus by considering a straightforward translation of λ-terms into proof-nets, since cut-elimination cannot be directly simulated by reduction: indeed, the proof of the complexity bound in the case of proof-nets follows a specific cut-elimination strategy; moreover, it uses a partly semantic argument. For such reasons we need to define some new measures on terms, which are not adapted from proof-nets, in order to give a direct proof of the same result in $\lambda^!$-calculus: we believe that this λ-calculus-based approach

may be of help in revealing the underlying working principles of ELL and in building characterizations of other complexity classes.
A nice aspect of this system, with respect to former polyvalent characterizations [Jon01; Lei02], is the fact that the complexity bound can be deduced by looking only at the interface of the program, namely its type, without referring to its constructions steps. Moreover, we distinguish the respective roles played by the syntactic aspect (well-formedness) and the typing; this allows us to illustrate how types can provide two different characterizations of the class k-FEXP, based only on the use of different datatypes, and it could prove to facilitate the possible future usage of such elementary $\lambda^!$-calculus with other typing systems.

Previous related works on ELL [Gir98; DJ03; Maz06; DL09] are all carried out in the setting of proof-nets. Other syntaxes have since then been investigated: in [Ter07; MA11] the authors propose each a specific term calculus corresponding to LLL and ELL respectively; in [CDR08] the standard λ-calculus is typed with a type system derived from ELL but, in order to maintain the stratification property, the call-by-value paradigm of reduction has to be assumed.
Our elementary $\lambda^!$-calculus is similar in its syntax to the one of [RR97; DMZ10], while our type assignment system is clearly inspired by [CDR08].

Outline of this chapter We begin with the definition of an elementary λ-calculus (Section 2.1), which is a variant of pure λ-calculus with an explicit account of the ! modality, and we consider the subset of all such terms satisfying a given well-formedness condition. The stratification of the computation then allows to obtain an elementary complexity bound when applying a well-formed program to a data: namely, the k-approximation of the computation can be performed in time bounded by a tower of 2's of height k, whose exponent depends on the size of the supplied data. Nevertheless, in order to characterize complexity classes we must consider both a complete computation and the shape of the result. For these reasons we introduce a type assignment system (Section 2.2), inspired by ELL, such that all well-typed terms are also well-formed and functions are characterized by their type: the result is a sound and complete characterization of k-EXP and k-FEXP, for $k \geq 0$, and in particular each hierarchy of complexity classes corresponds to a hierarchy of types depending on k. However, the latter characterization of functions is a bit disappointing, in the sense that it is does not account for the compositional closure of the complexity class FPTIME; in order to overcome this issue, we propose a different, maybe less natural typing for functions, which allows to compose programs whose interface (type) corresponding to the FPTIME class (Section 2.3). Finally, we show how such newly defined type can be employed fruitfully in order to characterize other

complexity classes (Section 2.4). The core of the current chapter is presented in [BDR14].

2.1 The elementary λ-calculus

2.1.1 Language of terms and term contexts

We introduce an elementary λ-calculus, which adds to ordinary λ-calculus a !-modality and distinguishes between two notions of λ-abstraction. The language of terms is defined by the following grammar:

$$\mathcal{M},\mathcal{N} ::= x \mid \lambda x.\mathcal{M} \mid \lambda^! x.\mathcal{M} \mid \mathcal{M}\mathcal{N} \mid !\mathcal{M}$$

where, as usual, x ranges over a countable set of term variables `Var`. The set of terms is denoted by $\Lambda^!$. The language itself is not new [RR97; DMZ10]; however, in the sequel we will focus our attention on a strict subset of terms, for which a specific condition of well-formedness holds.

As usual, terms are considered up to α-equivalence and the symbol $=$ denotes the syntactic equality modulo such renaming. Moreover, we assume the so-called *variable convention*, that is, all bound variables are different from the free variables, so that capture of free variables is prevented after substitution.
The usual notions of free variables, substitution and number of occurrences hold:

Definition 1 (Free variables)**.**

$$\begin{array}{lcl}
\mathrm{FV}(x) & = & x \\
\mathrm{FV}(\lambda x.\mathcal{M}) & = & \mathrm{FV}(\mathcal{M}) \setminus \{x\} \\
\mathrm{FV}(\lambda^! x.\mathcal{M}) & = & \mathrm{FV}(\mathcal{M}) \setminus \{x\} \\
\mathrm{FV}(\mathcal{M}\mathcal{N}) & = & \mathrm{FV}(\mathcal{M}) \cup \mathrm{FV}(\mathcal{N}) \\
\mathrm{FV}(!\mathcal{M}) & = & \mathrm{FV}(\mathcal{M})
\end{array}$$

Definition 2 (Substitution)**.**

$$\begin{array}{lcl}
y[\mathcal{N}/x] & = & \begin{cases} \mathcal{N} & \text{if } x = y \\ y & \text{otherwise} \end{cases} \\
(\lambda y.\mathcal{M})[\mathcal{N}/x] & = & \begin{cases} \lambda y.(\mathcal{M}[\mathcal{N}/x]) & \text{if } y \notin \mathrm{FV}(\mathcal{N}) \\ \lambda z.((\mathcal{M}[z/y])[\mathcal{N}/x]) & \text{otherwise} \end{cases} \\
(\lambda^! y.\mathcal{M})[\mathcal{N}/x] & = & \begin{cases} \lambda^! y.(\mathcal{M}[\mathcal{N}/x]) & \text{if } y \notin \mathrm{FV}(\mathcal{N}) \\ \lambda^! z.((\mathcal{M}[z/y])[\mathcal{N}/x]) & \text{otherwise} \end{cases} \\
\mathcal{M}\mathcal{P}[\mathcal{N}/x] & = & \mathcal{M}[\mathcal{N}/x][\mathcal{N}/x] \\
(!\mathcal{M})[\mathcal{N}/x] & = & !(\mathcal{M}[\mathcal{N}/x])
\end{array}$$

Definition 3 (Free occurrences)**.**

$$\begin{array}{lcl} n_0(x,y) & = & \begin{cases} 1 & \textit{if } x = y \\ 0 & \textit{otherwise} \end{cases} \\ n_0(x, \lambda y.\mathcal{M}) & = & n_0(x,\mathcal{M}) \quad \textit{where } x \neq y \\ n_0(x, \lambda^! y.\mathcal{M}) & = & n_0(x,\mathcal{M}) \quad \textit{where } x \neq y \\ n_0(x, \mathcal{M}\mathcal{P}) & = & n_0(x,\mathcal{M}) + n_0(x,\mathcal{P}) \\ n_0(x, !\mathcal{M}) & = & n_0(x,\mathcal{M}) \end{array}$$

In the following, we use the notation $!^i$ as a short for $\underbrace{!...!}_{i}$.

We consider the class of *term contexts*, with possibly several holes, generated by the following grammar:

$$\mathcal{C} ::= \Box \mid x \mid \lambda x.\mathcal{C} \mid \lambda^! x.\mathcal{C} \mid \mathcal{C}\mathcal{C} \mid !\mathcal{C}$$

where the symbol $\Box$ represents the *hole* in the term context. A context containing a single hole is a *simple context*.

Let $\mathcal{C}$ be a context having n holes, for $n \geq 1$: then $\mathcal{C}[\mathcal{N}_1]...[\mathcal{N}_n]$ is the term obtained by plugging the term $\mathcal{N}_j$ into the j-th hole of the context $(1 \leq j \leq n)$, where the holes are assumed to be ordered from left to right. If $n = 1$, then the case is that of a simple context.
Observe that, as usual, capture of variables may occur.

We define an *occurrence* of a term $\mathcal{N}$ in $\mathcal{M}$ as a simple context $\mathcal{C}$ such that $\mathcal{M} = \mathcal{C}[\mathcal{N}]$; in practice, we will simply write $\mathcal{N}$ for the occurrence, if there is no ambiguity, and call it a subterm of $\mathcal{M}$.

2.1.2 Measures of stratification

The notion of *depth* in proof-nets of light logics is crucial in order to obtain a bound on the cut-elimination procedure. Also in elementary λ-calculus we need some measure of stratification, in order to study how the computation evolves over different strata.
Let $\mathcal{M}$ be a term, and let $\mathcal{C}$ be a simple context such that $\mathcal{M} = \mathcal{C}[\mathcal{N}]$ for some term $\mathcal{N}$:

Definition 4 (Depth of a simple context)**.** *The depth of a simple context $\mathcal{C}$, denoted by $\delta(\mathcal{C})$, is defined by induction on $\mathcal{C}$ as follows:*

- $\delta(\Box) = 0$;
- $\delta(\lambda x.\mathcal{C}') = \delta(\mathcal{C}')$;
- $\delta(\lambda^! x.\mathcal{C}') = \delta(\mathcal{C}')$;

- $\delta(\mathcal{C}'\mathcal{Q}) = \delta(\mathcal{C}')$;

- $\delta(\mathcal{P}\mathcal{C}') = \delta(\mathcal{C}')$;

- $\delta(!\mathcal{C}') = \delta(\mathcal{C}') + 1$;

Observe that, intuitively, the depth of an occurrence $\mathcal{C}$ is the number of !s enclosing the hole of the simple context $\mathcal{C}$.

Such notion of depth can be generalized to the case of a multi-hole context:

Definition 5 (Depth of a multi-hole context)**.**

- $\mathcal{C}[\_]_{i_1}...[\_]_{i_n}$ *denotes a context having* $n \geq 0$ *holes, the* j*-th hole being at depth* $i_j \in \mathbb{N}$*; moreover,* $\mathcal{C}[\mathcal{N}_1]_{i_1}...[\mathcal{N}_n]_{i_n}$ *is the term obtained by plugging the term* $\mathcal{N}_j$ *into the* j*-th hole of the context* $(1 \leq j \leq n)$*.*

- $\mathcal{C}\lfloor \_, ..., \_ \rfloor_i$*, where the symbol* _ *occurs* $n \geq 0$ *times, denotes a context having exactly* n *subterms at depth* $i \in \mathbb{N}$*, all of which are holes; moreover,* $\mathcal{C}\lfloor \mathcal{N}_1, ..., \mathcal{N}_n \rfloor_i$ *is the term obtained by filling the context* $\mathcal{C}\lfloor \_, ..., \_ \rfloor_i$ *with the terms* $\mathcal{N}_1, ..., \mathcal{N}_n$*, where holes are intended to be ordered from left to right.*

Observe that, given a term $\mathcal{M}$ and an integer i, there is a unique context $\mathcal{C}$ such that $\mathcal{M} = \mathcal{C}\lfloor \mathcal{N}_1, ..., \mathcal{N}_n \rfloor_i$.

Example 1. *Let* $\mathcal{M} = (\lambda^! x.y!x!(\lambda^! z.!z))!I$*, where* $I = \lambda x.x$*: then* $\mathcal{M} = \mathcal{C}\lfloor x, \lambda^! z.!z, I \rfloor_1$*, where* $\mathcal{C} = (\lambda^! x.y!\Box!\Box)!\Box$*.*
Note that any other representation of $\mathcal{M}$ *through a context of depth* 1 *does not satisfy the constraints of the definition.*

Informally, we often write that a term $\mathcal{N}$ *is at depth* i in $\mathcal{M}$, for some $i \geq 0$, whenever there is a context $\mathcal{C}$ such that $\mathcal{M} = \mathcal{C}[\mathcal{N}]_i$. The notion of depth of a term comes directly from the definition of depth for a proof-net:

Definition 6 (Depth of a term)**.** *The depth of a term* $\mathcal{M}$*, denoted by* $\delta(\mathcal{M})$*, is defined by induction as follows:*

- $\delta(x) = 0$;

- $\delta(\lambda x.\mathcal{P}) = \delta(\mathcal{P})$;

- $\delta(\lambda^! x.\mathcal{P}) = \delta(\mathcal{P})$;

- $\delta(\mathcal{P}\mathcal{Q}) = \max\{\delta(\mathcal{P}), \delta(\mathcal{Q})\}$;

- $\delta(!\mathcal{P}) = \delta(\mathcal{P}) + 1$.

The depth of a term $\mathcal{M}$ can also be thought of as the maximal nesting of !s in $\mathcal{M}$, that is, the maximum depth of all simple contexts $\mathcal{C}$ such that $\mathcal{M} = \mathcal{C}[\mathcal{N}]$ for some subterm $\mathcal{N}$ of $\mathcal{M}$.

Example 2. *Let* $\mathcal{M} = !((\lambda x.x)\ !!y\ !y)$*: then we have* $\delta(!((\lambda x.x)\ !!\Box\ !y)) = 3$ *and* $\delta(!((\lambda x.x)\ !!y\ !\Box)) = 2$*; moreover,* $\delta(\mathcal{M}) = 3$*.*

2.1.3 Dynamics of elementary λ-calculus

The reduction $\to$ is the contextual closure of the following rewriting rules, which are naturally induced by the two notions of abstraction in the language:

$$(\lambda x.\mathcal{M})\mathcal{N} \longrightarrow \mathcal{M}[\mathcal{N}/x] \quad (\beta\text{-redex}) \qquad (\lambda^! x.\mathcal{M})!\mathcal{N} \longrightarrow \mathcal{M}[\mathcal{N}/x] \quad (!\text{-redex})$$

Observe that a term of the shape $(\lambda^! x.\mathcal{M})\mathcal{P}$ is a redex only if $\mathcal{P} = !\mathcal{N}$ for some term $\mathcal{N}$; otherwise such application, also called a *block*, cannot be reduced.
A redex $(\lambda x.\mathcal{M})\mathcal{N}$ or $(\lambda^! x.\mathcal{M})!\mathcal{N}$ is *erasing* if $x \notin \mathrm{FV}(\mathcal{M})$

The intuition underlying these two kinds of redexes is that a λ-abstraction expects an input at depth 0, while a $\lambda^!$-abstraction expects an input at depth 1. Therefore, while the former reduction may be fired regardless of the shape of its argument, for the latter it is required that the argument is (at least) at depth 1.

If $\mathcal{R}$ is a *redex at depth* i in $\mathcal{M}$, then there is a simple context $\mathcal{C}$ such that $\mathcal{M} = \mathcal{C}[\mathcal{R}]_i$ and $\delta(\mathcal{C}) = i$, where $\mathcal{R}$ is either a β or !-redex. Intuitively, $\to_i$ denotes a reduction of a redex at depth i; the notation $\overset{*}{\to}$ ($\overset{*}{\to_i}$) stands for the reflexive and transitive closure of $\to$ ($\to_i$).

We denote by $\mathbf{nf}_i$ the set of terms in *i-normal form*:

Definition 7 (i-normal form)**.** *The set* $\mathbf{nf}_i$*, for* $i \in \mathbb{N}$*, is defined by induction on* i *as follows:*

- *for* $i = 0$
  - $x \in \mathbf{nf}_0$ *for every* $x \in \mathtt{Var}$*;*
  - *for every* $\mathcal{N} \in \Lambda^!$*, if* $\mathcal{N} \in \mathbf{nf}_0$ *then* $\lambda x.\mathcal{N} \in \mathbf{nf}_0$*;*
  - *for every* $\mathcal{N} \in \Lambda^!$*, if* $\mathcal{N} \in \mathbf{nf}_0$ *then* $\lambda^! x.\mathcal{N} \in \mathbf{nf}_0$*;*
  - *for every* $\{\mathcal{P}_1, ..., \mathcal{P}_n\} \subset \Lambda^!$ *and* $n \geq 1$*, if* $\mathcal{P}_i \in \mathbf{nf}_0$ $(1 \leq i \leq n)$ *then* $x\mathcal{P}_1...\mathcal{P}_n \in \mathbf{nf}_0$*;*
  - *for every* $\{\mathcal{P}, \mathcal{N}\} \subset \Lambda^!$ *such that* $\mathcal{N} \neq !\mathcal{Q}$*, if* $\mathcal{P} \in \mathbf{nf}_0$ *and* $\mathcal{N} \in \mathbf{nf}_0$ *then* $(\lambda^! x.\mathcal{P})\mathcal{N} \in \mathbf{nf}_0$*;*
  - $!\mathcal{N} \in \mathbf{nf}_0$.
- *for* $i > 0$*, for every* $\mathcal{M} \in \Lambda^!$ *such that* $\mathcal{M} = \mathcal{C}\lfloor \mathcal{N}_1, ..., \mathcal{N}_n \rfloor_i$*, if* $\mathcal{N}_i \in \mathbf{nf}_0$ $(1 \leq i \leq n)$ *and* $\mathcal{M} \in \mathbf{nf}_{i-1}$ *then* $\mathcal{M} \in \mathbf{nf}_i$*.*

Informally, a term is in i-normal form if and only if it does not contain any redex at depth less than or equal to i. As a consequence, $\mathcal{M}$ is in normal form if and only if it is in $\delta(\mathcal{M})$-normal form.

Among the terms of elementary λ-calculus, we restrict our attention to a specific subclass of terms which is inspired by elementary linear logic (ELL):

Definition 8 (Well-formed term)**.** *A term $\mathcal{M}$ is well-formed if and only if:*

- *for any subterm $\lambda x.\mathcal{N}$ of $\mathcal{M}$, x occurs at most once and only at depth 0 in $\mathcal{N}$;*
- *for any subterm $\lambda^! x.\mathcal{N}$ of $\mathcal{M}$, x occurs any number of times and only at depth 1 in $\mathcal{N}$.*

Note that the abstraction λ is a (affine) linear one.

Consider a well-formed term $\lambda x.\mathcal{M}$; by Definition 5, if $x \in \mathrm{FV}(\mathcal{M})$ then there is a context $\mathcal{C}$ such that either $\mathcal{M} = \mathcal{C}[x]_0$ or $\mathcal{M} = \mathcal{C}\lfloor x, ..., x \rfloor_1$, where x occurs $n \geq 1$ times in the latter. Moreover, in the first case $\mathcal{M}[\mathcal{N}/x] = \mathcal{C}[\mathcal{N}]_0$, while in the second case $\mathcal{M}[\mathcal{N}/x] = \mathcal{C}\lfloor \mathcal{N}, ..., \mathcal{N} \rfloor_1$, for every term $\mathcal{N}$.

The motivation behind the definition of well-formed terms is that the depth of a subterm in a well-formed term does not change after reduction, even though the subterm itself might be duplicated: therefore, since a λ-abstraction expects an input at depth 0 and its bound variable is at depth 0, the substitutions occur at depth 0; on the contrary, a $\lambda^!$-abstraction expects an input at depth 1 and its bound variable occurs at depth 1, so the substitutions occur at depth 1.

Example 3.

- *$\lambda^! x.x$ is not well-formed, because x is bound by a λ and it occurs at depth 0; indeed this term corresponds to the linear logic principle* $!\mathtt{A} \multimap \mathtt{A}$ *(dereliction), which is not valid in* ELL.
- *$\lambda^! x.!!x$ is not well-formed, because x is bound by a $\lambda^!$ and it occurs at depth 2: again, this term corresponds to the principle* $!\mathtt{A} \multimap !!\mathtt{A}$ *(digging), which is not valid in* ELL.
- *$\lambda x.!x$ is not well-formed, because x is bound by a λ and it occurs at depth 1: this term corresponds to the principle* $\mathtt{A} \multimap !\mathtt{A}$*, which holds neither in linear logic nor in* ELL.
- *$\lambda f.\lambda x.f(fx)$ is not well-formed, because f is bound by a λ and it occurs twice; this term represents the Church integer 2 in ordinary λ-calculus.*
- *$\lambda^! f.!(\lambda x.f(fx))$ is well-formed: note that it is obtained from the term above by replacing the λ binding f by a $\lambda^!$ and by adding a ! in such a way that the occurrences of f are at depth 1.*

The class of well-formed terms is preserved by reduction:

Lemma 1. *If $\mathcal{M}$ is a well-formed term and $\mathcal{M} \to \mathcal{M}'$, then $\mathcal{M}'$ is a well-formed term; moreover, the depth of a subterm occurring both in $\mathcal{M}$ and $\mathcal{M}'$ does not change.*

Proof. Let $\mathcal{M} = \mathcal{C}[\mathcal{R}]$, where $\mathcal{R}$ is either a β-redex or a !-redex; the proof proceeds by induction on $\mathcal{C}$.

Let $\mathcal{C} = \Box$. If $\mathcal{M} = (\lambda x.\mathcal{P})\mathcal{Q}$, the well-formedness condition on $\mathcal{M}$ implies that x occurs at most once at depth 0 in $\mathcal{P}$. Let us consider the non-trivial case $\mathcal{P} = \mathcal{C}'[x]_0$: then the contractum $\mathcal{M}' = \mathcal{P}[\mathcal{Q}/x] = \mathcal{C}'[\mathcal{Q}]_0$ is such that $\mathcal{Q}$ is at depth 0, and so the depth of any of its subterms does not change; otherwise, if $\mathcal{M} = (\lambda^! x.\mathcal{P})!\mathcal{Q}$, then x occurs (possibly) more than once in $\mathcal{P}$ at depth 1. Let us consider the interesting case $\mathcal{P} = \mathcal{C}'\lfloor x, ..., x\rfloor_1$: then the contractum $\mathcal{M}' = \mathcal{P}[\mathcal{Q}/x] = \mathcal{C}'\lfloor \mathcal{Q}, ..., \mathcal{Q}\rfloor_1$ is such that every copy of $\mathcal{Q}$ is at depth 1, so the depth of its subterms does not change. In both cases, $\mathcal{P}$ and $\mathcal{Q}$ being well-formed terms imply $\mathcal{M}'$ is also well-formed.

Now let $\mathcal{C} = \lambda y.\mathcal{C}'$, so $\mathcal{M} = \lambda y.\mathcal{C}'[\mathcal{R}]$. By induction hypothesis $\mathcal{C}'[\mathcal{R}] \to \mathcal{C}'[\mathcal{R}']$ implies $\mathcal{C}'[\mathcal{R}']$ is well-formed and the depth of the (possibly missing) occurrence of y in $\mathcal{C}'[\mathcal{R}]$ does not change after reduction: therefore $\lambda y.\mathcal{C}'[\mathcal{R}']$ is a well-formed term and the depth of its subterms does not change.

The case of $\mathcal{C} = \lambda^! y.\mathcal{C}'$ is similar; all the other cases follow easily by induction.

□

From this moment forward, we only consider the subclass of well-formed elementary λ-terms and we refer to it as $\lambda^!$-*calculus*.
Terms of $\lambda^!$-calculus enjoy a confluence property, whose proof is adapted from [RP04] by taking into account the notion of depth:

Property 1 (Confluence)**.**

i. Let $\mathcal{M} \to_i \mathcal{P}$ and $\mathcal{M} \to_i \mathcal{Q}$, then there is a term $\mathcal{N}$ such that $\mathcal{P} \overset{*}{\to}_i \mathcal{N}$ and $\mathcal{Q} \overset{*}{\to}_i \mathcal{N}$.

ii. Let $\mathcal{M} \to \mathcal{P}$ and $\mathcal{M} \to \mathcal{Q}$, then there is a term $\mathcal{N}$ such that $\mathcal{P} \overset{*}{\to} \mathcal{N}$ and $\mathcal{Q} \overset{*}{\to} \mathcal{N}$.

iii. Let $\mathcal{M} \in \mathbf{nf}_i$ and $\mathcal{M} \to_j \mathcal{M}'$, with $j \geq i + 1$, then $\mathcal{M}' \in \mathbf{nf}_i$.

Proof.

i. The proof is given in Appendix A.1.

ii. The proof follows easily from point (i) of the current Property.

iii. We want to show that, by reducing a redex $\mathcal{R}$ at depth j, no new redexes are created at lower depths. Let $\mathcal{M} = \mathcal{C}[\mathcal{R}]_j$ and $\mathcal{R} \to \mathcal{P}$: we proceed by induction on $\mathcal{C}$.

The case $\mathcal{C} = \Box$ is not possible, since $j > 0$.

Let $\mathcal{C} = \lambda x.\mathcal{C}'$, so $\mathcal{M} = \lambda x.\mathcal{C}'[\mathcal{R}]_j$. By inductive hypothesis $\mathcal{C}'[\mathcal{R}]_j \in \mathbf{nf}_i$ and $j \geq i+1$ imply $\mathcal{C}'[\mathcal{P}]_j \in \mathbf{nf}_i$: then $\mathcal{M}' = \lambda x.\mathcal{C}'[\mathcal{P}]_j \in \mathbf{nf}_i$.
The case of $\mathcal{C} = \lambda^! x.\mathcal{C}'$ is similar.

Let $\mathcal{C} = \mathcal{N}\mathcal{C}'$, so $\mathcal{M} = \mathcal{N}\mathcal{C}'[\mathcal{R}]_j$. By inductive hypothesis $\mathcal{C}'[\mathcal{R}]_j \in \mathbf{nf}_i$ and $j \geq i+1$ imply $\mathcal{C}'[\mathcal{P}]_j \in \mathbf{nf}_i$.
Consider the shape of $\mathcal{N} \in \mathbf{nf}_i$:

- if $\mathcal{N} = \lambda x.\mathcal{Q}$, then $\mathcal{M} = (\lambda x.\mathcal{Q})\mathcal{C}'[\mathcal{R}]_j$, thus contradicting the hypothesis that $\mathcal{M} \in \mathbf{nf}_i$;
- if $\mathcal{N} = \lambda^! x.\mathcal{Q}$, then $\mathcal{C}'[\mathcal{P}]_j \neq !\mathcal{P}'$: indeed $\mathcal{C}'[\mathcal{P}]_j = !\mathcal{P}'$ would imply $\mathcal{C}' = !\mathcal{C}''$, since $j \geq 1$, and so $\mathcal{M} = (\lambda^! x.\mathcal{Q})!\mathcal{C}''[\mathcal{R}]_{j-1}$, thus contradicting the hypothesis that $\mathcal{M} \in \mathbf{nf}_i$;
- in every other case, $\mathcal{M}' \in \mathbf{nf}_i$.

Let $\mathcal{C} = \mathcal{C}'\mathcal{N}$, so $\mathcal{M} = \mathcal{C}'[\mathcal{R}]_j\mathcal{N}$. By inductive hypothesis $\mathcal{C}'[\mathcal{R}]_j \in \mathbf{nf}_i$ and $j \geq i+1$ imply $\mathcal{C}'[\mathcal{P}]_j \in \mathbf{nf}_i$.
Consider the shape of $\mathcal{C}'[\mathcal{P}]_j \in \mathbf{nf}_i$:

- if $\mathcal{C}'[\mathcal{P}]_j = \lambda x.\mathcal{Q}$, then $\mathcal{C}' = \lambda x.\mathcal{C}''$, thus contradicting the hypothesis that $\mathcal{M} \in \mathbf{nf}_i$;
- if $\mathcal{C}'[\mathcal{P}]_j = \lambda^! x.\mathcal{Q}$, then $\mathcal{C}' = \lambda^! x.\mathcal{C}''$ and $\mathcal{N} \neq !\mathcal{N}'$: indeed $\mathcal{N} = !\mathcal{N}'$ would imply $\mathcal{M} = (\lambda^! x.\mathcal{C}''[\mathcal{R}]_j)!\mathcal{N}'$, thus contradicting the hypothesis that $\mathcal{M} \in \mathbf{nf}_i$;
- in every other case, $\mathcal{M}' \in \mathbf{nf}_i$.

Let $\mathcal{C} = !\mathcal{C}'$, so $\mathcal{M} = !\mathcal{C}'[\mathcal{R}]_{j-1}$. By inductive hypothesis $\mathcal{C}'[\mathcal{R}]_{j-1} \in \mathbf{nf}_{i-1}$ and $j \geq i+1$ imply $\mathcal{C}'[\mathcal{P}]_{j-1} \in \mathbf{nf}_{i-1}$: then $\mathcal{M}' = !\mathcal{C}'[\mathcal{P}]_{j-1} \in \mathbf{nf}_i$.

□

As a consequence of the definition of $\lambda^!$-terms, we also obtain that their depth does not increase during reduction:

Theorem 1. *If $\mathcal{M}$ is a $\lambda^!$-term and $\mathcal{M} \to \mathcal{M}'$, then $\delta(\mathcal{M}') \leq \delta(\mathcal{M})$.*

Proof. If $\mathcal{M} = \mathcal{C}[\mathcal{R}]$ and $\mathcal{R}$ is either $(\lambda x.\mathcal{P})\mathcal{Q}$ or $(\lambda^! x.\mathcal{P})!\mathcal{Q}$, where $x \notin \mathrm{FV}(\mathcal{P})$, then $\mathcal{M}' = \mathcal{C}[\mathcal{P}]$; so by definition $\delta(\mathcal{M}') \leq \delta(\mathcal{M})$. Otherwise $x \in \mathrm{FV}(\mathcal{P})$ and by

Lemma 1 the depth of any subterm occurring both in $\mathcal{M}$ and in $\mathcal{M}'$ is the same, so $\delta(\mathcal{M}') = \delta(\mathcal{M})$.

□

2.1.4 Measures and strategies of reduction

In order to study the reduction of a term, and thus its complexity, it is necessary to define a measure of terms which is also based on their depth:

Definition 9 (Size of a term)**.**

- *The size of $\mathcal{M}$ at depth i, denoted by $|\mathcal{M}|_i$, is defined by induction on $\mathcal{M}$ as follows:*
  - *If $\mathcal{M} = x$, then $|x|_0 = 1$ and $|x|_i = 0$ for every $i \geq 1$;*
  - *If $\mathcal{M} = \lambda x.\mathcal{N}$, then $|\mathcal{M}|_0 = |\mathcal{N}|_0 + 1$ and $|\mathcal{M}|_i = |\mathcal{N}|_i$ for every $i \geq 1$;*
  - *If $\mathcal{M} = \lambda^! x.\mathcal{N}$, then $|\mathcal{M}|_0 = |\mathcal{N}|_0 + 1$ and $|\mathcal{M}|_i = |\mathcal{N}|_i$ for every $i \geq 1$;*
  - *If $\mathcal{M} = \mathcal{N}\mathcal{P}$, then $|\mathcal{M}|_0 = |\mathcal{N}|_0 + |\mathcal{P}|_0 + 1$ and $|\mathcal{M}|_i = |\mathcal{N}|_i + |\mathcal{P}|_i$ for every $i \geq 1$;*
  - *If $\mathcal{M} = !\mathcal{N}$, then $|\mathcal{M}|_0 = 0$ and $|\mathcal{M}|_{i+1} = |\mathcal{N}|_i$ for every $i \geq 0$;*
- *Let $\delta(\mathcal{M}) = d$; then the size of $\mathcal{M}$ from depth i is $|\mathcal{M}|_{i+} = \sum_{j=i}^{d} |\mathcal{M}|_j$, while the size of $\mathcal{M}$ is $|\mathcal{M}| = \sum_{i=0}^{d} |\mathcal{M}|_i$.*

The definition above is extended naturally to contexts by imposing $|\Box|_i = 0$ for every $i \geq 0$. Observe that $|\mathcal{C}[\mathcal{N}_1]_{i_1}...[\mathcal{N}_n]_{i_n}|_j = |\mathcal{C}|_j + a_1 + ... + a_n$, where $a_k = 0$ if $j < i_k$, $a_k = |\mathcal{N}_k|_{j-i_k}$ otherwise, for $1 \leq k \leq n$: indeed we need to take into account the (possible) increase of depth of each term $\mathcal{N}_k$ when plugged into the k-th hole of the context.

It is interesting to examine how the size at all depths lower than or equal to i changes, after reducing a redex at depth i:

Lemma 2. *If $\mathcal{M} \to_i \mathcal{M}'$, then $|\mathcal{M}'|_i < |\mathcal{M}|_i$ and $|\mathcal{M}'|_j = |\mathcal{M}|_j$ for $j < i$.*

Proof. If $\mathcal{M} \to_i \mathcal{M}'$ by reducing an erasing redex, then the proof is trivial. Otherwise, let $\mathcal{M} = \mathcal{C}[\mathcal{R}]_i$, where $\mathcal{R}$ is either a β-redex or a !-redex: the proof is by induction on $\mathcal{C}$.

Let $\mathcal{C} = \Box$, so $i = 0$. If $\mathcal{M} = (\lambda x.\mathcal{P})\mathcal{Q}$, then $\mathcal{P} = \mathcal{C}'[x]_0$ by definition of well-formed term, so $\mathcal{M}' = \mathcal{C}'[\mathcal{Q}]_0$: therefore $|\mathcal{M}'|_0 = |\mathcal{C}'|_0 + |\mathcal{Q}|_0 < |\mathcal{C}'|_0 + |\mathcal{Q}|_0 + 3 = |\mathcal{M}|_0$. Otherwise $\mathcal{M} = (\lambda^! x.\mathcal{P})!\mathcal{Q}$ and $\mathcal{P} = \mathcal{C}'\lfloor x, ..., x\rfloor_1$ by definition of well-formed term, so $\mathcal{M}' = \mathcal{C}'\lfloor \mathcal{Q}, ..., \mathcal{Q}\rfloor_1$: then $|\mathcal{M}'|_0 = |\mathcal{C}'|_0 < |\mathcal{C}'|_0 + 2 = |\mathcal{M}|_0$.

Let $\mathcal{C} = \lambda y.\mathcal{C}'$, so $\mathcal{M} = \lambda y.\mathcal{N}$ and $\lambda y.\mathcal{N} \to_i \lambda y.\mathcal{N}' = \mathcal{M}'$. By inductive hypothesis $|\mathcal{N}'|_i < |\mathcal{N}|_i$ and $|\mathcal{N}'|_j = |\mathcal{N}|_j$ for $j < i$. If $i = 0$, then $|\mathcal{M}'|_0 = 1 + |\mathcal{N}'|_0 < 1 + |\mathcal{N}|_0 = |\mathcal{M}|_0$;

otherwise $|\mathcal{M}'|_i = |\mathcal{N}'|_i < |\mathcal{N}|_i = |\mathcal{M}|_i$ and $|\mathcal{M}'|_0 = 1 + |\mathcal{N}'|_0 = 1 + |\mathcal{N}|_0 = |\mathcal{M}|_0$ if $j = 0$, while $|\mathcal{M}'|_j = |\mathcal{N}'|_j = |\mathcal{N}|_j = |\mathcal{M}|_j$ if $0 < j < i$.
The case of $\mathcal{C} = \lambda^!y.\mathcal{C}'$ is similar.

Let $\mathcal{C} = \mathcal{C}'\mathcal{Q}$, so $\mathcal{M} = \mathcal{N}\mathcal{Q}$ and $\mathcal{N}\mathcal{Q} \to_i \mathcal{N}'\mathcal{Q} = \mathcal{M}'$. By inductive hypothesis $|\mathcal{N}'|_i < |\mathcal{N}|_i$ and $|\mathcal{N}'|_j = |\mathcal{N}|_j$ for $j < i$. If $i = 0$, then $|\mathcal{M}'|_0 = 1 + |\mathcal{N}'|_0 + |\mathcal{Q}|_0 < 1 + |\mathcal{N}|_0 + |\mathcal{Q}|_0 = |\mathcal{M}|_0$; otherwise it is easy to see that $|\mathcal{M}'|_i = |\mathcal{N}'|_i + |\mathcal{Q}|_i < |\mathcal{N}|_i + |\mathcal{Q}|_i = |\mathcal{M}|_i$ and $|\mathcal{M}'|_0 = 1 + |\mathcal{N}'|_0 + |\mathcal{Q}|_0 = 1 + |\mathcal{N}|_0 + |\mathcal{Q}|_0 = |\mathcal{M}|_0$ if $j = 0$, while if if $0 < j < i$ then $|\mathcal{M}'|_j = |\mathcal{N}'|_j + |\mathcal{Q}|_j = |\mathcal{N}|_j + |\mathcal{Q}|_j = |\mathcal{M}|_j$.
The case of $\mathcal{C} = \mathcal{Q}\mathcal{C}'$ is similar.

Let $\mathcal{C} = !\mathcal{C}'$, so $\mathcal{M} = !\mathcal{N}$ and $!\mathcal{N} \to_i !\mathcal{N}' = \mathcal{M}'$ for $i > 0$; note that this implies $\mathcal{N} \to_{i-1} \mathcal{N}'$. By inductive hypohesis $|\mathcal{N}'|_{i-1} < |\mathcal{N}|_{i-1}$ and $|\mathcal{N}'|_j = |\mathcal{N}|_j$ for $j < i-1$: then $|\mathcal{M}'|_i = |\mathcal{N}'|_{i-1} < |\mathcal{N}|_{i-1} = |\mathcal{M}|_i$ and $|\mathcal{M}'|_j = |\mathcal{N}'|_{j-1} = |\mathcal{N}|_{j-1} = |\mathcal{M}|_j$ if $j < i - 1$.

□

By Prop. 1.iii, reducing a redex at a given depth does not create any redex at strictly lower depth. Such property of $\lambda^!$-calculus suggests that we consider the following non-deterministic *level-by-level reduction strategy*: if the term is not in normal form, then reduce (non deterministically) a redex at depth i, where $i \geq 0$ is the minimal depth such that $\mathcal{M} \notin \mathbf{nf}_i$.

A *level-by-level reduction sequence* is a reduction sequence following the level-by-level strategy. We say that a reduction sequence is *maximal* if either it is infinite or it finishes with a normal term.

Property 2. *Any reduction of a term $\mathcal{M}$ by the level-by-level strategy terminates.*

Proof. Let $\mathcal{M}$ be any $\lambda^!$-term and let $\delta = \delta(\mathcal{M})$. By Prop. 1 we know that, in order to reduce $\mathcal{M}$, it is sufficient to reduce it to a δ-normal form. To such aim, we show that any maximal level-by-level reduction sequence s of $\mathcal{M}$ contains an i-normal form, for any $i \leq \delta$: then the statement follows by choosing $i = \delta$.

If $i = 0$, then by Lemma 2 the number of reduction steps at depth 0 is bounded by $|\mathcal{M}|_0$.

Now let $i = k$ such that $0 < k < \delta$: consider a maximal level-by-level reduction sequence s, such that $\mathcal{M}_i$ is the first i-normal form reached by s. By Prop. 1.iii, all reduction steps in s after $\mathcal{M}_i$ occur at depth $j > i$; moreover, by Lemma 2 there are at most $|\mathcal{M}_i|_{i+1}$ reduction steps at depth $i+1$: therefore s reaches an $(i+1)$-normal form and so the statement holds for $i = k + 1$.

□

Observe that a maximal level-by-level reduction sequence of a $\lambda^!$-term $\mathcal{M}$ has the shape shown in Figure 2.1, where $\rightsquigarrow_i$ denotes one reduction step at depth i according

to the level-by-level strategy; we use simply $\rightsquigarrow$ when we do not refer to a particular depth. Note that, for all i, $\mathcal{M}^1_{i+1}$ belongs to $\mathbf{nf}_i$.

$$\mathcal{M}^1_0 \rightsquigarrow_0 ... \rightsquigarrow_0 \mathcal{M}^{n_0}_0 = \mathcal{M}^1_1 \rightsquigarrow_1 ... \rightsquigarrow_1 \mathcal{M}^{n_1}_1 ... \mathcal{M}^1_i \rightsquigarrow_i ... \rightsquigarrow_i \mathcal{M}^{n_i}_i = \mathcal{M}^1_{i+1} ... \rightsquigarrow_\delta \mathcal{M}^{n_\delta}_\delta = \mathcal{M}^1_{\delta+1}$$

Figure 2.1: General shape of a level-by-level reduction sequence.

In a particular case, namely in Corollary 2, we use a *leftmost* deterministic version of the level-by-level strategy, which proceeds at every level from left to right by taking into account the shape of the different redexes of $\lambda^!$-calculus: at every step, the leftmost subterm of the shape $\mathcal{M}\mathcal{N}$ is chosen, where $\mathcal{M}$ is an abstraction; if such application is a redex, then it is reduced, otherwise the application is a block, i.e. a term of the shape $(\lambda^! x.\mathcal{P})\mathcal{N}$ where $\mathcal{N} \neq !\mathcal{Q}$, so the strategy looks for the next redex in $\mathcal{N}$.
In practice, this corresponds to using the call-by-name discipline for β-redexes and the call-by-value discipline for !-redexes [RP04].

We denote by $\mathcal{M} \Longrightarrow \mathcal{N}$ the fact that $\mathcal{N}$ is obtained from $\mathcal{M}$ by performing one reduction step according to the leftmost-by-level strategy; all notations for $\rightarrow$ are extended to $\rightsquigarrow$ and $\Longrightarrow$ in a straightforward way.

2.1.5 Representation of functions and computation

Representation of functions

Since our primary aim is representing functions, we first need to encode data in order to represent input and output of such functions.

The representation of boolean values is given by the familiar encoding:

$$\begin{array}{rcl} \texttt{true} & = & \lambda x.\lambda y.x \\ \texttt{false} & = & \lambda x.\lambda y.y \end{array}$$

On the contrary, the representation of tally integers and binary words cannot be given by the usual Church encoding, since the terms corresponding to Church integers and words do not satisfy the well-formedness condition, that is, neither are terms of $\lambda^!$-calculus. In order to overcome such issue, we use the following encodings for Church integers and Church binary words:

$$\begin{array}{lcl} n \in \mathbb{N}, & \underline{n} & = \lambda^! f.!(\lambda x.f\ (f\ \ldots (f\ x)\ldots)) \\ w \in \{0,1\}^\star, \quad w = \langle i_1, \ldots, i_n \rangle, & \underline{w} & = \lambda^! f_0.\lambda^! f_1.!(\lambda x.f_{i_1}\ (f_{i_2}\ \ldots (f_{i_n}\ x)\ldots)) \end{array}$$

where, by abuse of notation, we also denote by $\underline{1}$ the term $\lambda^! f.!f$.

A last encoding of data is needed, in the form of Scott binary words:

$$\widehat{\epsilon} \stackrel{\text{def}}{=} \lambda f_0.\lambda f_1.\lambda x.x \qquad \widehat{0w} \stackrel{\text{def}}{=} \lambda f_0.\lambda f_1.\lambda x.f_0\widehat{w} \qquad \widehat{1w} \stackrel{\text{def}}{=} \lambda f_0.\lambda f_1.\lambda x.f_1\widehat{w}$$

Observe that the terms encoding booleans and Scott binary words are of depth 0, while those representing Church integers and Church binary words are of depth 1. We denote the length of a word $w \in \{0,1\}^\star$ by $\mathbf{length}(w)$.

Representation of the computation

Let $\mathcal{P}$ be a closed $\lambda^!$-term in normal form, which we call a *program*. We represent the computation of a program on a binary word $\underline{w}$, by considering applications of the form $\mathcal{P}!\underline{w}$, where the argument is at depth 1 because we want the program to be able to duplicate its input if needed. Concerning the shape of the result, since we want to allow computation at arbitrary depth, we require the output to be of the form $!^k\mathcal{D}$, where $k \in \mathbb{N}$ and $\mathcal{D}$ is one of the data representations above.

We thus say that a function $f : \{0,1\}^\star \to \{0,1\}^\star$ is represented by a program $\mathcal{P}$ if there exists $k \in \mathbb{N}$ such that, for any $w \in \{0,1\}^\star$, $\mathcal{P}!\underline{w} \stackrel{*}{\to} !^k\mathcal{D}$ where $\mathcal{D} = \widehat{f(w)}$. Note that here we are using the Church binary word representation for the input and the Scott binary word representation for the output; however, this definition can be adapted to functions with other domains and codomains: in particular the codomain can be simply $\{0,1\}$, which is represented by $\{\texttt{true}, \texttt{false}\}$.

2.1.6 Complexity of reduction

As introduced in the previous subsection, we are interested in studying the complexity of reduction of $\lambda^!$-terms of the shape $\mathcal{P}!\underline{w}$, where $\mathcal{P}$ is a program and w is a binary word fed to the program as input.
As far as the stratification is concerned, it is useful to analyze the complexity of the reduction of such terms to their k-normal form, that is by reducing up to depth k, for $k \in \mathbb{N}$.

Let us consider a level-by-level reduction sequence of $\mathcal{M} = \mathcal{P}!\underline{w}$, adopting the notations of Figure (2.1). By Lemma 2, the number of steps at depth i is bounded by $|\mathcal{M}_i^1|$ and there are $(d+1)$ total rounds, where $d = \delta(\mathcal{M})$; then, in order to bound the total number of steps, it is sufficient to bound $|\mathcal{M}_i^1|$ by means of $|\mathcal{M}|$.

The leftmost-by-level strategy

In Subsection 2.1.4 we introduced the leftmost-by-level strategy as a mean to prove a key lemma about complexity. Here we give a formal account of the leftmost-by-level strategy, starting with a notion of evaluation context:

Definition 10 (Stratified evaluation context)**.** *The stratified evaluation context of $\mathcal{M}$ at depth i, for $i \leq \delta(\mathcal{M})$, is denoted by $\mathcal{E}_i^{\mathcal{M}}$ and it is a simple context defined by induction on i:*

- *$\mathcal{E}_0^{\mathcal{M}}$ is defined inductively as*

$$\begin{array}{ll}
\boxdot & \text{if } \mathcal{M} = (\lambda x.\mathcal{P})\mathcal{Q} \text{ or } \mathcal{M} = (\lambda^! x.\mathcal{P})!\mathcal{Q} \\
(\lambda^! x.\mathcal{P})\mathcal{E}_0^{\mathcal{Q}} & \text{if } \mathcal{M} = (\lambda^! x.\mathcal{P})\mathcal{Q} \text{ and } \mathcal{Q} \neq !\mathcal{Q}' \\
(\lambda^! x.\mathcal{E}_0^{\mathcal{P}})\mathcal{Q} & \text{if } \mathcal{M} = (\lambda^! x.\mathcal{P})\mathcal{Q}, \mathcal{Q} \in \mathbf{nf}_0 \text{ and } \mathcal{Q} \neq !\mathcal{Q}' \\
\lambda x.\mathcal{E}_0^{\mathcal{N}} & \text{if } \mathcal{M} = \lambda x.\mathcal{N} \\
\lambda^! x.\mathcal{E}_0^{\mathcal{N}} & \text{if } \mathcal{M} = \lambda^! x.\mathcal{N} \\
\mathcal{N}\mathcal{E}_0^{\mathcal{Q}} & \text{if } \mathcal{M} = \mathcal{N}\mathcal{Q}, \mathcal{N} \text{ is not an abstraction and } \mathcal{N} \in \mathbf{nf}_0 \\
\mathcal{E}_0^{\mathcal{N}}\mathcal{Q} & \text{if } \mathcal{M} = \mathcal{N}\mathcal{Q} \text{ and } \mathcal{N} \text{ is not an abstraction} \\
\textit{undefined} & \text{in any other case}
\end{array}$$

- *Let $\mathcal{M} = \mathcal{C}\lfloor \mathcal{N}_1, ..., \mathcal{N}_n \rfloor_i$, and let $\mathcal{N}_k$ $(1 \leq k \leq n)$ be the leftmost subterm of $\mathcal{M}$ such that $\mathcal{N}_k \notin \mathbf{nf}_i$: then $\mathcal{E}_i^{\mathcal{M}} = \mathcal{E}_0^{\mathcal{N}_k}$*

The evaluation context of a term is defined accordingly as follows:

Definition 11 (Evaluation context)**.** *The evaluation context of a term $\mathcal{M}$ is:*

$$\mathcal{E}^{\mathcal{M}} = \begin{cases} \mathcal{E}_i^{\mathcal{M}} & \text{where } i \text{ is the least } i \text{ such that } \mathcal{E}_i^{\mathcal{M}} \text{ is defined, if any;} \\ \textit{undefined} & \text{otherwise.} \end{cases}$$

We are now able to give a formal definition of the leftmost-by-level strategy $\Longrightarrow$, that is, $\mathcal{M} \Longrightarrow \mathcal{M}'$ if there are an evaluation context $\mathcal{E}^{\mathcal{M}}$, a redex $\mathcal{N}$ and a term $\mathcal{N}'$ such that $\mathcal{M} = \mathcal{E}^{\mathcal{M}}[\mathcal{N}]$, $\mathcal{M}' = \mathcal{E}^{\mathcal{M}}[\mathcal{N}']$ and $\mathcal{N} \to \mathcal{N}'$.

In order to find a sharp bound for the increase of size at depth greater than or equal to i, we need to identify a measure at depth i which decreases with every reduction at depth $i-1$, for every $i > 0$; clearly the size at depth i as given in Definition 9 does not have such a property, since a !-redex can make the size at depth i grow quadratically.

Let $\mathcal{M} \Longrightarrow_i \mathcal{M}'$, which represents the reduction of a redex at depth i following the leftmost-by-level strategy. The measure satisfying the previous constraint is the maximum number of potential duplications of a subterm at depth i during the reduction of $\mathcal{M}$, called *active points*, which are denoted by $\lceil \mathcal{M} \rceil_i$. Note that such measure is *dynamic*, in the sense that it accounts for the number of occurrences of variables bound by a $\lambda^!$-abstraction which could be replaced during the reduction: therefore, this measure depends strictly on the evaluation strategy.

The formal notions of $\lambda^!$-bound occurrences and active points follow:

Definition 12 ($\lambda^!$-bound occurrences)**.** *The maximum number of $\lambda^!$-bound occurrences at depth i in $\mathcal{M}$, denoted by $\mathtt{o}_i(\mathcal{M})$, for $i > 0$, is defined inductively as follows:*

- *If $i = 1$, then*
  - *$\mathcal{M} = x$ implies $\mathtt{o}_1(\mathcal{M}) = 0$;*
  - *$\mathcal{M} = \lambda x.\mathcal{P}$ implies $\mathtt{o}_1(\mathcal{M}) = \mathtt{o}_1(\mathcal{P})$;*
  - *$\mathcal{M} = \lambda^! x.\mathcal{P}$ implies $\mathtt{o}_1(\mathcal{M}) = \max\{n_0(x, \mathcal{P}), \mathtt{o}_1(\mathcal{P})\}$;*
  - *$\mathcal{M} = \mathcal{N}\mathcal{P}$ implies $\mathtt{o}_1(\mathcal{M}) = \max\{\mathtt{o}_1(\mathcal{N}), \mathtt{o}_1(\mathcal{P})\}$;*
  - *$\mathcal{M} = !\mathcal{N}$ implies $\mathtt{o}_1(\mathcal{M}) = 0$*
- *If $i > 1$, then $\mathcal{M} = \mathcal{C}\lfloor\mathcal{N}_1, ..., \mathcal{N}_m\rfloor_{i-1}$ implies $\mathtt{o}_i(\mathcal{M}) = \max_{j=1}^{m}\ \mathtt{o}_1(\mathcal{N}_j)$.*

Definition 13 (Active points)**.** *The number of active points of $\mathcal{M}$ at depth i, denoted by $\lceil\mathcal{M}\rceil_i$, is defined by induction on $\mathcal{E}^{\mathcal{M}}$ as follows:*

- *If $\mathcal{E}^{\mathcal{M}} = \mathcal{E}_0^{\mathcal{M}}$, then*
  - *$\mathcal{E}_0^{\mathcal{M}} = \boxdot$ implies $\lceil\mathcal{M}\rceil_1 = \mathtt{o}_1(\mathcal{M})$;*
  - *$\mathcal{E}_0^{\mathcal{M}} = \lambda x.\mathcal{E}_0^{\mathcal{N}}$ implies $\lceil\mathcal{M}\rceil_1 = \lceil\mathcal{N}\rceil_1$;*
  - *$\mathcal{E}_0^{\mathcal{M}} = \lambda^! x.\mathcal{E}_0^{\mathcal{N}}$ implies $\lceil\mathcal{M}\rceil_1 = \lceil\mathcal{N}\rceil_1$;*
  - *$\mathcal{E}_0^{\mathcal{M}} = \mathcal{P}\mathcal{E}_0^{\mathcal{N}}$ implies $\lceil\mathcal{M}\rceil_1 = \lceil\mathcal{N}\rceil_1$;*
  - *$\mathcal{E}_0^{\mathcal{M}} = \mathcal{E}_0^{\mathcal{P}}\mathcal{N}$ implies $\lceil\mathcal{M}\rceil_1 = \max\{\lceil\mathcal{P}\rceil_1, \mathtt{o}_1(\mathcal{N})\}$;*
  - *$\mathcal{E}_0^{\mathcal{M}} = (\lambda^! x.\mathcal{P})\mathcal{E}_0^{\mathcal{N}}$ implies $\lceil\mathcal{M}\rceil_1 = \max\{\mathtt{o}_1(\lambda^! x.\mathcal{P}), \lceil\mathcal{N}\rceil_1\}$;*
  - *$\mathcal{E}_0^{\mathcal{M}} = (\lambda^! x.\mathcal{E}_0^{\mathcal{P}})\mathcal{N}$ implies $\lceil\mathcal{M}\rceil_1 = \lceil\mathcal{P}\rceil_1$.*
- *If $\mathcal{E}^{\mathcal{M}} = \mathcal{E}_i^{\mathcal{M}}$ for some $i > 0$ and $\mathcal{M} = \mathcal{C}\lfloor\mathcal{N}_1, ..., \mathcal{N}_m\rfloor_i$, then $\lceil\mathcal{M}\rceil_{i+1} = \max_{j=1}^{m}\lceil\mathcal{N}_j\rceil_1$.*
- *If $\mathcal{E}_i^{\mathcal{M}}$ is undefined for some $i \geq 0$, then $\lceil\mathcal{M}\rceil_{i+1} = 0$.*

Observe that the number of active points is undefined at depth 0: indeed there are no active points at depth 0, because all variables bound by a $\lambda^!$-abstraction occur at depth greater than 0. Moreover, it is easy to see that $\lceil\mathcal{M}\rceil_1 \leq |\mathcal{M}|_1$ for any $\mathcal{M}$, since the number of $\lambda^!$-bound occurrences is always bounded by the total number of occurrences of variables in $\mathcal{M}$. Furthermore, $\lceil\mathcal{M}\rceil_1 \leq |\mathcal{M}|_1 \leq |\mathcal{M}|$
Note that, when $\mathcal{E}_0^{\mathcal{M}} = (\lambda^! x.\mathcal{P})\mathcal{E}_0^{\mathcal{N}}$, it means that at this point of the computation we don't know yet whether $\mathcal{M}$ is a !-redex or a block; therefore we take into consideration the number of active points of $\mathcal{N}$, which will later be discarded in case $\mathcal{M}$ proves to be a block.

Example 4. *Consider the term* $\mathcal{M} = \lambda^!z.(\lambda^!y.(\lambda^!x.!(zxx))!(zyy))!z$*: then* $\mathtt{o}_1(\mathcal{M}) = 3$*, while* $\mathcal{E}_0^{\mathcal{M}} = \lambda^!z.\boxdot$ *and* $\lceil\mathcal{M}\rceil_1 = 2$*.*

Our main goal now is to show that the number of active points does not increase during a reduction. To do so, we first prove that the number of active points of a term $\mathcal{M}$ at depth i is bounded by $\mathtt{o}_i(\mathcal{M})$; moreover, reducing a redex $\mathcal{M}$ to $\mathcal{M}'$ implies that the number of $\lambda^!$-bound occurrences does not increase:

Lemma 3.

i. If x *occurs at most once and at depth* 0 *in* $\mathcal{P}$*, then* $\mathtt{o}_1(\mathcal{P}[\mathcal{Q}/x]) \leq \max\{\mathtt{o}_1(\mathcal{P}), \mathtt{o}_1(\mathcal{Q})\}$*; if* x *occurs at depth* 1 *in* $\mathcal{P}$*, then* $\mathtt{o}_1(\mathcal{P}[\mathcal{Q}/x]) \leq \mathtt{o}_1(\mathcal{P})$*.*

ii. $\lceil\mathcal{M}\rceil_{i+1} \leq \mathtt{o}_{i+1}(\mathcal{M})$ *for every* $\mathcal{M}$*,* $i \geq 0$*.*

iii. Let $\mathcal{M} = (\lambda x.\mathcal{P})\mathcal{Q}$ *or* $\mathcal{M} = (\lambda^!x.\mathcal{P})!\mathcal{Q}$*: then* $\mathtt{o}_1(\mathcal{P}[\mathcal{Q}/x]) \leq \lceil\mathcal{M}\rceil_1$*.*

Proof.

i. Easy: in the first case, $\mathcal{Q}$ is copied only once and so the number of $\lambda^!$-bound occurrences is left untouched; in the second case $\mathcal{Q}$ might be copied multiple times, but since all $\lambda^!$-bound occurrences of $\mathcal{Q}$ are at depth greater than 1, their number does not count at depth 1

ii. By induction on $\mathcal{E}^{\mathcal{M}}$.

If $\mathcal{E}^{\mathcal{M}} = \mathcal{E}_0^{\mathcal{M}}$, then:

- $\mathcal{E}_0^{\mathcal{M}} = \boxdot$ implies $\lceil\mathcal{M}\rceil_1 = \mathtt{o}_1(\mathcal{M})$, so the inequality is satisfied;
- $\mathcal{E}_0^{\mathcal{M}} = \lambda x.\mathcal{E}_0^{\mathcal{N}}$ implies $\lceil\mathcal{M}\rceil_1 = \lceil\mathcal{N}\rceil_1$: by inductive hypothesis $\lceil\mathcal{N}\rceil_1 \leq \mathtt{o}_1(\mathcal{N})$, so $\lceil\mathcal{M}\rceil_1 = \lceil\mathcal{N}\rceil_1 \leq \mathtt{o}_1(\mathcal{N}) = \mathtt{o}_1(\mathcal{M})$;
- $\mathcal{E}_0^{\mathcal{M}} = \lambda^!x.\mathcal{E}_0^{\mathcal{N}}$ implies $\lceil\mathcal{M}\rceil_1 = \lceil\mathcal{N}\rceil_1$: by inductive hypothesis $\lceil\mathcal{N}\rceil_1 \leq \mathtt{o}_1(\mathcal{N})$, so $\lceil\mathcal{M}\rceil_1 = \lceil\mathcal{N}\rceil_1 \leq \mathtt{o}_1(\mathcal{N}) = \mathtt{o}_1(\mathcal{M})$;
- $\mathcal{E}_0^{\mathcal{M}} = \mathcal{P}\mathcal{E}_0^{\mathcal{N}}$ implies $\lceil\mathcal{M}\rceil_1 = \lceil\mathcal{N}\rceil_1$: by inductive hypothesis $\lceil\mathcal{N}\rceil_1 \leq \mathtt{o}_1(\mathcal{N})$, so $\lceil\mathcal{M}\rceil_1 = \lceil\mathcal{N}\rceil_1 \leq \mathtt{o}_1(\mathcal{N}) \leq \mathtt{o}_1(\mathcal{M})$;
- $\mathcal{E}_0^{\mathcal{M}} = \mathcal{E}_0^{\mathcal{P}}\mathcal{N}$ implies $\lceil\mathcal{M}\rceil_1 = \max\{\lceil\mathcal{P}\rceil_1, \mathtt{o}_1(\mathcal{N})\}$: by inductive hypothesis $\lceil\mathcal{P}\rceil_1 \leq \mathtt{o}_1(\mathcal{P})$, so $\lceil\mathcal{M}\rceil_1 = \max\{\lceil\mathcal{P}\rceil_1, \mathtt{o}_1(\mathcal{N})\} \leq \max\{\mathtt{o}_1(\mathcal{P}), \mathtt{o}_1(\mathcal{N})\} = \mathtt{o}_1(\mathcal{M})$;
- $\mathcal{E}_0^{\mathcal{M}} = (\lambda^!x.\mathcal{P})\mathcal{E}_0^{\mathcal{N}}$ implies $\lceil\mathcal{M}\rceil_1 = \max\{\mathtt{o}_1(\lambda^!x.\mathcal{P}), \lceil\mathcal{N}\rceil_1\}$: by inductive hypothesis $\lceil\mathcal{N}\rceil_1 \leq \mathtt{o}_1(\mathcal{N})$, so $\lceil\mathcal{M}\rceil_1 = \max\{\mathtt{o}_1(\lambda^!x.\mathcal{P}), \lceil\mathcal{N}\rceil_1\} \leq \max\{\mathtt{o}_1(\lambda^!x.\mathcal{P}), \mathtt{o}_1(\mathcal{N})\} = \mathtt{o}_1(\mathcal{M})$;
- $\mathcal{E}_0^{\mathcal{M}} = (\lambda^!x.\mathcal{E}_0^{\mathcal{P}})\mathcal{N}$ implies $\lceil\mathcal{M}\rceil_1 = \lceil\mathcal{P}\rceil_1$: by inductive hypothesis $\lceil\mathcal{P}\rceil_1 \leq \mathtt{o}_1(\mathcal{P})$, so $\lceil\mathcal{M}\rceil_1 = \lceil\mathcal{P}\rceil_1 \leq \mathtt{o}_1(\mathcal{P}) \leq \mathtt{o}_1(\mathcal{M})$.

If $\mathcal{E}^{\mathcal{M}} = \mathcal{E}_i^{\mathcal{M}}$ and $\mathcal{M} = \mathcal{C}\lfloor \mathcal{N}_1, ..., \mathcal{N}_m \rfloor_i$, then $\lceil \mathcal{M} \rceil_{i+1} = \max_{j=1}^{m} \lceil \mathcal{N}_j \rceil_1$. By inductive hypothesis $\lceil \mathcal{N}_j \rceil_1 \leq \mathtt{o}_1(\mathcal{N}_j)$,for $1 \leq j \leq m$: then $\lceil \mathcal{M} \rceil_{i+1} = \max_{j=1}^{m} \lceil \mathcal{N}_j \rceil_1 \leq \max_{j=1}^{m} \mathtt{o}_1(\mathcal{N}_j) = \mathtt{o}_{i+1}(\mathcal{M})$.

If $\mathcal{E}_i^{\mathcal{M}}$ is undefined for some $i \geq 0$, then $\lceil \mathcal{M} \rceil_{i+1} = 0 \leq \mathtt{o}_{i+1}(\mathcal{M})$.

iii. Let $\mathcal{M} = (\lambda x.\mathcal{P})\mathcal{Q}$, so x can occur at most once and at depth 0 in $\mathcal{P}$. By point i of the current Lemma $\mathtt{o}_1(\mathcal{P}[\mathcal{Q}/x]) \leq \max\{\mathtt{o}_1(\mathcal{P}), \mathtt{o}_1(\mathcal{Q})\}$, so $\mathtt{o}_1(\mathcal{P}[\mathcal{Q}/x]) \leq \max\{\mathtt{o}_1(\mathcal{P}), \mathtt{o}_1(\mathcal{Q})\} = \mathtt{o}_1(\mathcal{M}) = \lceil \mathcal{M} \rceil_1$.

Now let $\mathcal{M} = (\lambda^! x.\mathcal{P})!\mathcal{Q}$, so x can occur any number of times and only at depth 1 in $\mathcal{P}$. By point i of the current Lemma $\mathtt{o}_1(\mathcal{P}[\mathcal{Q}/x]) \leq \mathtt{o}_1(\mathcal{P})$, so $\mathtt{o}_1(\mathcal{P}[\mathcal{Q}/x]) \leq \max \mathtt{o}_1(\mathcal{P}) \leq \mathtt{o}_1(\mathcal{M}) = \lceil \mathcal{M} \rceil_1$.

□

The previous results are of help in showing that the number of active points does not increase while reducing a term by the leftmost-by-level strategy:

Lemma 4. *$\mathcal{M} \Longrightarrow \mathcal{M}'$ implies $\lceil \mathcal{M}' \rceil_{i+1} \leq \lceil \mathcal{M} \rceil_{i+1}$, for every $i \geq 0$.*

Proof. We proceed by induction on $\mathcal{E}_0^{\mathcal{M}}$. Observe that if $\mathcal{M}' \in \mathbf{nf}_0$ then the proof is trivial, since $\lceil \mathcal{M}' \rceil_1 = 0 \leq \lceil \mathcal{M} \rceil_1$; therefore we consider all possible cases where $\mathcal{E}_0^{\mathcal{M}'}$ is defined.

Let $\mathcal{E}_0^{\mathcal{M}} = \boxdot$: then the proof follows by points ii and iii of Lemma 3.

Let $\mathcal{E}_0^{\mathcal{M}} = (\lambda^! x.\mathcal{P})\mathcal{E}_0^{\mathcal{N}}$, so $\mathcal{M} = (\lambda^! x.\mathcal{P})\mathcal{N}$ and $\mathcal{M}' = (\lambda^! x.\mathcal{P})\mathcal{N}'$ where $\mathcal{N} \Longrightarrow_0 \mathcal{N}'$. By inductive hypothesis, $\lceil \mathcal{N}' \rceil_1 \leq \lceil \mathcal{N} \rceil_1$. We must consider three cases:

- if $\mathcal{E}_0^{\mathcal{M}'} = (\lambda^! x.\mathcal{P})\mathcal{E}_0^{\mathcal{N}'}$, then the proof follows by induction;
- if $\mathcal{E}_0^{\mathcal{M}'} = (\lambda^! x.\mathcal{E}_0^{\mathcal{P}})\mathcal{N}'$, then by definition $\lceil \mathcal{M}' \rceil_1 = \lceil \mathcal{P} \rceil_1$: by Lemma 3.ii $\lceil \mathcal{P} \rceil_1 \leq \mathtt{o}_1(\mathcal{P})$, so $\lceil \mathcal{M}' \rceil_1 = \lceil \mathcal{P} \rceil_1 \leq \mathtt{o}_1(\mathcal{P}) \leq \max\{\mathtt{o}_1(\lambda^! x.\mathcal{P}), \lceil \mathcal{N} \rceil_1\} = \lceil \mathcal{M} \rceil_1$;
- if $\mathcal{E}_0^{\mathcal{M}'} = \boxdot$, then $\mathcal{N}' = !\mathcal{N}''$ for some term $\mathcal{N}''$ and $\mathcal{M}' = (\lambda^! x.\mathcal{P})!\mathcal{N}''$: by definition $\lceil \mathcal{M}' \rceil_1 = \mathtt{o}_1(\mathcal{M}') = \max\{\mathtt{o}_1(\lambda^! x.\mathcal{P}), \mathtt{o}_1(!\mathcal{N}'')\} = \mathtt{o}_1(\lambda^! x.\mathcal{P}) \leq \max\{\mathtt{o}_1(\lambda^! x.\mathcal{P}), \lceil \mathcal{N} \rceil_1\} = \lceil \mathcal{M} \rceil_1$;

All other cases follow easily by induction. Moreover, the property can be easily checked for $i > 0$ by using the definition of context.

□

Example 5. *Consider again the term $\mathcal{M} = \lambda^! z.(\lambda^! y.(\lambda^! x.!(zxx))!(zyy))!z$: since $\mathcal{E}_0^{\mathcal{M}} = \lambda^! z.\boxdot$, we have $\mathcal{M} \Longrightarrow \mathcal{M}' = \lambda^! z.(\lambda^! x.!(zxx))!(zzz)$ and $\lceil \mathcal{M} \rceil_1 = 2$. Moreover, $\mathcal{E}_0^{\mathcal{M}'} = \lambda^! z.\boxdot$, so $\lceil \mathcal{M}' \rceil_1 = 2$.*

As a further step, we study how the size of the whole term changes at depths higher than n, when performing a $\Longrightarrow_n$ step:

Lemma 5. *$\mathcal{M} \Longrightarrow_n \mathcal{N}$ implies $|\mathcal{N}|_i \leq \lceil \mathcal{M} \rceil_{n+1} \cdot |\mathcal{M}|_i + |\mathcal{M}|_i$ for every $i > n$.*

Proof. We proceed by induction on n, and then by induction on $\mathcal{E}_n^{\mathcal{M}}$. If $n = 0$, then we consider all possible cases of $\mathcal{E}_0^{\mathcal{M}}$.

- Let $\mathcal{E}_0^{\mathcal{M}} = \boxdot$, then there are two possibilities:

  i) if $\mathcal{M} = (\lambda x.\mathcal{P})\mathcal{Q}$, $\mathcal{N} = \mathcal{P}[\mathcal{Q}/x]$, then $|\mathcal{N}|_i \leq |\mathcal{M}|_i \leq \lceil \mathcal{M} \rceil_1 \cdot |\mathcal{M}|_i + |\mathcal{M}|_i$;

  ii) if $\mathcal{M} = (\lambda^! x.\mathcal{P})!\mathcal{Q}$, $\mathcal{N} = \mathcal{P}[\mathcal{Q}/x]$, then $|\mathcal{N}|_i \leq |\mathcal{M}|_i \cdot n_0(x, \mathcal{P}) + |\mathcal{M}|_i \leq |\mathcal{M}|_i \cdot \mathtt{o}_1(\mathcal{M}) + |\mathcal{M}|_i = \lceil \mathcal{M} \rceil_1 \cdot |\mathcal{M}|_i + |\mathcal{M}|_i$.

- Let $\mathcal{E}_0^{\mathcal{M}} = \lambda x.\mathcal{E}_0^{\mathcal{P}}$, then $\mathcal{M} = \lambda x.\mathcal{P}$ and $\mathcal{N} = \lambda x.\mathcal{P}'$, where $\mathcal{P} \Longrightarrow_0 \mathcal{P}'$ implies $|\mathcal{P}'|_i \leq \lceil \mathcal{P} \rceil_1 \cdot |\mathcal{P}|_i + |\mathcal{P}|_i$ for every $i \geq 0$ by inductive hypothesis; so by induction $|\mathcal{N}|_i = |\mathcal{P}'|_i \leq \lceil \mathcal{P} \rceil_1 \cdot |\mathcal{P}|_i + |\mathcal{P}|_i = \lceil \mathcal{M} \rceil_1 \cdot |\mathcal{M}|_i + |\mathcal{M}|_i$ for $i > 0$.

- The case $\mathcal{E}_0^{\mathcal{M}} = \lambda^! x.\mathcal{E}_0^{\mathcal{P}}$ is similar to the previous one.

- Let $\mathcal{E}_0^{\mathcal{M}} = \mathcal{P}\mathcal{E}_0^{\mathcal{Q}}$, then $\mathcal{M} = \mathcal{P}\mathcal{Q}$ and $\mathcal{N} = \mathcal{P}\mathcal{Q}'$, where $\mathcal{Q} \Longrightarrow_0 \mathcal{Q}'$ implies $|\mathcal{Q}'|_i \leq \lceil \mathcal{Q} \rceil_1 \cdot |\mathcal{Q}|_i + |\mathcal{Q}|_i$ for every $i \geq 0$ by inductive hypothesis; so, if by induction $|\mathcal{N}|_i = |\mathcal{P}|_i + |\mathcal{Q}'|_0 \leq \lceil \mathcal{Q} \rceil_1 \cdot |\mathcal{Q}|_i + |\mathcal{Q}|_i + |\mathcal{P}|_i = \lceil \mathcal{M} \rceil_1 \cdot |\mathcal{M}|_i + |\mathcal{M}|_i$ for $i > 0$.

- The case $\mathcal{E}_0^{\mathcal{M}} = (\lambda^! x.\mathcal{E}_0^{\mathcal{P}})\mathcal{Q}$ is similar to the previous one.

- Let $\mathcal{E}_0^{\mathcal{M}} = \mathcal{E}_0^{\mathcal{P}}\mathcal{Q}$, then $\mathcal{M} = \mathcal{P}\mathcal{Q}$ and $\mathcal{N} = \mathcal{P}'\mathcal{Q}$, where $\mathcal{P} \Longrightarrow_0 \mathcal{P}'$ implies $|\mathcal{P}'|_i \leq \lceil \mathcal{P} \rceil_1 \cdot |\mathcal{P}|_i + |\mathcal{P}|_i$ for every $i \geq 0$ by inductive hypothesis. By definition $\lceil \mathcal{M} \rceil_1 = \max\{\lceil \mathcal{P} \rceil_1, \mathtt{o}_1(\mathcal{Q})\}$, so $|\mathcal{N}|_i = |\mathcal{P}'|_i + |\mathcal{Q}|_0 \leq \lceil \mathcal{P} \rceil_1 \cdot |\mathcal{P}|_i + |\mathcal{P}|_i + |\mathcal{Q}|_i = \lceil \mathcal{M} \rceil_1 \cdot |\mathcal{M}|_i + |\mathcal{M}|_i$ for $i > 0$.

- The case $\mathcal{E}_0^{\mathcal{M}} = (\lambda^! x.\mathcal{P})\mathcal{E}_0^{\mathcal{Q}}$ follows by induction as the previous one.

If $n > 0$, let $\mathcal{M} = \mathcal{C}\lfloor \mathcal{N}_1, ..., \mathcal{N}_k, ..., \mathcal{N}_m \rfloor_n$ for some index k, m such that $\mathcal{N}_j \in \mathbf{nf}_0$ for $1 \leq i \leq k-1$, $\mathcal{N}_k \notin \mathbf{nf}_0$, so $\mathcal{E}_n^{\mathcal{M}} = \mathcal{E}_0^{\mathcal{N}_k}$ and $\mathcal{N} = \mathcal{C}\lfloor \mathcal{N}_1, ..., \mathcal{N}'_k, ..., \mathcal{N}_m \rfloor_n$. By inductive hypothesis $\mathcal{N}_k \Longrightarrow_0 \mathcal{N}'_k$ implies $|\mathcal{N}'_k|_i \leq \lceil \mathcal{N}_k \rceil_1 \cdot |\mathcal{N}_k|_i + |\mathcal{N}_k|_i$ for every $i > 0$. Moreover, by definition $\lceil \mathcal{M} \rceil_{n+1} = \max_{j=1}^{m} \lceil \mathcal{N}_j \rceil_1$: therefore

$$\begin{aligned} |\mathcal{N}|_i &= |\mathcal{C}|_i + |\mathcal{N}_1|_{i-n} + ... + |\mathcal{N}'_k|_{i-n} + ... + |\mathcal{N}_m|_{i-n} \\ &\leq |\mathcal{C}|_i + |\mathcal{N}_1|_{i-n} + ... + (\lceil \mathcal{N}_k \rceil_1 \cdot |\mathcal{N}_k|_{i-n} + |\mathcal{N}_k|_{i-n}) + ... + |\mathcal{N}_m|_{i-n} \\ &= \lceil \mathcal{N}_k \rceil_1 \cdot |\mathcal{N}_k|_{i-n} + |\mathcal{M}|_i \\ &\leq \lceil \mathcal{M} \rceil_{n+1} \cdot |\mathcal{M}|_i + |\mathcal{M}|_i. \end{aligned}$$

□

All these results allow us to easily obtain a sort of stratified version of the size-growth result when performing a reduction step at depth n, where the size of the reduced term at depth i is bounded by a function of the size of the initial term at depth i and of the number of its active points at depth $n + 1$:

Lemma 6. *$\mathcal{M} \overset{*}{\Longrightarrow}_n \mathcal{M}'$ in k reduction steps implies $|\mathcal{M}|'_i \leq |\mathcal{M}|_i \cdot (\lceil \mathcal{M} \rceil_{n+1} + 1)^k$, for every $i > n$.*

Proof. Let $\mathcal{M} = \mathcal{M}_1 \Longrightarrow_n \mathcal{M}_2 \Longrightarrow_n ... \Longrightarrow_n \mathcal{M}_k = \mathcal{M}'$: we proceed by induction on k.

Let $k = 2$, so $\mathcal{M} \Longrightarrow \mathcal{M}'$ and $\lceil \mathcal{M} \rceil_{n+1} = |\mathcal{M}|_{n+1}$: then $|\mathcal{M}|'_i \leq |\mathcal{M}|_i \cdot (\lceil \mathcal{M} \rceil_{n+1} + 1)$ by Lemma 5.

Now let $\mathcal{M} \overset{*}{\Longrightarrow}_n \mathcal{M}_h$ in $h-1$ reduction steps, for some $h > 2$. By inductive hypothesis $|\mathcal{M}_h|_i \leq |\mathcal{M}|_i \cdot (\lceil \mathcal{M} \rceil_{n+1} + 1)^{h-1}$, for every $i > n$. If $\mathcal{M}_h \Longrightarrow_n \mathcal{M}_{h+1}$, then

$$\begin{aligned}
|\mathcal{M}_{h+1}|_i &\leq |\mathcal{M}_h|_i \cdot (\lceil \mathcal{M}_h \rceil_{n+1} + 1) && \text{by Lemma 5} \\
&\leq |\mathcal{M}_h|_i \cdot (\lceil \mathcal{M} \rceil_{n+1} + 1) && \text{by Lemma 4} \\
&\leq |\mathcal{M}|_i \cdot (\lceil \mathcal{M} \rceil_{n+1} + 1)^{h-1} \cdot (\lceil \mathcal{M} \rceil_{n+1} + 1) && \text{by inductive hypothesis} \\
&= |\mathcal{M}|_i \cdot (\lceil \mathcal{M} \rceil_{n+1} + 1)^h.
\end{aligned}$$

□

The main result of this subsection, namely a bound of the growth in size for a reduction step at depth i, now follows easily:

Corollary 2 (Size-growth)**.** *If $\mathcal{M} \overset{*}{\Longrightarrow}_i \mathcal{M}'$ by c reduction steps, then $|\mathcal{M}'| \leq |\mathcal{M}| \cdot (|\mathcal{M}| + 1)^c$ $(0 \leq i \leq \delta(\mathcal{M}))$.*

Proof. As a consequence of Definition 13 we know that $\lceil \mathcal{M} \rceil_{i+1} \leq |\mathcal{M}|$. By Lemma 6, $|\mathcal{M}'|_j = |\mathcal{M}|_j$ for $j < i$, so $|\mathcal{M}'|_j \leq |\mathcal{M}|_j \cdot (\lceil \mathcal{M} \rceil_{i+1} + 1)^c$ also holds for $j \leq i$: then $|\mathcal{M}'| = \sum_{j=0}^{\delta(\mathcal{M}')} |\mathcal{M}'|_j \leq \sum_{j=0}^{\delta(\mathcal{M})} |\mathcal{M}|_j \cdot (\lceil \mathcal{M} \rceil_{i+1} + 1) \leq |\mathcal{M}| \cdot (\lceil \mathcal{M} \rceil_{i+1} + 1)^c$. □

Complexity bound

In the previous section we gave a bound on the reduction of a $\lambda^!$-term, using the leftmost-by-level strategy, with respect to the size of the initial term. Nevertheless, such dedicated strategy was necessary only in order to define the active points of a term; in the present section the result of Corollary 2 is then employed to prove a more general result based on the level-by-level reduction strategy.

Let 2_i^n be defined inductively as $2_0^x = x$ and $2_{i+1}^x = 2^{2_i^x}$. We employ the result of the previous subsection in order to prove a bound on the reduction of a $\lambda^!$-term up to a given depth:

Property 3. *Let $\mathcal{P}$ be a program; for any $k \geq 2$, there exists a polynomial q such that, for any $w \in \{0,1\}^\star$, $\mathcal{P}!\underline{w} \overset{*}{\rightsquigarrow} \mathcal{M}_k^1 \in \mathbf{nf}_{k-1}$ in at most $2_{k-2}^{q(n)}$ steps and $|\mathcal{M}_k^1| \leq 2_{k-2}^{q(n)}$, where $n = \mathbf{length}(w)$. In particular, if $k = 2$ then the bound is polynomial in n.*

Proof. We proceed by induction on k.

Without loss of generality, we can assume $\mathcal{P}$ to be a $\lambda^!$-abstraction of the form $\lambda^! y.\mathcal{Q}$; otherwise $\mathcal{P}!\underline{w}$ is a normal form and the property holds.

- Let $k = 2$ and consider a level-by-level reduction sequence of $\mathcal{M}_0^1 = \mathcal{P}!\underline{w}$: we examine round 0 and round 1 of the reduction sequence, following the notation of Figure 2.1.

  At round 0, the only reduction is $(\lambda^! y.\mathcal{Q})!\underline{w} \to \mathcal{Q}[\underline{w}/y] = \mathcal{M}_1^1$. Observe that $\mathcal{M}_1^1 \in \mathbf{nf}_0$ because the occurrences of y in $\mathcal{Q}$ are at depth 1.

  Let us now consider round 1. Let $b = n_0(y, \mathcal{Q})$, which is independent of n: then $|\mathcal{Q}[\underline{w}/y]|_1 \leq |\mathcal{Q}|_1 + b \cdot |\underline{w}|_0$ and, by definition of $\underline{w}$, $|\underline{w}|_0 = 2$. Therefore, $|\mathcal{M}_1^1|_1 = |\mathcal{Q}[\underline{w}/y]|_1 \leq |\mathcal{Q}|_1 + 2b$. Let c be the constant $|\mathcal{Q}|_1 + 2b$, which is again independent of n: by Lemma 2, the number of steps at the end of round 1 is bounded by c, so the part of the statement concerning the number of steps holds.
  Now let $\mathcal{M}_2^1 \in \mathbf{nf}_1$ be the term obtained at the end of round 1. By Property 1.i (confluence) $\mathcal{M}_1^1 \overset{*}{\Longrightarrow}_1 \mathcal{M}_2^1$ and by Lemma 2 such reduction is performed in c' steps, where $c' \leq |\mathcal{M}_1^1|_1 \leq c$; therefore $|\mathcal{M}_2^1| \leq |\mathcal{M}_1^1| \cdot (|\mathcal{M}_1^1| + 1)^c$ by Corollary 2. Moreover $|\mathcal{M}_1^1| \leq |\mathcal{Q}| + b|\underline{w}|$, so the size is polynomial in n.

- Let us assume the property holds for k: we prove that it holds also for $k + 1$.

  By inductive hypothesis, $\mathcal{M}$ reduces to $\mathcal{M}_k^1$ in at most $2_{k-2}^{q(n)}$ steps and $|\mathcal{M}_k^1| \leq 2_{k-2}^{q(n)}$. Let $\mathcal{M}_k^1 \overset{*}{\rightsquigarrow}_k \mathcal{M}_{k+1}^1 \in \mathbf{nf}_k$.
  By Lemma 2 the reduction sequence has at most $|\mathcal{M}_k^1|_k$ steps and $|\mathcal{M}_k^1|_k \leq |\mathcal{M}_k^1| \leq 2_{k-2}^{q(n)}$, therefore $\mathcal{M}$ reduces to $\mathcal{M}_{k+1}^1$ in at most $2 \cdot 2_{k-2}^{q(n)} \leq 2_{k-2}^{2q(n)}$ steps. Moreover, by Property 1.i (depth-wise confluence) $\mathcal{M}_k^1 \overset{*}{\Longrightarrow} \mathcal{M}_{k+1}^1$ and by Lemma 2 and Corollary 2 we obtain

$$\begin{aligned}
|\mathcal{M}_{k+1}^1| &\leq |\mathcal{M}_k^1| \cdot (|\mathcal{M}_k^1| + 1)^{2_{k-2}^{q(n)}} \\
&\leq 2_{k-2}^{q(n)} \cdot (2_{k-2}^{q(n)} + 1)^{2_{k-2}^{q(n)}} \\
&\leq 2_{k-2}^{q(n)} \cdot (2_{k-2}^{2q(n)})^{2_{k-2}^{q(n)}} \\
&\leq 2_{k-2}^{q(n)} \cdot 2^{2_{k-3}^{2q(n)} \cdot 2_{k-2}^{q(n)}} \\
&\leq 2_{k-2}^{q(n)} \cdot 2^{2_{k-2}^{3q(n)}} \\
&\leq 2_{k-1}^{q'(n)}
\end{aligned}$$

  for some polynomial $q'(n)$: therefore the statement holds for $k + 1$.

□

Approximations

From Property 3 we can easily derive a $2^{q(n)}_{k-2}$ bound on the number of *steps* of the reduction of $\mathcal{P}!\underline{w}$, not only to its $(k-1)$-normal form, but also to its k-normal form $\mathcal{M}^1_{k+1}$. Unfortunately, this does not yield directly a *time* bound $O(2^{q(n)}_{k-2})$ for the simulation of this reduction on a Turing machine: indeed, even though the size of the final term of round k fulfills the desired bound, the same does not hold for intermediate terms of round k, for which the size at depth $k+1$ can grow exponentially.

Nonetheless, since we are only interested in the result of the computation at depth k, the size of the subterms at depth $k+1$ is actually irrelevant. In order to overcome such issue we introduce a notion of *approximation*, which allows to compute up to a certain depth k while ignoring the higher strata of the term: such notion is inspired by the semantics of stratified coherence spaces [Bai04], where partial information is managed with the aim of obtaining more and more refinements of the actual computation.

In order to accomodate approximations, we extend the calculus with a constant $*$, whose sizes are $|*|_0 = 1$ and $|*|_{i+1} = 0$ for every $i \geq 0$. The definition is then extended to term contexts:

Definition 14 (i-th approximation)**.** *The i-th approximation of a context $\mathcal{C}$ for $i \in \mathbb{N}$, denoted by $\overline{\mathcal{C}}^i$, is defined by induction on $\mathcal{C}$ as follows:*

- $\overline{!\mathcal{C}}^0 = !*$;
- $\overline{!\mathcal{C}}^{i+1} = !\overline{\mathcal{C}}^i$;
- $\overline{x}^i = x$;
- $\overline{\Box}^i = \Box$;
- $\overline{\mathcal{C}\mathcal{C}'}^i = \overline{\mathcal{C}}^i\overline{\mathcal{C}'}^i$;
- $\overline{\lambda x.\mathcal{C}}^i = \lambda x.\overline{\mathcal{C}}^i$;
- $\overline{\lambda^! x.\mathcal{C}}^i = \lambda^! x.\overline{\mathcal{C}}^i$.

Note that $\overline{\mathcal{C}\lfloor\mathcal{N}_1 ... \mathcal{N}_n\rfloor_j}^i = \overline{\mathcal{C}}^i\lfloor\overline{\mathcal{N}_1}^{i-j} ... \overline{\mathcal{N}_n}^{i-j}\rfloor_j$ if $j \leq i$; otherwise $\overline{\mathcal{C}\lfloor\mathcal{N}_1 ... \mathcal{N}_n\rfloor_j}^i = \overline{\mathcal{C}}^i$, since all subterms filling the holes at depth $j > i$ do not count when the approximation at depth i is considered.

In practice, when considering the i-th approximation of a term $\mathcal{M}$, $\overline{\mathcal{M}}^i$ is obtained by replacing all subterms of $\mathcal{M}$ at depth $i+1$ by the constant $*$.

Example 6. *For every Church binary word $\underline{w}$, the approximations at depth* 0 *and $i+1$ are respectively $\overline{\underline{w}}^0 = \lambda^! f_0.\lambda^! f_1.!*$ and $\overline{\underline{w}}^{i+1} = \underline{w}$, for $i \geq 0$.*

First we examine the effect of approximation on substitutions, reductions and contexts respectively:

Lemma 7 (Approximation of substitution)**.**

i. If the occurrence of x in $\mathcal{M}$ is at depth 0, then $\overline{\mathcal{M}[\mathcal{N}/x]}^i = \overline{\mathcal{M}}^i[\overline{\mathcal{N}}^i/x]$.

ii. If all occurrences of x in $\mathcal{M}$ are at depth 1, then $\overline{\mathcal{M}[\mathcal{N}/x]}^0 = \overline{\mathcal{M}}^0$, $\overline{\mathcal{M}[\mathcal{N}/x]}^{i+1} = \overline{\mathcal{M}}^{i+1}[\overline{\mathcal{N}}^i/x]$, for $i \geq 0$.

Proof.

i. Let $\mathcal{M} = \mathcal{C}[x]_0$: then $\overline{\mathcal{M}[\mathcal{N}/x]}^i = \overline{\mathcal{C}[\mathcal{N}]_0}^i = \overline{\mathcal{C}}^i[\overline{\mathcal{N}}^i]_0 = \overline{\mathcal{M}}^i[\overline{\mathcal{N}}^i/x]$ for every $i \geq 0$.

ii. Let $\mathcal{M} = \mathcal{C}[x]_1...[x]_1$: then $\overline{\mathcal{M}[\mathcal{N}/x]}^0 = \overline{\mathcal{C}[\mathcal{N}]_1...[\mathcal{N}]_1}^0 = \overline{\mathcal{C}}^0 = \overline{\mathcal{M}}^0$ and $\overline{\mathcal{M}[\mathcal{N}/x]}^{i+1} = \overline{\mathcal{C}[\mathcal{N}]_1...[\mathcal{N}]_1}^{i+1} = \overline{\mathcal{C}}^{i+1}[\overline{\mathcal{N}}^i]_1...[\overline{\mathcal{N}}^i]_1 = \overline{\mathcal{C}}^{i+1}[\overline{\mathcal{N}}^i/x]$ for every $i \geq 0$.

□

Lemma 8 (Approximation of reduction)**.** $\overline{(\lambda x.\mathcal{M})\mathcal{N}}^i \to \overline{\mathcal{M}[\mathcal{N}/x]}^i$ *and* $\overline{(\lambda^! x.\mathcal{M})!\mathcal{N}}^i \to \overline{\mathcal{M}[\mathcal{N}/x]}^i$ *for* $i \geq 0$.

Proof. If the redex is erasing the proof is trivial.

Otherwise, the proof follows easily by Lemma 7: in the first case, by definition of $\lambda^!$-terms we know that $\mathcal{M} = \mathcal{C}[x]_0$, so $\overline{(\lambda x.\mathcal{M})\mathcal{N}}^i = (\lambda x.\overline{\mathcal{C}[x]_0}^i)\overline{\mathcal{N}}^i \to \overline{\mathcal{C}}^i[\overline{\mathcal{N}}^i]_0 = \overline{\mathcal{M}}^i[\overline{\mathcal{N}}^i/x] = \overline{\mathcal{M}[\mathcal{N}/x]}^i$. in the second case, since $\mathcal{M} = \mathcal{C}\lfloor x...x\rfloor_1$ by definition, $\overline{(\lambda^! x.\mathcal{M})!\mathcal{N}}^0 = (\lambda^! x.\overline{\mathcal{M}}^0)!* \to \overline{\mathcal{M}}^0 = \overline{\mathcal{M}[\mathcal{N}/x]}^0$ and $\overline{(\lambda^! x.\mathcal{M})!\mathcal{N}}^{i+1} = (\lambda^! x.\overline{\mathcal{C}\lfloor x...x\rfloor_1}^{i+1})\overline{\mathcal{N}}^i \to \overline{\mathcal{C}}^{i+1}\lfloor\overline{\mathcal{N}}^i...\overline{\mathcal{N}}^i\rfloor_1 = \overline{\mathcal{M}}^{i+1}[\overline{\mathcal{N}}^i/x] = \overline{\mathcal{M}[\mathcal{N}/x]}^{i+1}$ for every $i \leq 0$.

□

Lemma 9 (Approximation and depth)**.**

i. $\mathcal{M} \to_j \mathcal{M}'$ implies $\overline{\mathcal{M}}^i \to_j \overline{\mathcal{M}'}^i$ if $j \leq i$, $\overline{\mathcal{M}}^i = \overline{\mathcal{M}'}^i$ otherwise.

ii. $\overline{\mathcal{M}}^i \to_i \overline{\mathcal{M}'}^i$ implies $|\overline{\mathcal{M}'}^i| < |\overline{\mathcal{M}}^i|$.

Proof.

i. As $\mathcal{M} \to \mathcal{M}'$ by one step at depth j, we have that $\mathcal{M} = \mathcal{C}[\mathcal{P}]_j$ and $\mathcal{M}' = \mathcal{C}[\mathcal{P}']_j$where $\mathcal{P}$ is a redex and $\mathcal{P}'$ is its contractum. Therefore by Lemma 7 $\overline{\mathcal{M}}^i = \overline{\mathcal{M}'}^i$ if $i + 1 \leq j$, otherwise $\overline{\mathcal{M}}^i = \overline{\mathcal{C}}^i[\overline{\mathcal{P}}^{i-j}]_j \to \overline{\mathcal{C}}^i[\overline{\mathcal{P}'}^{i-j}]_j = \overline{\mathcal{M}'}^i$ by using Lemma 8, where the reductions takes place at depth j.

ii. Let $\overline{\mathcal{M}}^i \to \overline{\mathcal{M}'}^i$ by one reduction step at depth i. By Lemma 2 $|\overline{\mathcal{M}'}^i|_j \leq |\overline{\mathcal{M}}^i|_j$ for every $j < i$ and $|\overline{\mathcal{M}'}^i|_i < |\overline{\mathcal{M}}^i|_i$; now we examine the size of $\overline{\mathcal{M}}^i$ at depth $i+1$.

We know that $\overline{\mathcal{M}}^i = \mathcal{C}[\mathcal{P}]_i$ and $\overline{\mathcal{M}'}^i = \mathcal{C}[\mathcal{P}']_i$, where $\mathcal{P}$ is a redex and $\mathcal{P}'$ is its contractum:

- if $\mathcal{P}$ is a β-redex, then $|\overline{\mathcal{M}'}^i|_{i+1} < |\overline{\mathcal{M}}^i|_{i+1}$;
- if $\mathcal{P}$ is a !-redex, then $\overline{\mathcal{M}}^i = \mathcal{C}[\mathcal{P}]_i$ and $\mathcal{P} = (\lambda^! x.\mathcal{N})!*$; moreover, all occurrences of x in $\mathcal{N}$ are at depth 1, so they occur at depth $i+1$ in $\overline{\mathcal{M}}^i$. Since any subterm of $\overline{\mathcal{M}}^i$ at depth $i+1$ is the constant $*$, there are no occurrences of x in $\mathcal{N}$: therefore $\mathcal{P}' = \mathcal{N}$ and $|\overline{\mathcal{M}'}^i|_{i+1} < |\overline{\mathcal{M}}^i|_{i+1}$.

If $j \geq i+2$, then $|\overline{\mathcal{M}'}^i|_j = 0 = |\overline{\mathcal{M}}^i|_j$ and so the statement follows.

□

Finally, with the aid of approximants, we are able to prove the time bound for the computation of the application $\mathcal{P}!\underline{w}$ on a Turing machine:

Property 4. *Let $\mathcal{P}$ be a program; for any $k \geq 2$, there exists a polynomial q such that for any $w \in \{0,1\}^\star$, the reduction of $\overline{\mathcal{P}!\underline{w}}^k$ to its k-normal form can be computed in time $O(2_{k-2}^{q(n)})$ on a Turing machine, where $n = \mathbf{length}(w)$.*

Proof. Observe that $\overline{\mathcal{P}!\underline{w}}^k = \overline{\mathcal{P}}^k!\underline{w}$. By Prop. 3 and Lemma 9.i, $\overline{\mathcal{P}}^k!\underline{w}$ reduces to its $(k-1)$-normal form $\overline{\mathcal{M}_k^1}^k$ in $O(2_{k-2}^{q(n)})$ steps and with intermediary terms of size $O(2_{k-2}^{q(n)})$.
By Lemma 9.ii, the reduction of $\overline{\mathcal{M}_k^1}^k$ at depth k is done in $O(2_{k-2}^{q(n)})$ steps and with intermediary terms of size $O(2_{k-2}^{q(n)})$: therefore, we can conclude by using the fact that one reduction step in a term $\mathcal{M}$ can be simulated in time $p(|\mathcal{M}|)$ on a Turing machine [Ter07], for a suitably chosen polynomial p.

□

2.2 A type assignment system for $\lambda^!$-calculus

In Section 2.1 we proved a complexity bound for computation up to a given depth in the untyped $\lambda^!$-calculus. Note, however, that we cannot make any claim about the result of the application of a program to a binary word: indeed, although the output is well-formed, we can tell nothing else about its shape.

Since we are interested in characterizing predicates (from words to booleans) and functions (from words to words), we would like such result to have the specific shape of either a boolean or a word. To this aim, we now introduce a type assignment

$$\frac{}{\Gamma, x : \mathtt{A} \mid \Delta \mid \Theta \vdash x : \mathtt{A}}\ (Ax^L) \qquad \frac{}{\Gamma \mid \Delta \mid x : \sigma, \Theta \vdash x : \sigma}\ (Ax^P)$$

$$\frac{\Gamma, x : \mathtt{A} \mid \Delta \mid \Theta \vdash \mathcal{M} : \tau}{\Gamma \mid \Delta \mid \Theta \vdash \lambda x.\mathcal{M} : \mathtt{A} \multimap \tau}\ (\multimap I^L) \qquad \frac{\Gamma \mid \Delta, x :!\sigma \mid \Theta \vdash \mathcal{M} : \tau}{\Gamma \mid \Delta \mid \Theta \vdash \lambda^! x.\mathcal{M} :!\sigma \multimap \tau}\ (\multimap I^I)$$

$$\frac{\Gamma_1 \mid \Delta \mid \Theta \vdash \mathcal{M} : \sigma \multimap \tau \quad \Gamma_2 \mid \Delta \mid \Theta \vdash \mathcal{N} : \sigma \quad \Gamma_1 \# \Gamma_2}{\Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash \mathcal{M}\mathcal{N} : \tau}\ (\multimap E)$$

$$\frac{\Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \mathtt{S} \quad \mathtt{a} \notin \mathrm{FTV}(\Gamma) \cup \mathrm{FTV}(\Delta) \cup \mathrm{FTV}(\Theta)}{\Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \forall \mathtt{a}.\mathtt{S}}\ (\forall I)$$

$$\frac{\Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \forall \mathtt{a}.\mathtt{S}}{\Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \mathtt{S}[\sigma/\mathtt{a}]}\ (\forall E) \qquad \frac{\emptyset \mid \emptyset \mid \Theta' \vdash \mathcal{M} : \sigma}{\Gamma \mid !\Theta', \Delta \mid \Theta \vdash !\mathcal{M} :!\sigma}\ (!)$$

$$\frac{\Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \mathtt{S}[\mu\mathtt{a}.\mathtt{S}/\mathtt{a}]}{\Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \mu\mathtt{a}.\mathtt{S}}\ (\mu I) \qquad \frac{\Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \mu\mathtt{a}.\mathtt{S}}{\Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \mathtt{S}[\mu\mathtt{a}.\mathtt{S}/\mathtt{a}]}\ (\mu E)$$

Table 2.1: Derivation rules for typed $\lambda^!$-calculus.

system for $\lambda^!$-calculus based on ELL, such that all typed terms are also well-formed and therefore all previous results are preserved in the typed setting.

2.2.1 An elementary typing system

The set $\mathcal{T}$ of types is generated by the following grammar:

$$\begin{array}{lll} \mathtt{A} & ::= \mathtt{a} \mid \mathtt{S} & \text{(linear types)} \\ \mathtt{S} & ::= \sigma \multimap \sigma \mid \forall \mathtt{a}.\mathtt{S} \mid \mu\mathtt{a}.\mathtt{S} & \text{(strict linear types)} \\ \sigma & ::= \mathtt{A} \mid !\sigma & \text{(types)} \end{array}$$

where a ranges over a countable set of type variables Var.
Observe that, among linear types, we distinguish the subclass of *strict* linear types, also featuring polymorphic types ($\forall\mathtt{a}.\mathtt{S}$) and type fixpoints ($\mu\mathtt{a}.\mathtt{S}$). Such subclass contains all non-atomic linear types and is closed by substitution:

Lemma 10 (Strict linear types)**.** *If* S *is a strict linear type and* σ *is a type, then* $\mathtt{S}[\sigma/\mathtt{a}]$ *is a strict linear type.*

Proof. By induction on S.

Let $\mathtt{S} = \tau \multimap \rho$: then $\tau[\sigma/\mathtt{a}] \multimap \rho[\sigma/\mathtt{a}]$ is an arrow type, thus a strict linear type.

Let $\mathtt{S} = \forall \mathtt{b}.\mathtt{S}'$. By inductive hypothesis $\mathtt{S}'[\sigma/\mathtt{a}]$ is a strict linear type: then $(\forall \mathtt{b}.\mathtt{S}')[\sigma/\mathtt{a}]$ is a strict linear type.

Finally, let $\mathtt{S} = \mu \mathtt{b}.\mathtt{S}'$. By inductive hypothesis $\mathtt{S}'[\sigma/\mathtt{a}]$ is a strict linear type: then $(\mu \mathtt{b}.\mathtt{S}')[\sigma/\mathtt{a}]$ is a strict linear type.

□

Let $\overline{\mathtt{a}}$ be a (possibly empty) sequence of type variables: then $\forall \overline{\mathtt{a}}.\mathtt{S}$ (resp. $\mu \overline{\mathtt{a}}.\mathtt{S}$) denotes both $\mathtt{S}$, if $\overline{\mathtt{a}}$ is empty, and $\forall \mathtt{a}_1.\forall \mathtt{a}_2....\forall \mathtt{a}_n.\mathtt{S}$ (resp. $\mu \mathtt{a}_1.\mu \mathtt{a}_2....\mu \mathtt{a}_n.\mathtt{S}$), if $\overline{\mathtt{a}} = \mathtt{a}_1\mathtt{a}_2....\mathtt{a}_n$ for some $n \geq 1$.

A *basis* is a partial function from variables to types, with finite domain, ranging over Γ, Δ, Θ. As usual, the domain of a basis Γ is denoted by $\mathrm{dom}(\Gamma)$ and it represents the set of variables for which there exists a mapping in Γ, that is, $\mathrm{dom}(\Gamma) = \{x \mid \Gamma(x) \text{ is defined}\}$.
Taking inspiration from [CDR08], we consider three different bases having pairwise disjoint domains, called the *linear*, *modal* and *parking* basis, such that Γ assigns linear types to variables, while Δ maps variables to modal types:

- Γ is the *linear basis*, where for every $x \in \mathrm{dom}(\Gamma)$ there is a linear type $\mathtt{A}$ such that $\Gamma(x) = \mathtt{A}$;
- Δ is the *modal basis*, where for every $x \in \mathrm{dom}(\Delta)$ there is a type σ such that $\Delta(x) = !\sigma$;
- Θ is the *parking basis*, where for every $x \in \mathrm{dom}(\Delta)$ there is a type σ such that $\Theta(x) = \sigma$.

The rules of the type assignment system are given in Table 2.1, where $\Gamma_1 \# \Gamma_2$ stands for $\mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \emptyset$, while $\Delta_1 \subseteq \Delta_2$ means that $x : \sigma \in \Delta_1$ implies $x : \sigma \in \Delta_2$. Moreover, $\mathrm{FTV}(\sigma)$ denotes the set of free variables of σ, while $\mathrm{FTV}(\Gamma)$ denotes the set $\{\mathtt{a} \mid \mathtt{a} \in \cup_{x \in \mathrm{dom}(\Gamma)} \mathrm{FTV}(\Gamma(x))\}$.
The typing system proves statements of the shape $\Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \sigma$; derivations are ranged over by Π, Σ, Φ, so that we write $\Pi \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \sigma$ to identify particular derivation proving the desired statement. When all three bases are empty, we write the derivation as $\Pi \triangleright \vdash \mathcal{M} : \sigma$. As usual, we use the notation $\mathcal{M}^\sigma$ as a shorthand for *there is a derivation assigning type σ to $\mathcal{M}$*.

We say that a term $\mathcal{M}$ is *well-typed* if and only if there is a derivation $\Pi \triangleright \Gamma \mid \Delta \mid \emptyset \vdash \mathcal{M} : \sigma$ for some Γ, Δ, σ: indeed, variables in the parking context are considered as having an intermediary status, since they eventually have to shift to the modal context in order for the term to be well-typed.

Some comments about the rules are in order:

- in rule $(\multimap E)$, $\mathcal{M}$ and $\mathcal{N}$ share variables in the modal and parking basis, but their linear bases must be disjoint;

- there is no axiom rule for introducing variables in the modal basis, but a variable can be introduced in the parking basis and then moved to the modal basis by applying rule (!);

- there is no abstraction rule for variables in the parking basis, thus underlining the fact that parking variables only have a "temporary" status;

- with respect to the system of [CDR08], rule (!) is restricted so that both the linear and modal bases in the premise are empty; such restriction guarantees that typed terms are also well-formed, a condition which is crucial in order to obtain a stratified complexity bound for a specific k depending on the type, instead of the usual rough elementary bound.

- rules $(\mu I), (\mu E), (\forall I)$ and $(\forall E)$, also referred to as *non-constructive* rules, are the only rules which do not contribute to the syntactic construction of the term.

The following generation Lemma allows to infer the general shape of a derivation from the shape of its subject:

Lemma 11 (Generation)**.** *Let* $\Pi \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \sigma$.

i. $\mathcal{M} = x$ *implies* $x \in \mathrm{dom}(\Gamma) \cup \mathrm{dom}(\Theta)$ *and* Π *ends with an application of either* (Ax^L) *or* (Ax^P) *rule, followed by a (possibly empty) sequence of non-constructive rules.*

ii. $\mathcal{M} = \lambda x.\mathcal{N}$ *implies* Π *ends with an application of* $(\multimap I^L)$ *rule, followed by a (possibly empty) sequence of non-constructive rules.*

iii. $\mathcal{M} = \lambda^! x.\mathcal{N}$ *implies* Π *ends with an application of* $(\multimap I^I)$ *rule, followed by a (possibly empty) sequence of non-constructive rules.*

iv. $\mathcal{M} = \mathcal{N}\mathcal{P}$ *implies* Π *ends with an application of* $(\multimap E)$ *rule, followed by a (possibly empty) sequence of non-constructive rules.*

v. $\mathcal{M} = !\mathcal{N}$ *implies* Π *ends with an application of* (!) *rule and* $\sigma = !\tau$*, for some* τ.

Proof. All points follow easily by checking the rules of the system.

□

Our aim is to give a characterization of a hierarchy of complexity classes with respect to typed programs, which should enjoy the stratification property given in

Theorem 1: therefore, in order to examine the computation of a typed $\lambda^!$-term, we need to show that both the type and the maximum depth of a term are preserved during reduction.

By observing the shape of the rules, it is easy to see that new mappings of variables to types can be added to any basis by means of implicit weakening; moreover, it is always possible to move a variable from the linear to the parking basis:

Lemma 12 (Weakening)**.** *If* $\Pi \triangleright \Gamma_1 \mid \Delta_1 \mid \Theta_1 \vdash \mathcal{M} : \sigma$*, then there is a derivation* $\Sigma \triangleright \Gamma_1, \Gamma_2 \mid \Delta_1, \Delta_2 \mid \Theta_1, \Theta_2 \vdash \mathcal{M} : \sigma$*, for every* $\Gamma_2, \Delta_2, \Theta_2$ *disjoint from each other and from* $\Gamma_1, \Delta_1, \Theta_1$*.*

Proof. Easy, by induction on Π.

□

Lemma 13 (Shifting)**.** *If* $\Pi \triangleright \Gamma, x : \mathtt{A} \mid \Delta \mid \Theta \vdash \mathcal{M} : \tau$ *then there is a derivation* $\Sigma \triangleright \Gamma \mid \Delta \mid \Theta, x : \mathtt{A} \vdash \mathcal{M} : \tau$*.*

Proof. Easy, by induction on Π.

□

The substitution lemma, essential to the proof of the subject reduction property, is split into three points, one for each basis in which the substituted variable may occur:

Lemma 14 (Substitution)**.** *Let* $\Gamma_1 \# \Gamma_2$*.*

i) If $\Pi \triangleright \Gamma_1, x : \mathtt{A} \mid \Delta \mid \Theta \vdash \mathcal{M} : \tau$ *and* $\Sigma \triangleright \Gamma_2 \mid \Delta \mid \Theta \vdash \mathcal{N} : \mathtt{A}$*, then there is* $\Phi \triangleright \Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash \mathcal{M}[\mathcal{N}/x] : \tau$*.*

ii) If $\Pi \triangleright \Gamma_1 \mid \Delta \mid \Theta, x : \sigma \vdash \mathcal{M} : \tau$ *and* $\Sigma \triangleright \emptyset \mid \Delta \mid \Theta \vdash \mathcal{N} : \sigma$*, then there is* $\Phi \triangleright \Gamma_1 \mid \Delta \mid \Theta \vdash \mathcal{M}[\mathcal{N}/x] : \tau$*.*

iii) If $\Pi \triangleright \Gamma_1 \mid \Delta, x :!\sigma \mid \Theta \vdash \mathcal{M} : \tau$ *and* $\Sigma \triangleright \Gamma_2 \mid \Delta \mid \Theta \vdash !\mathcal{N} :!\sigma$*, then there is* $\Phi \triangleright \Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash \mathcal{M}[\mathcal{N}/x] : \tau$*.*

Proof. All three points follow by induction on Π.

i) Let Π end with an application of rule (Ax^L); then either Π is

$$\frac{}{\Gamma_1, x : \mathtt{A} \mid \Delta \mid \Theta \vdash x : \mathtt{A}}\ (Ax^L)$$

where $\mathcal{M} = x$ and $\tau = \mathtt{A}$, so $\Phi \triangleright \Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash \mathcal{N} : \mathtt{A}$ follows directly by applying Lemma 12 to Σ, or Π is

$$\frac{}{\Gamma_1', x : \mathtt{A}, y : \mathtt{B} \mid \Delta \mid \Theta \vdash y : \mathtt{B}}\ (Ax^L)$$

where $\mathcal{M} = y$, $\tau = \mathtt{B}$ and $\Gamma_1 = \Gamma_1', y : \mathtt{B}$, so $S(\Sigma, \Pi)$ is

$$\frac{}{\Gamma_1', \Gamma_2, y : \mathtt{B} \mid \Delta \mid \Theta \vdash y : \mathtt{B}} \ (Ax^L)$$

Let Π be

$$\frac{}{\Gamma_1, x : \mathtt{A} \mid \Delta \mid \Theta', y : \tau \vdash y : \tau} \ (Ax^P)$$

where $\mathcal{M} = y$ and $\Theta = \Theta', y : \tau$, so Φ is

$$\frac{}{\Gamma_1, \Gamma_2 \mid \Delta \mid \Theta', y : \tau \vdash y : \tau} \ (Ax^L)$$

Let Π be

$$\frac{\Pi' \triangleright \Gamma_1, x : \mathtt{A}, y : \mathtt{B} \mid \Delta \mid \Theta \vdash \mathcal{P} : \phi}{\Gamma_1, x : \mathtt{A} \mid \Delta \mid \Theta \vdash \lambda y.\mathcal{P} : \mathtt{B} \multimap \phi} \ (\multimap I^L)$$

where $\mathcal{M} = \lambda y.\mathcal{P}$ and $\tau = \mathtt{B} \multimap \phi$. By inductive hypothesis on Π' there is $\Phi' \triangleright \Gamma_1, \Gamma_2, y : \mathtt{B} \mid \Delta \mid \Theta \vdash \mathcal{P}[\mathcal{N}/x] : \phi$; then Φ is obtained by applying rule $(\multimap I^L)$ to Φ'.

Let Π be

$$\frac{\Pi' \triangleright \Gamma_1, x : \mathtt{A} \mid \Delta, y :!\rho \mid \Theta \vdash \mathcal{P} : \phi}{\Gamma_1, x : \mathtt{A} \mid \Delta \mid \Theta \vdash \lambda y.\mathcal{P} :!\rho \multimap \phi} \ (\multimap I^I)$$

where $\mathcal{M} = \lambda y.\mathcal{P}$ and $\tau =!\rho \multimap \phi$. By Lemma 12 there is $\Gamma_2 \mid \Delta, y :!\rho \mid \Theta \vdash \mathcal{N} : \mathtt{A}$, so by inductive hypothesis on Π' there is $\Phi' \triangleright \Gamma_1, \Gamma_2 \mid \Delta, y :!\rho \mid \Theta \vdash \mathcal{P}[\mathcal{N}/x] : \phi$; then Φ is obtained by applying rule $(\multimap I^L)$ to Φ'.

Let Π be

$$\frac{\Pi_1 \triangleright \Gamma, x : \mathtt{A} \mid \Delta \mid \Theta \vdash \mathcal{P} : \rho \multimap \tau \quad \Pi_2 \triangleright \Gamma' \mid \Delta \mid \Theta \vdash \mathcal{Q} : \rho}{\Gamma, \Gamma', x : \mathtt{A} \mid \Delta \mid \Theta \vdash \mathcal{P}\mathcal{Q} : \tau} \ (\multimap E)$$

where $x \in \mathrm{FV}(\mathcal{P})$, $x \notin \mathrm{FV}(\mathcal{Q})$, $\mathcal{M} = \mathcal{P}\mathcal{Q}$ and $\Gamma_1 = \Gamma, \Gamma'$. By inductive hypothesis on Π_1 and Π_2 there are $\Phi_1 \triangleright \Gamma, \Gamma_2 \mid \Delta \mid \Theta \vdash \mathcal{P}[\mathcal{N}/x] : \rho \multimap \tau$ and $\Phi_2 \triangleright \Gamma'' \mid \Delta \mid \Theta \vdash \mathcal{Q} : \rho$: then Φ is obtained by applying rule $(\multimap E)$ to Φ_1 and Φ_2.
The case of $x \notin \mathrm{FV}(\mathcal{P}), x \in \mathrm{FV}(\mathcal{Q})$ is similar.

Let Π be

$$\frac{\Pi' \triangleright \emptyset \mid \emptyset \mid \Theta' \vdash \mathcal{P} : \rho}{\Gamma_1, x : \mathtt{A} \mid !\Theta', \Delta' \mid \Theta \vdash !\mathcal{P} : !\rho}\ (!)$$

where $\mathcal{M} = !\mathcal{P}$, $\tau = !\rho$ and $\Delta = !\Theta', \Delta'$. Observe that $!(\mathcal{P}[\mathcal{N}/x]) = !\mathcal{P}$, since $x \notin \mathrm{FV}(\mathcal{P})$, so Φ is

$$\frac{\Pi' \triangleright \emptyset \mid \emptyset \mid \Theta' \vdash \mathcal{P} : \rho}{\Gamma_1, \Gamma_2 \mid !\Theta', \Delta' \mid \Theta \vdash !\mathcal{P} : !\rho}\ (!)$$

The cases of $(\forall I), (\forall E), (\mu I), (\mu E)$ follow easily by induction.

ii) Let Π be

$$\frac{}{\Gamma_1', y : \mathtt{A} \mid \Delta \mid \Theta, x : \sigma \vdash y : \mathtt{A}}\ (Ax^L)$$

where $\mathcal{M} = y$, $\tau = \mathtt{A}$ and $\Gamma_1 = \Gamma_1', y : \mathtt{A}$, so Φ is

$$\frac{}{\Gamma_1', y : \mathtt{A} \mid \Delta \mid \Theta \vdash y : \mathtt{A}}\ (Ax^L)$$

Let Π end with an application of rule (Ax^P); then either Π is

$$\frac{}{\Gamma_1 \mid \Delta \mid \Theta, x : \sigma \vdash x : \sigma}\ (Ax^P)$$

where $\mathcal{M} = x$ and $\tau = \sigma$, so $\Phi \triangleright \Gamma_1 \mid \Delta \mid \Theta \vdash \mathcal{N} : \tau$ follows directly by applying Lemma 12 to Σ, or Π is

$$\frac{}{\Gamma_1 \mid \Delta \mid \Theta', x : \sigma, y : \tau \vdash y : \tau}\ (Ax^P)$$

where $\mathcal{M} = y$ and $\Theta = \Theta', y : \tau$, so Φ is

$$\frac{}{\Gamma_1 \mid \Delta \mid \Theta', y : \tau \vdash y : \tau}\ (Ax^P)$$

Let Π be

$$\frac{\Pi' \triangleright \Gamma_1, y : \mathtt{B} \mid \Delta \mid \Theta, x : \sigma \vdash \mathcal{P} : \phi}{\Gamma_1 \mid \Delta \mid \Theta, x : \sigma \vdash \lambda y.\mathcal{P} : \mathtt{B} \multimap \phi}\ (\multimap I^L)$$

where $\mathcal{M} = \lambda y.\mathcal{P}$ and $\tau = \mathtt{B} \multimap \phi$. By inductive hypothesis on Π' there is $\Phi' \triangleright \Gamma_1, y : \mathtt{B} \mid \Delta \mid \Theta \vdash \mathcal{P}[\mathcal{N}/x] : \phi$; then Φ is obtained by applying rule $(\multimap I^L)$ to Φ'.

Let Π be

$$\frac{\Pi' \triangleright \Gamma_1 \mid \Delta, y :!\rho \mid \Theta, x : \sigma \vdash \mathcal{P} : \phi}{\Gamma_1 \mid \Delta \mid \Theta, x : \sigma \vdash \lambda y.\mathcal{P} :!\rho \multimap \phi} \ (\multimap I^I)$$

where $\mathcal{M} = \lambda y.\mathcal{P}$ and $\tau =!\rho \multimap \phi$. By Lemma 12 there is $\emptyset \mid \Delta, y :!\rho \mid \Theta \vdash \mathcal{N} : \sigma$, so by inductive hypothesis on Π' there is $\Phi' \triangleright \Gamma_1 \mid \Delta, y :!\rho \mid \Theta \vdash \mathcal{P}[\mathcal{N}/x] : \phi$; then Φ is obtained by applying rule $(\multimap I^L)$ to Φ'.

Let Π be

$$\frac{\Pi_1 \triangleright \Gamma \mid \Delta \mid \Theta, x : \sigma \vdash \mathcal{P} : \rho \multimap \tau \quad \Pi_2 \triangleright \Gamma' \mid \Delta \mid \Theta, x : \sigma \vdash \mathcal{Q} : \rho}{\Gamma, \Gamma' \mid \Delta \mid \Theta, x : \sigma \vdash \mathcal{P}\mathcal{Q} : \tau} \ (\multimap E)$$

where $\mathcal{M} = \mathcal{P}\mathcal{Q}$ and $\Gamma = \Gamma, \Gamma'$. By inductive hypothesis on Π_1 and Π_2 there are $\Phi_1 \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathcal{P}[\mathcal{N}/x] : \rho \multimap \tau$ and $\Phi_2 \triangleright \Gamma'' \mid \Delta \mid \Theta \vdash \mathcal{Q} : \rho$: then Φ is obtained by applying rule $(\multimap E)$ to Φ_1 and Φ_2.

Let Π be

$$\frac{\Pi' \triangleright \emptyset \mid \emptyset \mid \Theta' \vdash \mathcal{P} : \rho}{\Gamma_1 \mid !\Theta', \Delta' \mid \Theta, x : \sigma \vdash !\mathcal{P} :!\rho} \ (!)$$

where $\mathcal{M} =!\mathcal{P}$, $\tau =!\rho$ and $\Delta =!\Theta', \Delta'$. Observe that $!(\mathcal{P}[\mathcal{N}/x]) =!\mathcal{P}$, since $x \notin \mathrm{FV}(\mathcal{P})$, so Φ is

$$\frac{\Pi' \triangleright \emptyset \mid \emptyset \mid \Theta' \vdash \mathcal{P} : \rho}{\Gamma_1 \mid !\Theta', \Delta' \mid \Theta \vdash !\mathcal{P} :!\rho} \ (!)$$

The cases of $(\forall I), (\forall E), (\mu I), (\mu E)$ follow easily by induction.

iii) Observe that, by Lemma 11, Σ is

$$\frac{\Sigma' \triangleright \emptyset \mid \emptyset \mid \Theta'' \vdash \mathcal{N} : \sigma}{\Gamma_2 \mid \Delta \mid \Theta \vdash !\mathcal{N} :!\sigma} \ (!)$$

where $\Delta =!\Theta'', \Delta''$ for some Δ''.

Let Π be

$$\frac{}{\Gamma_1', y : \mathtt{B} \mid \Delta, x :!\sigma \mid \Theta \vdash y : \mathtt{B}} \ (Ax^L)$$

where $\mathcal{M} = y$, $\tau = \mathtt{B}$ and $\Gamma_1 = \Gamma_1', y : \mathtt{B}$, so $S(\Sigma, \Pi)$ is

$$\frac{}{\Gamma_1', \Gamma_2, y : \mathtt{B} \mid \Delta \mid \Theta \vdash y : \mathtt{B}} \ (Ax^L)$$

Let Π be

$$\frac{}{\Gamma_1 \mid \Delta, x :!\sigma \mid \Theta', y : \tau \vdash y : \tau}\ (Ax^P)$$

where $\mathcal{M} = y$ and $\Theta = \Theta', y : \tau$, so $S(\Sigma, \Pi)$ is

$$\frac{}{\Gamma_1, \Gamma_2 \mid \Delta \mid \Theta', y : \tau \vdash y : \tau}\ (Ax^P)$$

Let Π be

$$\frac{\Pi' \triangleright \Gamma_1, y : \mathtt{B} \mid \Delta, x :!\sigma \mid \Theta \vdash \mathcal{P} : \phi}{\Gamma_1 \mid \Delta, x :!\sigma \mid \Theta \vdash \lambda y.\mathcal{P} : \mathtt{B} \multimap \phi}\ (\multimap I^L)$$

where $\mathcal{M} = \lambda y.\mathcal{P}$ and $\tau = \mathtt{B} \multimap \phi$. By inductive hypothesis on Π' there is $\Phi' \triangleright \Gamma_1, \Gamma_2, y : \mathtt{B} \mid \Delta \mid \Theta \vdash \mathcal{P}[\mathcal{N}/x] : \phi$; then Φ is obtained by applying rule $(\multimap I^L)$ to Φ'.

Let Π be

$$\frac{\Pi' \triangleright \Gamma_1 \mid \Delta, x :!\sigma, y :!\rho \mid \Theta \vdash \mathcal{P} : \phi}{\Gamma_1 \mid \Delta, x :!\sigma \mid \Theta \vdash \lambda y.\mathcal{P} :!\rho \multimap \phi}\ (\multimap I^I)$$

where $\mathcal{M} = \lambda y.\mathcal{P}$ and $\tau = !\rho \multimap \phi$. By Lemma 12 there is $\Gamma_2 \mid \Delta, y :!\rho \mid \Theta \vdash !\mathcal{N} : !\sigma$, so by inductive hypothesis on Π' there is $\Phi' \triangleright \Gamma_1, \Gamma_2 \mid \Delta, y :!\rho \mid \Theta \vdash \mathcal{P}[\mathcal{N}/x] : \phi$; then Φ is obtained by applying rule $(\multimap I^L)$ to Φ'.

Let Π be

$$\frac{\Pi_1 \triangleright \Gamma \mid \Delta, x :!\sigma \mid \Theta \vdash \mathcal{P} : \rho \multimap \tau \quad \Pi_2 \triangleright \Gamma' \mid \Delta, x :!\sigma \mid \Theta \vdash \mathcal{Q} : \rho}{\Gamma, \Gamma' \mid \Delta, x :!\sigma \mid \Theta \vdash \mathcal{P}\mathcal{Q} : \tau}\ (\multimap E)$$

where $\mathcal{M} = \mathcal{P}\mathcal{Q}$ and $\Gamma_1 = \Gamma, \Gamma'$. From Σ' we can build the following derivation:

$$\frac{\Sigma' \triangleright \emptyset \mid \emptyset \mid \Theta'' \vdash \mathcal{N} : \sigma}{\emptyset \mid \Delta \mid \Theta \vdash !\mathcal{N} :!\sigma}\ (!)$$

By inductive hypothesis on Π_1 and Π_2 there are $\Phi_1 \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathcal{P}[\mathcal{N}/x] : \rho \multimap \tau$ and $\Phi_2 \triangleright \Gamma' \mid \Delta \mid \Theta \vdash \mathcal{Q}[\mathcal{N}/x] : \rho$: then Φ is obtained by applying rule $(\multimap E)$ to Φ_1 and Φ_2.

Let Π end with an application of rule (!); then either $x \notin \mathrm{FV}(\mathcal{P})$ and Π is

$$\frac{\Pi' \triangleright \emptyset \mid \emptyset \mid \Theta' \vdash \mathcal{P} : \rho}{\Gamma_1 \mid \Delta, x :!\sigma \mid \Theta \vdash !\mathcal{P} :!\rho}\ (!)$$

where $\mathcal{M} = !\mathcal{P}$, $\tau = !\rho$ and $\Delta = !\Theta', \Delta'$, so Φ is

$$\frac{\Pi' \triangleright \emptyset \mid \emptyset \mid \Theta' \vdash \mathcal{P} : \rho}{\Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash !\mathcal{P} : !\rho}\ (!)$$

or $x \in \mathrm{FV}(\mathcal{P})$ and Π is

$$\frac{\Pi' \triangleright \emptyset \mid \emptyset \mid \Theta', x : \sigma \vdash \mathcal{P} : \rho}{\Gamma_1 \mid \Delta, x : !\sigma \mid \Theta \vdash !\mathcal{P} : !\rho}\ (!)$$

where $\mathcal{M} = !\mathcal{P}$, $\tau = !\rho$ and $\Delta = !\Theta', \Delta'$. By Lemma 12 we can build $\Pi'' \triangleright \emptyset \mid \emptyset \mid \Theta''', x : \sigma \vdash \mathcal{P} : \rho$ and $\Sigma'' \triangleright \emptyset \mid \emptyset \mid \Theta''' \vdash \mathcal{N} : \sigma$ such that $\Theta''' = \Theta', \Theta''$ and $\Delta = !\Theta''', \Delta'''$ for some context Δ'''. Then by applying point ii of the current Lemma to Π'', followed by one application of rule (!), Φ is

$$\frac{\Phi' \triangleright \emptyset \mid \emptyset \mid \Theta''' \vdash \mathcal{P}[\mathcal{N}/x] : \rho}{\Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash !(\mathcal{P}[\mathcal{N}/x]) : !\rho}\ (!)$$

The cases of $(\forall I), (\forall E), (\mu I), (\mu E)$ follow easily by induction.

□

As usual, a *detour* occurs in a derivation whenever the application of a rule introducing a connective is immediately followed by an application of a rule eliminating the same connective. Here we have three kinds of detours, namely the $\forall$-detour, the μ-detour and the $\multimap$-detour.
Because of the non-constructive rules and their respective detours, derivations of $\lambda^!$-terms can often become quite involved. However, all $\forall$ and μ detours in a derivation can be erased without altering the structure of the term, so generating a *clean* derivation of the same statement:

Lemma 15. *Let $\Pi \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \sigma$; then there is a clean derivation Π', proving the same statement as Π, which is obtained from Π by erasing all $\forall$-detours and μ-detours.*

Proof. It is sufficient to erase all $\forall$-detour and μ-detours as in Figures 2.2 and 2.3.
□

Using Lemma 14 and Lemma 15, the subject reduction property follows easily:

Theorem 3 (Subject reduction)**.** *$\Pi \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \sigma$ and $\mathcal{M} \to \mathcal{M}'$ imply $\Pi' \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathcal{M}' : \sigma$.*

$$\dfrac{\dfrac{\Pi \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \mathtt{S}}{\Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \forall \mathtt{a}.\mathtt{S}}\,(\forall I)}{\Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \mathtt{S}[\mathtt{S}'/\mathtt{a}]}\,(\forall E) \quad \rightsquigarrow \quad \Pi[\mathtt{S}'/\mathtt{a}] \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \mathtt{S}[\mathtt{S}'/\mathtt{a}]$$

Figure 2.2: Erasing of a $\forall$-detour.

$$\dfrac{\dfrac{\Pi \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \mathtt{S}[\mu\mathtt{a}.\mathtt{S}/\mathtt{a}]}{\Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \mu\mathtt{a}.\mathtt{S}}\,(\mu I)}{\Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \mathtt{S}[\mu\mathtt{a}.\mathtt{S}/\mathtt{a}]}\,(\mu E) \quad \rightsquigarrow \quad \Pi \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \mathtt{S}[\mu\mathtt{a}.\mathtt{S}/\mathtt{a}]$$

Figure 2.3: Erasing of a $\forall$-detour.

Proof. Let $\mathcal{M} = \mathcal{C}[\mathcal{R}]$, where either $\mathcal{R} = (\lambda x.\mathcal{P})\mathcal{Q}$ or$(\lambda^! x.\mathcal{P})!\mathcal{Q}$: we proceed by induction on $\mathcal{C}$.

Let $\mathcal{C} = \Box$, so $\mathcal{M} = \mathcal{R}$ and $\mathcal{M}' = \mathcal{P}[\mathcal{Q}/x]$. If $\mathcal{M}$ is a β-redex, then by Lemma 11 Π is

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma', x : \mathtt{A}'' \mid \Delta \mid \Theta \vdash \mathcal{P} : \sigma''}{\Gamma' \mid \Delta \mid \Theta \vdash \lambda x.\mathcal{P} : \mathtt{A}'' \multimap \sigma''}\,(\multimap I^L)}{\Gamma' \mid \Delta \mid \Theta \vdash \lambda x.\mathcal{P} : \mathtt{A}' \multimap \sigma'}\,\delta' \qquad \Gamma'' \mid \Delta \mid \Theta \vdash \mathcal{N} : \mathtt{A}'}{\Sigma \triangleright \Gamma', \Gamma'' \mid \Delta \mid \Theta \vdash (\lambda x.\mathcal{P})\mathcal{N} : \sigma'}\,(\multimap E)}{\Gamma', \Gamma'' \mid \Delta \mid \Theta \vdash (\lambda x.\mathcal{P})\mathcal{N} : \sigma}\,\delta$$

Otherwise, if $\mathcal{M}$ is a !-redex, then by Lemma 11 Π is

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma' \mid \Delta, x : \tau'' \mid \Theta \vdash \mathcal{P} : \sigma''}{\Gamma' \mid \Delta \mid \Theta \vdash \lambda^! x.\mathcal{P} : \tau'' \multimap \sigma''}\,(\multimap I^L)}{\Gamma' \mid \Delta \mid \Theta \vdash \lambda^! x.\mathcal{P} : \tau' \multimap \sigma'}\,\delta' \qquad \Gamma'' \mid \Delta \mid \Theta \vdash !\mathcal{N} : \tau'}{\Sigma \triangleright \Gamma', \Gamma'' \mid \Delta \mid \Theta \vdash (\lambda^! x.\mathcal{P})!\mathcal{N} : \sigma'}\,(\multimap E)}{\Gamma', \Gamma'' \mid \Delta \mid \Theta \vdash (\lambda^! x.\mathcal{P})!\mathcal{N} : \sigma}\,\delta$$

In both cases, δ and δ' are (possibly empty) sequences of non-constructive rules: it is easy to check that δ' is a sequence of $\forall$-detours and μ-detours, since both the initial and the final types are arrow types. By Lemma 15 there is a clean derivation proving the same statement as Σ, such that δ' is empty and $\mathtt{A}'' = \mathtt{A}', \tau'' = \tau', \sigma'' = \sigma'$. By Lemma 14 there is $\Sigma' \triangleright \Gamma', \Gamma'' \mid \Delta \mid \Theta \vdash \mathcal{P}[\mathcal{N}/x] : \sigma'$: then Π' is obtained by applying sequence δ to Σ'.

All the other cases follow easily by induction.

□

Furthermore we examine the depth of the free occurrences of a variable in a typed $\lambda^!$-term. The following property, similar to the result given in Theorem 1 for the

untyped calculus, essentially depends on the restriction imposed on the premise of rule (!), which is responsible for the fact that all free variables are either at depth 0 or 1, so effectively guaranteeing that the depth of a (sub)term does not increase:

Property 5 (Depth of a free variable)**.** *Let* $\Pi \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathcal{M} : \sigma$ *and* $x \in \mathrm{FV}(\mathcal{M})$*:*

- $x \in \mathrm{dom}(\Gamma) \cup \mathrm{dom}(\Theta)$ *if and only if* x *occurs at depth* 0 *in* $\mathcal{M}$*,*

- $x \in \mathrm{dom}(\Delta)$ *if and only if* x *occurs at depth* 1 *in* $\mathcal{M}$*.*

Proof. By induction on Π.

Let Π be

$$\frac{}{\Gamma, x : \mathtt{A} \mid \Delta \mid \Theta \vdash x : \mathtt{A}} \ (Ax^L)$$

Then $x \in \mathrm{dom}(\Gamma, x : \mathtt{A}) \cup \mathrm{dom}(\Theta)$ if and only if x occurs at depth 0 in x. The case of (Ax^P) is similar.

Let Π be

$$\frac{\emptyset \mid \emptyset \mid \Theta', x : \tau \vdash \mathcal{N} : \sigma}{\Gamma \mid !\Theta', \Delta', x :!\tau \mid \Theta \vdash !\mathcal{N} :!\sigma} \ (!)$$

By inductive hypothesis $x \in \mathrm{dom}(\Theta', x : \tau)$ if and only if x occurs at depth 0 in $\mathcal{N}$: then $x \in \mathrm{dom}(!\Theta', \Delta', x :!\tau)$ if and only if x occurs at depth 1 in $!\mathcal{N}$.

All the other cases follow easily by induction.

□

Finally, by exploiting the results of Theorem 3 and Property 5, we are able to show that the typed terms are exactly those of $\lambda^!$-calculus:

Property 6. *If a term is well-typed, then it is also well-formed.*

Proof. Consider a derivation Π typing $\mathcal{M}$. For each subderivation of Π of the shape

$$\frac{\Gamma, x : \mathtt{A} \mid \Delta \mid \Theta \vdash \mathcal{P} : \sigma}{\Gamma \mid \Delta \mid \Theta \vdash \lambda x.\mathcal{P} : \mathtt{A} \multimap \sigma} \ (\multimap I^L) \qquad \text{or} \qquad \frac{\Gamma \mid \Delta, y :!\tau \mid \Theta \vdash \mathcal{Q} : \sigma}{\Gamma \mid \Delta \mid \Theta \vdash \lambda^! y.\mathcal{Q} :!\tau \multimap \sigma} \ (\multimap I^I)$$

by Prop. 5 x occurs at most once at depth 0 in $\mathcal{P}$, while all occurrences of y (if any) are at depth 1 in $\mathcal{Q}$: this corresponds exactly to the notion of well-formed terms given in Definition 8.

□

2.2.2 Types of data and pairs

In Section 2.1.5 we introduced the kind of $\lambda^!$-terms which serve for the encoding of data. For such terms we define the following *datatypes*, adapted from System F, representing respectively booleans, Church tally integers, Church binary words and Scott binary words:

$$\begin{array}{lcl} \mathrm{B} & \stackrel{\text{def}}{=} & \forall \mathtt{a}.\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a} \\ \mathrm{N} & \stackrel{\text{def}}{=} & \forall \mathtt{a}.!(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a}) \\ \mathrm{W} & \stackrel{\text{def}}{=} & \forall \mathtt{a}.!(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a}) \\ \mathrm{W_S} & \stackrel{\text{def}}{=} & \mu\mathtt{b}.\forall \mathtt{a}.(\mathtt{b} \multimap \mathtt{a}) \multimap (\mathtt{b} \multimap \mathtt{a}) \multimap (\mathtt{a} \multimap \mathtt{a}) \end{array}$$

Note that Scott binary words have already been used in the literature for the study of light logics [LB06; RV10; BT10].

Representation of pairs As usual, it is possible to define the connective $\otimes$ for pairs by second order as:

$$\sigma \otimes \tau \stackrel{\text{def}}{=} \forall \mathtt{a}.(\sigma \multimap \tau \multimap \mathtt{a}) \multimap \mathtt{a}.$$

For such new type construction, the following syntactic sugar on terms is specified:

$$\begin{array}{lcll} \mathcal{M}_1 \otimes \mathcal{M}_2 & \stackrel{\text{def}}{=} & \lambda x.x\mathcal{M}_1\mathcal{M}_2 & : \sigma_1 \otimes \sigma_2 \\ \lambda(x_1 \otimes x_2).\mathcal{M} & \stackrel{\text{def}}{=} & \lambda x.(x\lambda y_1 y_2.\lambda z.zy_1y_2)(\lambda x_1x_2.\mathcal{M}) & : (\mathtt{A}_1 \otimes \mathtt{A}_2) \multimap \tau \\ \lambda^!(x_1 \otimes x_2).\mathcal{M} & \stackrel{\text{def}}{=} & \lambda x.(x\lambda^! y_1 y_2.\lambda z.z!y_1!y_2)(\lambda^! x_1x_2.\mathcal{M}) & : (!\sigma_1 \otimes !\sigma_2) \multimap \tau \\ & & & \\ \pi_i & \stackrel{\text{def}}{=} & \lambda(x_1 \otimes x_2).x_i & : (\mathtt{A}_1 \otimes \mathtt{A}_2) \multimap \mathtt{A}_i \quad \text{for } i \in \{1,2\} \\ \pi_i^! & \stackrel{\text{def}}{=} & \lambda^!(x_1 \otimes x_2).!x_i & : (!\sigma_1 \otimes !\sigma_2) \multimap !\sigma_i \quad \text{for } i \in \{1,2\} \end{array}$$

along with the two derivable reductions rules:

$$(\lambda(x_1 \otimes x_2).\mathcal{N})(\mathcal{M}_1 \otimes \mathcal{M}_2) \to \mathcal{N}[\mathcal{M}_1/x_1, \mathcal{M}_2/x_2]$$

$$(\lambda^!(x_1 \otimes x_2).\mathcal{N})(!\mathcal{M}_1 \otimes !\mathcal{M}_2) \to \mathcal{N}[\mathcal{M}_1/x_1, \mathcal{M}_2/x_2].$$

Observe that, contrary to what one would expect, we chose not to define $\lambda^!(x_1 \otimes x_2).\mathcal{M}$ simply as $\lambda x.x\lambda^! x_1x_2.\mathcal{M}$. The reason behind this choice is that the application of the latter could require the substitution of a non-linear type to a type variable, an operation which is forbidden by our typing system: indeed such behavior would undermine subject reduction, a crucial property for the characterization we aim to prove.

It is easy to check that the typing rules in Table 2.2 are derivable: the full proof is given in Appendix A.2.

$$\frac{\Gamma, x_1 : \mathtt{A}_1, x_2 : \mathtt{A}_2 \mid \Delta \mid \Theta \vdash \mathcal{M} : \tau}{\Gamma \mid \Delta \mid \Theta \vdash \lambda(x_1 \otimes x_2).\mathcal{M} : (\mathtt{A}_1 \otimes \mathtt{A}_2) \multimap \tau} \ (\multimap I^L_\otimes)$$

$$\frac{\Gamma \mid \Delta, x_1 :!\sigma_1, x_2 :!\sigma_2 \mid \Theta \vdash \mathcal{M} : \tau}{\Gamma \mid \Delta \mid \Theta \vdash \lambda^!(x_1 \otimes x_2).\mathcal{M} : (!\sigma_1 \otimes !\sigma_2) \multimap \tau} \ (\multimap I^I_\otimes)$$

$$\frac{\Gamma_1 \mid \Delta \mid \Theta \vdash \mathcal{M}_1 : \sigma_1 \quad \Gamma_2 \mid \Delta \mid \Theta \vdash \mathcal{M}_2 : \sigma_2 \quad \Gamma_1 \# \Gamma_2}{\Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash \mathcal{M}_1 \otimes \mathcal{M}_2 : \sigma_1 \otimes \sigma_2} \ (\otimes I)$$

Table 2.2: Rules for the $\otimes$ connective.

Basic and composite operations on datatypes First, let us observe that Scott words allow for an easy definition of the basic operations on binary words and of the `case` function:

$$\begin{array}{lcl}
\mathtt{cons}^0 & \stackrel{\text{def}}{=} & \lambda w.\lambda s_0.\lambda s_1.\lambda x.s_0 w : \mathrm{W_S} \multimap \mathrm{W_S} \\
\mathtt{cons}^1 & \stackrel{\text{def}}{=} & \lambda w.\lambda s_0.\lambda s_1.\lambda x.s_1 w : \mathrm{W_S} \multimap \mathrm{W_S} \\
\mathtt{tail} & \stackrel{\text{def}}{=} & \lambda w.w\ I\ I\ \mathtt{nil} : \mathrm{W_S} \multimap \mathrm{W_S} \\
\mathtt{case} & \stackrel{\text{def}}{=} & \lambda s_0.\lambda s_1.\lambda x.\lambda w.w s_0 s_1 x : \forall \mathtt{a}.(\mathrm{W_S} \multimap \mathtt{a}) \multimap (\mathrm{W_S} \multimap \mathtt{a}) \multimap \mathtt{a} \multimap (\mathrm{W_S} \multimap \mathtt{a})
\end{array}$$

For Church binary words, it is possible to define the following terms:

$$\begin{array}{lcl}
\mathtt{length} & \stackrel{\text{def}}{=} & \lambda w.\lambda^! s.(\lambda^! x.!(\lambda z.xz))(wss) : \mathrm{W} \multimap \mathrm{N} \\
\mathtt{conv} & \stackrel{\text{def}}{=} & \lambda w.(\lambda^! x.!(x\ \mathtt{nil}))(w\ !\mathtt{cons}^0\ !\mathtt{cons}^1) : \mathrm{W} \multimap !\mathrm{W_S}
\end{array}$$

The former returns the length of a Church binary word as a tally integer, while the latter turns a Church binary word into the corresponding Scott binary word.

Consider the following basic operations on Church binary words:

$$\begin{array}{lcl}
\mathtt{S}_0 & \stackrel{\text{def}}{=} & \lambda w.\lambda^! s_0.\lambda^! s_1.(\lambda^! y.!(\lambda x.y(s_0 x)))(w!s_0!s_1) : \mathrm{W} \multimap \mathrm{W} \\
\mathtt{S}_1 & \stackrel{\text{def}}{=} & \lambda w.\lambda^! s_0.\lambda^! s_1.(\lambda^! y.!(\lambda x.y(s_1 x)))(w!s_0!s_1) : \mathrm{W} \multimap \mathrm{W} \\
\mathtt{Z} & \stackrel{\text{def}}{=} & \lambda^! s_0.\lambda^! s_1.!(\lambda x.x) : \mathrm{W}
\end{array}$$

By combining such operations, we can define a program which "raises" the depth of a Church binary word, namely $\mathtt{coer} \stackrel{\text{def}}{=} \lambda w.(\lambda^! y.!(y\mathtt{Z}))(w!\mathtt{S}_0!\mathtt{S}_1) : \mathrm{W} \multimap !\mathrm{W}$, whose behavior is $\mathtt{coer}\ \underline{w} =! \underline{w}$. By applying Property 1 to such term, it is possible to build the program $\mathtt{coer}^k \stackrel{\text{def}}{=} \lambda^! w.\mathtt{coer}_k(\mathtt{coer}_{k-1}(...(\mathtt{coer}_1\ !w)...))$ of type $!\mathrm{W} \multimap !^{k+1}\mathrm{W}$ such that $\mathtt{coer}^k !\underline{w} =!^{k+1} \underline{w}$, for every $k \geq 1$ and for any Church word $\underline{w}$.

Moreover, we define $\vdash \mathtt{iter} :!\mathtt{A} \multimap !(\mathtt{A} \multimap \mathtt{A}) \multimap \mathrm{N} \multimap !\mathtt{A}$ as the term which, when applied to a base, a step function and a value n, iterates the step function n times

starting from the base:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\emptyset \mid \emptyset \mid x : \mathtt{A} \multimap \mathtt{A}, b : \mathtt{A}, s : \mathtt{A} \multimap \mathtt{A} \vdash x\ b : \mathtt{A}}\ (!)}{\emptyset \mid x :!(\mathtt{A} \multimap \mathtt{A}), \Delta \mid \emptyset \vdash !(x\ b) :!\mathtt{A}}\ (\multimap I)}{\emptyset \mid \Delta \mid \emptyset \vdash \lambda^! x.!(x\ b) :!(\mathtt{A} \multimap \mathtt{A}) \multimap !\mathtt{A}} \qquad \dfrac{\vdots}{y : \mathrm{N} \mid \Delta \mid \emptyset \vdash y\ !s :!(\mathtt{A} \multimap \mathtt{A})}\ (\multimap E)}{\dfrac{y : \mathrm{N} \mid \Delta \mid \emptyset \vdash (\lambda^! x.!(x\ b))(y\ !s) :!\mathtt{A}}{\emptyset \mid \Delta \mid \emptyset \vdash \lambda y.(\lambda^! x.!(x\ b))(y\ !s) : \mathrm{N} \multimap !\mathtt{A}}\ (\multimap I^L)}{\vdash \lambda^! b.\lambda^! s.\lambda y.(\lambda^! x.!(x\ b))(y\ !s) :!\mathtt{A} \multimap !(\mathtt{A} \multimap \mathtt{A}) \multimap \mathrm{N} \multimap !\mathtt{A}}\ (\multimap I^I)$$

where $\Delta = b :!\mathtt{A}, s :!(\mathtt{A} \multimap \mathtt{A})$. It is easy to check that, given terms `base` and `step`, the terms $(\lambda^! b.\lambda^! s.\lambda y.(\lambda^! x.!(x\ b))(y\ !s))$ `base step` $\underline{n}$ and $!(\mathtt{step}^n\ \mathtt{base})$ reduce to the same normal form.

2.2.3 Properties of typed $\lambda^!$-terms

In order to give a characterization of functions through programs of $\lambda^!$-calculus, we need to be able to extract the output result which is computed by the program on a given input: in particular, what we aim to prove is a sort of inverted Generation Lemma, in which the shape of the term can be inferred from its type, where both the term and the type satisfy some conditions of normalization and closure.

We start by examining under which conditions a $\lambda^!$-term with a ! type is either a ! term or an application:

Property 7. *Let $\Pi \triangleright \Gamma \mid \Delta \mid \emptyset \vdash \mathcal{M} :!\sigma$, where $\mathcal{M} \in \mathbf{nf}_0$. Then:*

i. $\Gamma = \emptyset$ implies $\mathcal{M} = !\mathcal{N}$ for some $\mathcal{N}$.

ii. if there is $x \in \mathrm{FV}(\mathcal{M})$ such that $\Gamma(x)$ is defined, then $\mathcal{M} = \mathcal{N}\mathcal{Q}$ for some $\mathcal{N}$,$\mathcal{Q}$.

Proof. Both points follow by induction on Π. Observe that the only rules which can assign a non-linear type are (Ax^P), (!) and $(\multimap E)$.

i. Let $\Gamma = \emptyset$.

Since the parking basis is empty, Π cannot be an application of (Ax^P).

If Π ends with an application of rule (!), then the proof is trivial.

Now let Π end with an application of rule $(\multimap E)$. Observe that $\mathcal{M} \in \mathbf{nf}_0$, so $\mathcal{M}$ is not a redex; moreover, the case $\mathcal{M} = x\mathcal{P}_1 ... \mathcal{P}_n$ is not possible, since by Property 5 x should occur either in the linear or in the parking basis, which are both empty. To rule out the last possibility, assume that $\mathcal{M}$ is a block and Π is

$$\dfrac{\Pi_1 \triangleright \emptyset \mid \Delta \mid \emptyset \vdash \lambda^! x.\mathcal{P} :!\tau \multimap !\sigma \quad \Pi_2 \triangleright \emptyset \mid \Delta \mid \emptyset \vdash \mathcal{N}_2 :!\tau}{\emptyset \mid \Delta \mid \emptyset \vdash (\lambda^! x.\mathcal{P})\mathcal{Q} :!\sigma}\ (\multimap E)$$

where $\mathcal{Q} \neq !\mathcal{N}$. However, by inductive hypothesis on Π_2 we know that $\mathcal{Q} = !\mathcal{N}$ for some $\mathcal{N}$: then $\mathcal{M}$ cannot be an application, so the last rule is not $(\multimap E)$.

ii. Let $\Gamma = \Gamma', x : \mathtt{A}$. Since the conclusion of rule (!) is such that no variable in the linear basis occurs in $\mathcal{M}$, Π cannot end with an application of rule (!): then Π ends with an application of rule $(\multimap E)$ and $\mathcal{M} = \mathcal{N}\mathcal{Q}$ for some $\mathcal{N}, \mathcal{Q}$.

□

Secondly, we identify some criteria according to which a term is *not* an application:

Lemma 16. *Let* $\Pi \triangleright \Gamma \mid \emptyset \mid \emptyset \vdash \mathcal{M} : \sigma$ *such that* $\mathcal{M} \in \mathbf{nf}_0$*:*

i. if $\Gamma(x) = \mathtt{a}$ *for every* $x \in \mathrm{FV}(\mathcal{M})$*, then* $\mathcal{M}$ *is not an application;*

ii. if $\sigma \neq \mathtt{a}$ *and for every* $x \in \mathrm{FV}(\mathcal{M})$ *either* $\Gamma(x) = \mathtt{a}$ *or* $\Gamma(x) = \mathtt{A} \multimap \mathtt{a}$*, then* $\mathcal{M}$ *is not an application.*

Proof. For both points we proceed by induction on Π.

The only non obvious case is that of Π ending with an application of rule $(\multimap E)$, followed by a (possibly empty) sequence δ of non-constructive rules: then $\mathcal{M} = \mathcal{M}_1\mathcal{M}_2$ and Π is

$$\dfrac{\dfrac{\Pi_1 \triangleright \Gamma_1 \mid \emptyset \mid \emptyset \vdash \mathcal{M}_1 : \rho \multimap \tau \quad \Pi_2 \triangleright \Gamma_2 \mid \emptyset \mid \emptyset \vdash \mathcal{M}_2 : \rho}{\Gamma_1, \Gamma_2 \mid \emptyset \mid \emptyset \vdash \mathcal{M}_1\mathcal{M}_2 : \tau}\,(\multimap E)}{\Gamma_1, \Gamma_2 \mid \emptyset \mid \emptyset \vdash \mathcal{M}_1\mathcal{M}_2 : \sigma}\,\delta$$

By inductive hypothesis $\mathcal{M}_1$ is not an application; moreover, $\mathcal{M}_1$ cannot be a linear abstraction, since by hypothesis $\mathcal{M} \in \mathbf{nf}_0$. Assume $\mathcal{M}_1 = \lambda^! x.\mathcal{P}$, with $\rho = !\mu$ for some μ. If $\Gamma_2 = \emptyset$, then by Property 7.i $\mathcal{M}_2 = !\mathcal{Q}$: but this contradicts the hypothesis, since $\mathcal{M} \in \mathbf{nf}_0$. Otherwise, by Property 7.ii $\mathcal{M}_2$ is an application: but this contradicts the inductive hypothesis, since every premise in Γ_2 has type $\mathtt{a}$ or $\mathtt{A} \multimap \mathtt{a}$ and $!\mu \neq \mathtt{a}$, thus neither $\mathcal{M}_2$ nor $\mathcal{M}$ are applications. Finally, assume $\mathcal{M}_1$ to be a variable y:

i. in the first case, since all variables in Γ_1 have type $\mathtt{a}$, $\mathcal{M}_1$ cannot be a variable;

ii. in the second case, since all variables in Γ_1 have either type $\mathtt{a}$ or $\mathtt{A} \multimap \mathtt{a}$ and $\rho \multimap \tau$ is an arrow type, then $\rho \multimap \tau = \mathtt{A} \multimap \mathtt{a}$: but this would mean $\tau = \mathtt{a} = \sigma$ (δ being the empty sequence), which in turn would contradict the hypothesis that $\sigma \neq \mathtt{a}$, so $\mathcal{M}_1$ cannot be a variable.

□

With these ingredients, we are finally able to prove the *reading property* of B and $\mathrm{W_S}$, namely the fact that every derivation assigning such datatype is inhabited by a corresponding data, where both the type and the term are at depth $k \geq 0$:

Property 8 (Reading property of B and $\mathrm{W_S}$)**.**

i. *If* $\vdash \mathcal{M} :!^k\mathrm{B}$ *and* $\mathcal{M} \in \mathbf{nf}_k$*, then* $\mathcal{M} =!^k\mathtt{true}$ *or* $!^k\mathtt{false}$*, for* $k \geq 0$*.*

ii. *If* $\Gamma \mid \emptyset \mid \emptyset \vdash \mathcal{M} :!^k\mathrm{W_S}$ *and* $\mathcal{M} \in \mathbf{nf}_k$*, such that for each* $x \in \mathrm{dom}(\Gamma)$ *either* $\Gamma(x) = \mathtt{a}$ *or* $\Gamma(x) = \mathtt{A} \multimap \mathtt{a}$*, then* $\mathcal{M} =!^k\widehat{w}$ *for some Scott binary word* $\widehat{w}$*, for* $k \geq 0$*.*

Proof. Assume Π to be a clean derivation by Lemma 15: both points are proved by induction on k.

i. Let $k = 0$. By Lemma 16, $\mathcal{M}$ is not an application; moreover, $\mathcal{M}$ being closed implies that it cannot be a variable: then either $\mathcal{M} = \lambda x.\mathcal{N}$ or $\lambda^! x.\mathcal{N}$, so by Lemma 11 the derivation is

$$\dfrac{\dfrac{\Pi \triangleright \Gamma \mid \Delta \mid \emptyset \vdash \mathcal{N} : \tau}{\emptyset \mid \emptyset \mid \emptyset \vdash \mathcal{M} : \sigma \multimap \tau}\ (\multimap I^*)}{\emptyset \mid \emptyset \mid \emptyset \vdash \mathcal{M} : \forall \mathtt{a}.\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a}}\ \delta$$

where either the rule is $(\multimap I^L)$, with $\Gamma = x : \sigma$ and $\Delta = \emptyset$, or the rule is $(\multimap I^I)$, with $\Gamma = \emptyset$ and $\Delta = x : \sigma$, where δ is a sequence of non-constructive rules. Since the derivation is clean, δ is an application of rule $(\forall I)$, so $\sigma \multimap \tau = \mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a}$ and $\Pi \triangleright x : \mathtt{a} \mid \emptyset \mid \emptyset \vdash \mathcal{N} : \mathtt{a} \multimap \mathtt{a}$ is followed by an application of rule $(\multimap I^L)$.
Again by Lemma 16, $\mathcal{N}$ cannot be an application; moreover, $\mathcal{N}$ cannot be a variable because of the context: then either $\mathcal{N} = \lambda y.\mathcal{P}$ or $\mathcal{N} = \lambda^! y.\mathcal{P}$. By repeating the same reasoning as before, $\mathcal{N} = \lambda y.\mathcal{P}$ and Π is obtained by applying rule $(\multimap I)$ to $\Pi' \triangleright x : \mathtt{a}, y : \mathtt{a} \mid \emptyset \mid \emptyset \vdash \mathcal{P} : \mathtt{a}$.
By Lemma 16.i, $\mathcal{P}$ cannot be an application; moreover $\mathcal{P}$ cannot be an abstraction, otherwise $x : \mathtt{a}, y : \mathtt{a} \mid \emptyset \mid \emptyset \vdash \mathcal{P} : \mathtt{a}$ would be obtained by applying either rule $(\multimap I^L)$ or rule $(\multimap I^I)$ to Π', followed by a sequence of non-constructive rules: but both abstraction rules derive an arrow type, whereas $\mathtt{a}$ cannot be obtained from $\sigma \multimap \tau$ by means of applications of non-constructive rules.
Therefore $\mathcal{P}$ is a variable, so either $\mathcal{M} = \lambda x.\lambda y.x$ or $\mathcal{M} = \lambda x.\lambda y.y$.

Now let $k = i + 1$. By Property 7.i, $\mathcal{M} =!\mathcal{N}$; moreover, by Lemma 11 the derivation ends with an application of rule (!) to $\emptyset \mid \emptyset \mid \emptyset \vdash \mathcal{N} :!^i\mathrm{B}$. By inductive hypothesis either $\mathcal{N} =!^i\mathtt{true}$ or $\mathcal{N} =!^i\mathtt{false}$: then the proof follows from the fact that $\mathcal{M} =!\mathcal{N}$, so either $\mathcal{M} =!^{i+1}\mathtt{true}$ or $\mathcal{M} =!^{i+1}\mathtt{false}$.

ii. Let $k = 0$. By Lemma 16.ii, $\mathcal{M}$ is not an application; moreover, $\mathcal{M}$ cannot be a variable because of the constraints on the type of the variables in Γ: then either $\mathcal{M} = \lambda f_0.\mathcal{N}$ or $\lambda^! f_0.\mathcal{N}$. By Lemma 11, the derivation is

$$\dfrac{\dfrac{\Pi \triangleright \Gamma' \mid \Delta' \mid \emptyset \vdash \mathcal{N} : \tau}{\Gamma \mid \emptyset \mid \emptyset \vdash \mathcal{M} : \sigma \multimap \tau}\ (\multimap I^*)}{\Gamma \mid \emptyset \mid \emptyset \vdash \mathcal{M} : \mu\mathtt{b}.\forall \mathtt{a}.(\mathtt{b} \multimap \mathtt{a}) \multimap (\mathtt{b} \multimap \mathtt{a}) \multimap (\mathtt{a} \multimap \mathtt{a})}\ \delta$$

where the last rule is either $(\multimap I^L)$, with $\Gamma' = \Gamma, f_0 : \sigma$ and $\Delta = \emptyset$, or $(\multimap I^I)$, with $\Gamma' = \Gamma$ and $\Delta = f_0 : \sigma$, where δ is a sequence of applications of non-constructive rules: then, since the derivation is clean, δ is an application of rule $(\forall I)$ followed by an application of rule (μI), so $\sigma \multimap \tau = (\mathrm{W_S} \multimap \mathtt{a}) \multimap (\mathrm{W_S} \multimap \mathtt{a}) \multimap (\mathtt{a} \multimap \mathtt{a})$ and $\Pi \triangleright f_0 : \mathrm{W_S} \multimap \mathtt{a}, \Gamma \mid \emptyset \mid \emptyset \vdash \mathcal{N} : (\mathrm{W_S} \multimap \mathtt{a}) \multimap (\mathtt{a} \multimap \mathtt{a})$ is followed by an application of rule $(\multimap I^L)$: observe that $\mathtt{a} \notin \mathrm{FTV}(\Gamma)$ because of the side condition of rule $(\forall I)$.

By Lemma 16.ii, $\mathcal{N}$ cannot be an application; moreover, given the basis Γ, $\mathcal{N}$ cannot be a variable: then $\mathcal{N}$ is an abstraction. By repeating the same reasoning as before, $\mathcal{N} = \lambda f_1.\mathcal{P}$ and Π is obtained by applying rule $(\multimap I^L)$ to $\Pi' \triangleright f_0 : \mathrm{W_S} \multimap \mathtt{a}, f_1 : \mathrm{W_S} \multimap \mathtt{a}, \Gamma \mid \emptyset \mid \emptyset \vdash \mathcal{P} : \mathtt{a} \multimap \mathtt{a}$.

By following exactly the same steps, $\mathcal{P} = \lambda x.\mathcal{Q}$ and Π' is obtained by applying rule $(\multimap I^L)$ to $\Pi'' \triangleright f_0 : \mathrm{W_S} \multimap \mathtt{a}, f_1 : \mathrm{W_S} \multimap \mathtt{a}, x : \mathtt{a}, \Gamma \mid \emptyset \mid \emptyset \vdash \mathcal{Q} : \mathtt{a}$.

Notice that $\mathcal{Q}$ cannot be an abstraction, since it has a variable type, so $\mathcal{Q}$ is either a variable or an application. In the former case, since $\mathtt{a}$ does not occur free in Γ, the only variable that can have type $\mathtt{a}$ is x, so $\mathcal{M} = \lambda f_0.\lambda f_1.\lambda x.x = \widehat{\epsilon}$.

In the latter case, $\mathcal{Q} = \mathcal{Q}_1\mathcal{Q}_2$ and Π'' is obtained by applying rule $(\multimap E)$ to $\Gamma_1 \mid \emptyset \mid \emptyset \vdash \mathcal{Q}_1 : \sigma \multimap \mathtt{a}$, $\Gamma_2 \mid \emptyset \mid \emptyset \vdash \mathcal{Q}_2 : \sigma$, where $\Gamma_1, \Gamma_2 = f_0 : \mathrm{W_S} \multimap \mathtt{a}, f_1 : \mathrm{W_S} \multimap \mathtt{a}, x : \mathtt{a}, \Gamma$. Since by hypothesis $\mathcal{Q} \in \mathbf{nf}_0$, we know that $\mathcal{Q}_1$ is not a linear abstraction; moreover, by Lemma 16.ii $\mathcal{Q}_1$ is not an application: then $\mathcal{Q}_1$ is a variable and the derivation is

$$\dfrac{\dfrac{}{\Gamma_1 \mid \emptyset \mid \emptyset \vdash f_i : \mathrm{W_S} \multimap \mathtt{a}}\,(Ax^L) \qquad \Sigma \triangleright \Gamma_2 \mid \emptyset \mid \emptyset \vdash \mathcal{Q}_2 : \mathrm{W_S}}{\Gamma_1, \Gamma_2 \mid \emptyset \mid \emptyset \vdash f_i\mathcal{Q}_2 : \mathtt{a}}\,(\multimap E)$$

for $i \in \{0, 1\}$. By induction hypothesis on Σ, $\mathcal{Q}_2$ is a binary word $\hat{w}$: then either $\mathcal{M} = \lambda f_0.\lambda f_1.\lambda x.f_0\widehat{w} = \widehat{0w}$ or $\mathcal{M} = \lambda f_0.\lambda f_1.\lambda x.f_1\widehat{w} = \widehat{1w}$.

Now let $k = i + 1$. By applying Property 7.i we know that $\mathcal{M} =!\mathcal{N}$; moreover, by Lemma 11 the derivation ends with an application of rule (!) to $\emptyset \mid \emptyset \mid \emptyset \vdash \mathcal{N} :!^i\mathrm{W_S}$. By inductive hypothesis $\mathcal{N} =!^i\widehat{w}$ for some Scott binary word $\widehat{w}$: then the proof follows from the fact that $\mathcal{M} =!\mathcal{N} =!^{i+1}\widehat{w}$.

□

2.2.4 Complexity soundness

We are interested in giving a precise account of the hierarchy of classes characterized by the typed $\lambda^!$-calculus. In particular, we want to show that it is possible to represent exactly the class $\mathtt{k\text{-}EXP}$ by considering a subclass of typed $\lambda^!$-terms, whose type depends on the parameter k. Before presenting the main complexity result, we briefly introduce the notations used to identify the hierarchy of complexity classes.

Complexity classes

Let $\texttt{FDTIME}(f(n))$ and $\texttt{DTIME}(f(n))$ be respectively the class of functions and the class of predicates on binary words computable on a deterministic Turing machine in time $O(f(n))$. The complexity classes that we wish to capture are the following ones:

$$\begin{array}{rcll} \texttt{k-EXP} & = & \cup_{i\in\mathbb{N}}\texttt{DTIME}(2_k^{n^i}) & \text{for } k \geq 0 \\ \texttt{k-FEXP} & = & \cup_{i\in\mathbb{N}}\texttt{FDTIME}(2_k^{n^i}) & \text{for } k \geq 0 \end{array}$$

In particular, observe that the classes of polynomial time predicates and polynomial time functions are defined respectively by $\texttt{PTIME} = \cup_{i\in\mathbb{N}}\texttt{DTIME}(n^i) = \texttt{0-EXP}$ and $\texttt{FPTIME} = \cup_{i\in\mathbb{N}}\texttt{FDTIME}(n^i) = \texttt{0-FEXP}$.

Soundness result

In this subsection we want to study the relation between the type of a program representing a function and the complexity class to which such function belongs. In particular, we show that a closed term of type $!\mathrm{W} \multimap !^{k+2}\mathrm{B}$ and a closed term of type $!\mathrm{W} \multimap !^{k+2}\mathrm{W_S}$ represent respectively a predicate and a function which, when applied to a word of length n, can be evaluated in time $O(2_k^{p(n)})$ for some polynomial p.

Let $F(\sigma)$ denote the class of functions represented by programs of type σ. First, we prove that $F(!\mathrm{W} \multimap !^{k+2}\mathrm{B}) \subseteq \texttt{k-EXP}$, namely, that every program having type $!\mathrm{W} \multimap !^{k+2}\mathrm{B}$ represents (in the sense of Section 2.1.5) a predicate of $\texttt{k-EXP}$:

Theorem 4. *Let $\vdash \mathcal{P} : !\mathrm{W} \multimap !^{k+2}\mathrm{B}$ and $\vdash \underline{w} : \mathrm{W}$, where $\mathcal{P}$ is a program and $\mathbf{length}(w) = n$; then the reduction $\mathcal{P}!\underline{w} \xrightarrow{*} !^{k+2}\mathcal{D}$ can be computed in time $2_k^{p(n)}$, where $\mathcal{D}$ is either $\texttt{true}$ or $\texttt{false}$ and p is a polynomial.*

Proof. Let $\mathcal{M}'$ be the normal form of $\mathcal{P}!\underline{w}$. By Property 4, we know that $\overline{\mathcal{P}!\underline{w}}^{k+2}$ can be reduced to a term $\mathcal{N} \in \mathbf{nf}_{k+2}$ in time $O(2_k^{p(n)})$ on a Turing machine, where $n = \mathbf{length}(w)$. Moreover $\overline{\mathcal{M}'}^{k+2} = \mathcal{N}$ by combining Lemma 9.i and Property 1.ii. Since $\mathcal{P}!\underline{w}$ has type $!^{k+2}\mathrm{B}$, by Theorem 3 $\mathcal{M}'$ is a closed term of type $!^{k+2}\mathrm{B}$: then by Property 8.i $\mathcal{M}'$ is either $!^{k+2}\texttt{true}$ or $!^{k+2}\texttt{false}$.
Since $\mathcal{N} = \overline{\mathcal{M}'}^{k+2} = \mathcal{M}'$, $\mathcal{P}!\underline{w}$ can be computed in time $O(2_k^{p(n)})$.

□

The soundness result can be extended to functions in order to show that $F(!\mathrm{W} \multimap !^{k+2}\mathrm{W_S}) \subseteq \texttt{k-FEXP}$, that is, every program of type $!\mathrm{W} \multimap !^{k+2}\mathrm{W_S}$ represents a function of $\texttt{k-FEXP}$. This can be easily proved by retracing the same steps of the previous proof and by applying the reading property of Scott binary words in order to extract the output:

Theorem 5. *Let $\vdash \mathcal{P} : !\mathrm{W} \multimap !^{k+2}\mathrm{W_S}$ and $\vdash \underline{w} : \mathrm{W}$, where $\mathcal{P}$ is a program and $\mathbf{length}(w) = n$; then the reduction $\mathcal{P}!\underline{w} \xrightarrow{*} !^{k+2}\widehat{w'}$ can be computed in time $2_k^{p(n)}$, where $\widehat{w'}$ is a Scott binary word and p is a polynomial.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Pi \quad \emptyset \mid \emptyset \mid w : \mathrm{W} \vdash \mathtt{coer}\ \underline{w} : !\mathrm{W}}{\emptyset \mid \emptyset \mid w : \mathrm{W} \vdash \mathtt{iter}(\mathtt{coer}\ \underline{w}) : !(\mathrm{W} \multimap \mathrm{W}) \multimap \mathrm{N} \multimap !\mathrm{W}}\ (\multimap E) \quad \Sigma}{\emptyset \mid \emptyset \mid w : \mathrm{W} \vdash \mathtt{iter}(\mathtt{coer}\ \underline{w})(\mathtt{!lenX2}) : \mathrm{N} \multimap !\mathrm{W}}\ (\multimap E) \quad \dfrac{\emptyset \mid \emptyset \mid w : \mathrm{W} \vdash (\mathtt{length}\ \underline{w}) : \mathrm{N}}{}}{\emptyset \mid \emptyset \mid w : \mathrm{W} \vdash \mathtt{iter}(\mathtt{coer}\ \underline{w})(\mathtt{!lenX2})(\mathtt{length}\ \underline{w}) : !\mathrm{W}}\ (\multimap E)}{\emptyset \mid w : !\mathrm{W} \mid \emptyset \vdash !(\mathtt{iter}(\mathtt{coer}\ \underline{w})(\mathtt{!lenX2})(\mathtt{length}\ \underline{w})) : !^2\mathrm{W}}\ (!)}{\vdash \lambda^! w.!(\mathtt{iter}(\mathtt{coer}\ \underline{w})(\mathtt{!lenX2})(\mathtt{length}\ \underline{w})) : !\mathrm{W} \multimap !^2\mathrm{W}}\ (\multimap I)$$

where

$$\Pi \quad \triangleright \quad \emptyset \mid \emptyset \mid w : \mathrm{W} \vdash \mathtt{iter} : !\mathrm{W} \multimap !(\mathrm{W} \multimap \mathrm{W}) \multimap \mathrm{N} \multimap !\mathrm{W}$$
$$\Sigma \quad \triangleright \quad \emptyset \mid \emptyset \mid w : \mathrm{W} \vdash !\mathtt{lenX2} : !(\mathrm{W} \multimap \mathrm{W})$$

Figure 2.4: Term of type $!\mathrm{W} \multimap !^2\mathrm{W}$ representing an exponential function.

Proof. We proceed exactly as before, up to the application of the reading property: by Property 8.ii, the output of the computation is $!^{k+2}\widehat{w'}$, where $\widehat{w'}$ is a Scott binary word, and the evaluation of $\mathcal{P}!\underline{w}$ is done in time $O(2_k^{p(n)})$.

□

One might wonder why we should use Scott binary words as the output type for the representation of functions, instead of Church binary words, as the latter would seem like a more obvious direction to take.
Considering the case of $k = 0$, the reason behind such refusal is easily explained by the fact that Church binary words are data of depth 1: from the point of view of expressivity, there are some FPTIME functions which are not captured by terms of type $!\mathrm{W} \multimap !\mathrm{W}$, while programs of type $!\mathrm{W} \multimap !^2\mathrm{W}$ allow also the typing of functions outside the FPTIME class, like exponential functions.
As a proof of the latter claim, let $\mathtt{lenX2} \stackrel{\text{def}}{=} \lambda w.\lambda^! f_0.\lambda^! f_1.(\lambda^! y.!(\lambda x.y(yx)))(w f_0 f_1)$ be the program of type $\mathrm{W} \multimap \mathrm{W}$ whose behavior is that of doubling the length of a binary word: then we can easily build $\lambda^! w.!(\mathtt{iter}(\mathtt{coer}\ \underline{w})(\mathtt{!lenX2})(\mathtt{length}\ \underline{w}))$ of type $!\mathrm{W} \multimap !^2\mathrm{W}$ (see Figure 2.4) which, when supplied with a binary word $!\underline{w}$, returns a word of length 2^n, where $n = \mathbf{length}(w)$.

2.2.5 Completeness

In Section 2.2.4 we proved both $F(!\mathrm{W} \multimap !^{k+2}\mathrm{B}) \subseteq$ k-EXP and $F(!\mathrm{W} \multimap !^{k+2}\mathrm{W_S}) \subseteq$ k-FEXP: we now want to strengthen these results by examining the converse inclusions, namely, by showing that for every predicate of k-EXP (resp. function of k-FEXP) there is *at least* one program of type $!\mathrm{W} \multimap !^{k+2}\mathrm{B}$ (resp. $!\mathrm{W} \multimap !^{k+2}\mathrm{W_S}$) representing it.

In order to achieve such results, we simulate time-bounded Turing machines through terms of $\lambda^!$-calculus; we then prove that the computation of a desired function can be performed through a term of $\lambda^!$-calculus respecting the given time

complexity bound. We start by showing how to encode polynomials on tally integers, which are needed to bound the iterations of the machine; we then proceed to define the encoding of a Turing machine itself through $\lambda^!$-calculus.

Encoding polynomials

We begin by proving the functoriality of the ! modality, which is needed to define coercions of some key $\lambda^!$-terms:

Proposition 1. *Let $\vdash \mathcal{M} : \sigma_1 \multimap ... \multimap \sigma_n \multimap \tau$ for some $n > 0$: then there is a term $\mathcal{M}_k$ such that $\vdash \mathcal{M}_k : !^k\sigma_1 \multimap ... \multimap !^k\sigma_n \multimap !^k\tau$, and $\mathcal{M}_k!^k\mathcal{P}_1...!^k\mathcal{P}_n$ and $!^k(\mathcal{M}\mathcal{P}_1...\mathcal{P}_n)$ have the same normal form, for any closed term $\mathcal{P}_i$ ($1 \leq i \leq n$), for any $k \geq 1$.*

Proof. By induction on k.

Let $k = 1$. By applying n times rule $(\multimap E)$ to $\vdash \mathcal{M} : \sigma_1 \multimap ... \multimap \sigma_n \multimap \tau$ and to the parking axioms $\emptyset \mid \emptyset \mid x_i : \sigma_i \vdash x_i : \sigma_i$ $(1 \leq i \leq n)$, we obtain the proof $\emptyset \mid \emptyset \mid x_1 : \sigma_1, ..., x_n : \sigma_n \vdash \mathcal{M}x_1...x_n : \tau$. By applying rule (!) to such derivation, followed by n applications of rule $(\multimap I^I)$, we obtain $\vdash \lambda^!x_1...x_n.!(\mathcal{M}x_1...x_n) : !\sigma_1 \multimap ...!\sigma_n \multimap !\tau$. Observe that $(\lambda^!x_1...x_n.!(\mathcal{M}x_1...x_n))!\mathcal{P}_1...!\mathcal{P}_n$ and $!(\mathcal{M}\mathcal{P}_1...\mathcal{P}_n)$ have the same normal form, since the former can be easily reduced to the latter by n reduction steps:

$$\begin{array}{lll} (\lambda^!x_1...x_n.!(\mathcal{M}x_1...x_n))!\mathcal{P}_1...!\mathcal{P}_n & \to & (\lambda^!x_2...x_n.!(\mathcal{M}\mathcal{P}_1x_2...x_n))!\mathcal{P}_2...!\mathcal{P}_n \\ & \overset{*}{\to} & (\lambda^!x_n.!(\mathcal{M}\mathcal{P}_1\mathcal{P}_2...\mathcal{P}_{n-1}x_n))!\mathcal{P}_n \\ & \to & !(\mathcal{M}\mathcal{P}_1...\mathcal{P}_n) \end{array}$$

The inductive case follows easily.

□

The multiplication and the sum of two tally integers are given respectively by $\vdash \mathtt{mult} : \mathrm{N} \multimap \mathrm{N} \multimap \mathrm{N}$ and $\vdash \mathtt{add} : \mathrm{N} \multimap \mathrm{N} \multimap \mathrm{N}$, where $\mathtt{mult} \overset{\text{def}}{=} \lambda n.\lambda m.\lambda^!s.n(m!s)$ and $\mathtt{add} \overset{\text{def}}{=} \lambda n.\lambda m.\lambda^!s.(\lambda^!x.\lambda^!y.!(\lambda z.x(yz)))(n!s)(m!s)$.

Example 7. *The type derivation for* add *is the following:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Sigma \rhd \emptyset \mid \emptyset \mid s : \mathtt{a} \multimap \mathtt{a}, x : \mathtt{a} \multimap \mathtt{a}, y : \mathtt{a} \multimap \mathtt{a} \mid \emptyset \vdash \lambda z.x(yz) : \mathtt{a} \multimap \mathtt{a}}{\emptyset \mid s :!(\mathtt{a} \multimap \mathtt{a}), x :!(\mathtt{a} \multimap \mathtt{a}), y :!(\mathtt{a} \multimap \mathtt{a}) \mid \emptyset \vdash !(\lambda z.x(yz)) :!(\mathtt{a} \multimap \mathtt{a})}\,(!)}{\emptyset \mid s :!(\mathtt{a} \multimap \mathtt{a}) \mid \emptyset \vdash \lambda^!x.\lambda^!y.!(\lambda z.x(yz)) :!(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a})}\,(\multimap I^I) \quad \Pi_n}{n : \mathrm{N} \mid s :!(\mathtt{a} \multimap \mathtt{a}) \mid \emptyset \vdash (\lambda^!x.\lambda^!y.!(\lambda z.x(yz)))(n!s) :!(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a})}\,(\multimap E) \quad \Pi_m}{n : \mathrm{N}, m : \mathrm{N} \mid s :!(\mathtt{a} \multimap \mathtt{a}) \mid \emptyset \vdash (\lambda^!x.\lambda^!y.!(\lambda z.x(yz)))(n!s)(m!s) :!(\mathtt{a} \multimap \mathtt{a})}\,(\multimap E)}{n : \mathrm{N}, m : \mathrm{N} \mid \emptyset \mid \emptyset \vdash \lambda^!s.(\lambda^!x.\lambda^!y.!(\lambda z.x(yz)))(n!s)(m!s) :!(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a})}\,(\multimap I^I)}{n : \mathrm{N}, m : \mathrm{N} \mid \emptyset \mid \emptyset \vdash \lambda^!s.(\lambda^!x.\lambda^!y.!(\lambda z.x(yz)))(n!s)(m!s) : \mathrm{N}}\,(\forall I)}{\vdash \lambda n.\lambda m.\lambda^!s.(\lambda^!x.\lambda^!y.!(\lambda z.x(yz)))(n!s)(m!s) : \mathrm{N} \multimap \mathrm{N} \multimap \mathrm{N}}\,(\multimap I^L)$$

where $\Pi_n \rhd n : \mathrm{N} \mid s :!(\mathtt{a} \multimap \mathtt{a}) \mid \emptyset \vdash n!s :!(\mathtt{a} \multimap \mathtt{a})$, $\Pi_m \rhd m : \mathrm{N} \mid s :!(\mathtt{a} \multimap \mathtt{a}) \mid \emptyset \vdash m!s :!(\mathtt{a} \multimap \mathtt{a})$ and Σ are easy to obtain.

From this point forward, for any given $\mathcal{M}$ of type $\sigma_1 \multimap ... \multimap \sigma_n \multimap \tau$ we denote by $\mathcal{M}_k$ the term of type $!^k\sigma_1 \multimap ... \multimap !^k\sigma_n \multimap !^k\tau$ such that $!^k(\mathcal{M}\ \mathcal{P}_1...\mathcal{P}_n)$ and $\mathcal{M}_k!^k\mathcal{P}_1...!^k\mathcal{P}_n$ reduce to the same normal form, by implicitly applying Property 1.

By composing addition and multiplication with the term representing the exponential function, it is possible to represent all the functions of the form $2_k^{q(n)}$ on tally integers, for some polynomial $q(n)$ and $k \geq 0$:

Lemma 17. *If p is a polynomial over one variable with coefficients in $\mathbb{N}$, then*

i) there is $\mathcal{M}$ representing $p(n)$ such that $\vdash \mathcal{M} :!\mathrm{N} \multimap !\mathrm{N}$;

ii) there is $\mathcal{M}$ representing $2_k^{p(n)}$ such that $\vdash \mathcal{M} :!\mathrm{N} \multimap !^{k+1}\mathrm{N}$, for any $k \geq 1$.

Proof. i) Given $\Pi_{\mathtt{add}} \triangleright \vdash \mathtt{add} : \mathrm{N} \multimap \mathrm{N} \multimap \mathrm{N}$ and $\Pi_{\mathtt{mult}} \triangleright \vdash \mathtt{mult} : \mathrm{N} \multimap \mathrm{N} \multimap \mathrm{N}$, there is a term $\mathcal{M}$, obtained by composing `add` and `mult` in a suitable way, representing the polynomial $q(n)$, such that $\mathcal{M} :!\mathrm{N} \multimap !\mathrm{N}$. Indeed, let $\Pi \triangleright \emptyset \mid \emptyset \mid x : \mathrm{N} \vdash \mathcal{N} : \mathrm{N}$ be a composition of `add` and `mult` with argument x; then the derivation for $\mathcal{M}$ is

$$\dfrac{\dfrac{\Pi \triangleright \emptyset \mid \emptyset \mid x : \mathrm{N} \vdash \mathcal{N} : \mathrm{N}}{\emptyset \mid x :!\mathrm{N} \mid \emptyset \vdash !\mathcal{N} :!\mathrm{N}}\ (!)}{\vdash \lambda^! x.!\mathcal{N} :!\mathrm{N} \multimap !\mathrm{N}}\ (\multimap I^I)$$

As an example, the polynomial function $q(n) = n^2$ is represented by the program $\vdash \lambda^! x.!(\mathtt{mult}\ x\ x) :!\mathrm{N} \multimap !\mathrm{N}$.

ii) The function $f(n) = 2n$ is represented by $\Pi_{\mathtt{double}} \triangleright \vdash \mathtt{double} \stackrel{\mathrm{def}}{=} \mathtt{mult}\ \underline{2} : \mathrm{N} \multimap \mathrm{N}$; then the function 2^n can be represented by iterating `double`, so the resulting derivation $\Pi_{\mathtt{exp}}$ is

$$\dfrac{\dfrac{\vdots}{\triangleright \vdash \lambda^! y.!(y\ \underline{1}) :!(\mathrm{N} \multimap \mathrm{N}) \multimap !\mathrm{N} \quad x : \mathrm{N} \mid \emptyset \mid \emptyset \vdash x\ !\mathtt{double} :!(\mathrm{N} \multimap \mathrm{N})}\ (\multimap E)}{\vdash \mathtt{exp} \stackrel{\mathrm{def}}{=} \lambda x.(\lambda^! y.!(y\ \underline{1}))(x\ !\mathtt{double}) : \mathrm{N} \multimap !\mathrm{N}}\ (\multimap I^L)$$

Let $\Pi_{\mathtt{exp^k}} \triangleright \vdash \mathtt{exp^k} : \mathrm{N} \multimap !^k\mathrm{N}$, where $\mathtt{exp^k}n = \mathtt{exp}_{k-1}(\mathtt{exp}_{k-2}(...(\mathtt{exp}_1(\mathtt{exp}\ n))...))$ is the program representing function 2_k^n, for every $k \geq 0$.
The derivation obtained by applying Property 1 to $\Pi_{\mathtt{exp^k}}$ is composed with $\vdash \mathcal{M} :!\mathrm{N} \multimap !\mathrm{N}$, where $\mathcal{M}$ represents the polynomial $q(n)$; then the term of type $!\mathrm{N} \multimap !^{k+1}\mathrm{N}$ representing $2_k^{q(n)}$ is obtained in the following way:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\emptyset \mid \emptyset \mid x : \mathrm{N}, y : \mathrm{N} \vdash \mathtt{exp^k} y :!^k\mathrm{N}}\ (!)}{\emptyset \mid x :!\mathrm{N}, y :!\mathrm{N} \mid \emptyset \vdash !(\mathtt{exp^k} y) :!^{k+1}\mathrm{N}}\ (\multimap I^I)}{\emptyset \mid x :!\mathrm{N} \mid \emptyset \vdash \lambda^! y.!(\mathtt{exp^k} y) :!\mathrm{N} \multimap !^{k+1}\mathrm{N}} \quad \dfrac{\vdots}{\emptyset \mid x :!\mathrm{N} \mid \emptyset \vdash \mathcal{M}!x :!\mathrm{N}}}{\dfrac{\emptyset \mid x :!\mathrm{N} \mid \emptyset \vdash (\lambda^! y.!(\mathtt{exp^k} y))(\mathcal{M}x) :!^{k+1}\mathrm{N}}{\vdash \lambda^! x.(\lambda^! y.!(\mathtt{exp^k} y))(\mathcal{M}x) :!\mathrm{N} \multimap !^{k+1}\mathrm{N}}\ (\multimap I)}\ (\multimap E)}$$

□

Simulation of Turing machines

In the previous section we showed how to represent polynomials and exponentials on tally integers in $\lambda^!$-calculus: now, such results are employed to simulate k-EXP-time bounded Turing machines.

For simplicity, the notation for the tensor product can be generalized to the n-ary case, with $\mathcal{M}_1 \otimes ... \otimes \mathcal{M}_n \stackrel{\text{def}}{=} \lambda x.x\mathcal{M}_1...\mathcal{M}_n$ and $\lambda(x_1 \otimes ... \otimes x_n).\mathcal{N} \stackrel{\text{def}}{=} z\lambda x_1. ...\lambda x_n.\mathcal{N}$, for $n \geq 2$. If $\sigma_1 \otimes ... \otimes \sigma_n \stackrel{\text{def}}{=} \forall \mathtt{a}.(\sigma_1 \multimap ... \multimap \sigma_n \multimap \mathtt{a}) \multimap \mathtt{a}$, then the rules of Table 2.2 can be extended accordingly to $(\multimap I^L_{\otimes n})$, $(\multimap I^I_{\otimes n})$ and $(\otimes_n I)$.

Furthermore, we can generalize the boolean datatype to the n-ary case by defining $\mathrm{B}^n \stackrel{\text{def}}{=} \forall \mathtt{a}.\underbrace{\mathtt{a} \multimap ... \multimap \mathtt{a}}_{n} \multimap \mathtt{a}$, typing normal forms of the shape $\lambda x_1....\lambda x_n.x_i$ for some $i \in \{1, ..., n\}$ and $n \geq 1$; note that $\mathrm{B}^2 = \mathrm{B}$.

Configurations and transitions of a Turing machine Let $\mathfrak{M}$ be a Turing machine with alphabet $\Sigma = \{0, 1\}$ and a set of n states $Q_n = \{q_1, ..., q_n\}$. We represent the configurations of a one-tape Turing machine $\mathfrak{M}$ over a binary alphabet with n states through a term $\mathcal{N}_L \otimes \mathcal{P} \otimes \mathcal{N}_R \otimes \mathcal{P}$, having type $\mathrm{C} \stackrel{\text{def}}{=} \mathrm{W_S} \otimes \mathrm{B} \otimes \mathrm{W_S} \otimes \mathrm{B}^n$, where:

1. the first component of type $\mathrm{W_S}$ represents the portion of the tape on the left-hand side of the scanned symbol, in reverse order;
2. the second component of type B represents the currently scanned symbol;
3. the third component of type $\mathrm{W_S}$ represents the portion of the tape on the right-hand side of the scanned symbol;
4. the fourth and final component of type B^n represents the current state of the machine.

We define $\tilde{1}$ to be the first value of type B^n, where n is the number of states of $\mathfrak{M}$, which is conventionally chosen to represent the initial state of the machine.

As a last tool in order to prove the completeness result, we need to define three more terms, which allow us to simulate the operations of a Turing machine through $\lambda^!$-calculus:

Lemma 18 (Transitions of a Turing machine)**.** *Let $\mathfrak{M}$ be a one-tape deterministic Turing machine over a binary alphabet; then the following programs can be typed:*

i) $\vdash \mathtt{init} : \mathrm{W_S} \multimap \mathrm{C}$, *mapping a Scott binary word to the corresponding initial configuration of $\mathfrak{M}$;*

ii) $\vdash \mathtt{step} : \mathrm{C} \multimap \mathrm{C}$, *computing the next configuration of $\mathfrak{M}$ based on the current configuration;*

iii) $\vdash \texttt{accept} : \mathrm{C} \multimap \mathrm{B}$, *returning* `true` *(respectively* `false`*) if the state of the current configuration is accepting (respectively rejecting);*

iv) $\vdash \texttt{extract} : \mathrm{C} \multimap \mathrm{W_S}$, *returning the binary word written on the tape.*

Proof. We show how to define each of the described terms; the respective typings are trivial and thus their proof is omitted.

i. We define `init` as $\texttt{init} \stackrel{\text{def}}{=} \texttt{case}\ \mathcal{M}_1\mathcal{M}_2\mathcal{M}_3$, where $\mathcal{M}_1 = \lambda w.\texttt{nil} \otimes \texttt{true} \otimes w \otimes \tilde{1}$, $\mathcal{M}_2 = \lambda w.\texttt{nil} \otimes \texttt{false} \otimes w \otimes \tilde{1}$ and $\mathcal{M}_3 = \texttt{nil} \otimes \texttt{false} \otimes \texttt{nil} \otimes \tilde{1}$.

ii. The term `step` can be defined by retracing the transition function of $\mathfrak{M}$ and by doing a `case` distinction.

iii. We define $\texttt{accept} \stackrel{\text{def}}{=} \lambda(y_L \otimes x \otimes y_R \otimes s).s\mathcal{Q}_1 ... \mathcal{Q}_n$ where, for $1 \leq i \leq n$, $\mathcal{Q}_i = \texttt{true}$ (respectively $\mathcal{Q}_i = \texttt{false}$) if the state encoded by the i-th element of type B^n is accepting (respectively rejecting) for $\mathfrak{M}$.

iv. The last term, returning the right-hand tape portion of the tape, is trivially defined as $\texttt{extract} \stackrel{\text{def}}{=} \lambda(y_L \otimes x \otimes y_R \otimes s).y_R$.

□

Finally we have all the necessary ingredients to prove the main completeness result:

Theorem 6 (Extensional completeness)**.**

i. Let f be a binary predicate in k-`EXP`*, for any $k \geq 0$; then there is a term $\mathcal{M}$ representing f such that $\vdash \mathcal{M} : !\mathrm{W} \multimap !^{k+2}\mathrm{B}$.*

ii. Let g be a function on binary words in k-`FEXP`*, for $k \geq 0$; then there is a term $\mathcal{M}$ representing g such that $\vdash \mathcal{M} : !\mathrm{W} \multimap !^{k+2}\mathrm{W_S}$.*

Proof.

i. Let $\mathfrak{M}$ be a deterministic Turing machine of time $2_k^{q(n)}$ computing a binary function f on binary words.
By Lemma 17 there is a derivation $\vdash \mathcal{Q} : !\mathrm{N} \multimap !^{k+1}\mathrm{N}$ representing $2_k^{q(n)}$. Moreover, the following derivations can be easily built:

$$\begin{array}{lcl} \Pi & \triangleright & \emptyset \mid w : !\mathrm{W} \mid \emptyset \vdash \texttt{init}_{k+2}(\texttt{conv}_{k+1}(\texttt{coer}^k\ !w)) : !^{k+2}\mathrm{C} \\ \Sigma & \triangleright & \emptyset \mid w : !\mathrm{W} \mid \emptyset \vdash !^{k+2}\texttt{step} : !^{k+2}(\mathrm{C} \multimap \mathrm{C}) \\ \Phi & \triangleright & \emptyset \mid w : !\mathrm{W} \mid \emptyset \vdash \mathcal{Q}(\texttt{length}_1\ !w) : !^{k+1}\mathrm{N} \end{array}$$

Observe that Π types the initial configuration at depth $k+2$, which is obtained by applying the initializing function to the input word w, while Σ types the step

function at depth $k+2$ and Φ types the tally integer representation of $2_k^{q(n)}$ at depth $k+1$.
Let $\mathcal{P} = \mathtt{iter}_{k+1}(\mathtt{init}_{k+2}(\mathtt{conv}_{k+1}(\mathtt{coer}^k\ !w)))(!^{k+2}\mathtt{step})(\mathcal{Q}(\mathtt{length}_1\ !w))$ be the term obtained by applying $\mathtt{iter}_{k+1}$ to such typed terms, thus iterating the step function $2_k^{q(n)}$ times starting from the initial configuration, which is typed by applying $\emptyset \mid w :!\mathrm{W} \mid \emptyset \vdash \mathtt{iter}_{k+1} :!^{k+2}\mathrm{C} \multimap !^{k+2}(\mathrm{C} \multimap \mathrm{C}) \multimap !^{k+1}\mathrm{N} \multimap !^{k+2}\mathrm{C}$ to derivations Π, Σ and Φ in turns; let $\Psi \triangleright \emptyset \mid w :!\mathrm{W} \mid \emptyset \vdash \mathcal{P} :!^{k+2}\mathrm{C}$ be the derivation obtained by this construction.
Then we can apply rule $(\multimap E)$ to $\emptyset \mid w :!\mathrm{W} \mid \emptyset \vdash \mathtt{accept}_{k+2} :!^{k+2}\mathrm{C} \multimap !^{k+2}\mathrm{B}$ and to Ψ, followed by one application of rule $(\multimap I^I)$ to abstract over w, in order to obtain the desired derivation.

ii. The proof retraces the same steps of the previous point, up to the construction of Ψ; then the desired derivation is obtained by applying rule $(\multimap E)$ to $\emptyset \mid w :!\mathrm{W} \mid \emptyset \vdash \mathtt{extract}_{k+2} :!^{k+2}\mathrm{C} \multimap !^{k+2}\mathrm{W_S}$ and to Ψ, followed by one application of rule $(\multimap I^I)$ to abstract over w.

□

Observe that, as usual, such completeness results hold in the sense of *function* representation, whereas few polynomial time *algorithms* can actually be implemented by $\lambda^!$-term of the desired type. We do not claim our typed language to be algorithmically expressive in this sense; instead, the interest of the present approach lies in the simple setting it provides in order to obtain a characterization of a family of complexity classes in a generic way.

By looking at the type of programs representing functions of `FPTIME`, one obvious drawback of such characterization is the fact that it does not account for compositionality; indeed polytime functions are closed by composition, while the mismatch of input and output type of our programs, paired with the nonexistence of a coercion from Scott to Church binary words, makes it so that they cannot be composed: for such reason, in the next section we offer an alternative characterization which is capable of solving the issue.

2.3 Composite types for an alternative characterization

In order to be able to compose programs representing `FPTIME` functions, we want to define a datatype which is better suited to serve as both input and output type of a function. To this aim, we wish to represent a word $w' \in \{0,1\}^\star$ through a pair

$\langle n, w\rangle$, where $n \in \mathbb{N}, w \in \{0,1\}^\star$, such that the following invariant holds:

$$w' = \begin{cases} w, & \text{if } \mathbf{length}(w) \leq n \\ \text{prefix of } w \text{ of length } n, & \text{otherwise.} \end{cases}$$

The $\otimes$ operator on types, defined in Section 2.2.2, comes in handy for the representation of a pair of datatypes. Nevertheless, the depth of each component has to be chosen carefully. Consider again the case of $k = 0$, namely, the characterization of `FPTIME`: if we take the pair $\mathrm{N} \otimes \mathrm{W}_S$ as both input and output type of a program, then we do not achieve much expressivity since no duplication is possible at depth 0; similarly, by considering a pair of the shape $!\mathrm{N} \otimes !\mathrm{W}_S$, then the iteration of the second component based on the first integer component is forbidden, again affecting completeness.

However, the latter property gives a hint on the right shape of the pair to be chosen: the `iter` term is typed in such a way that, if the natural number it expects is at depth i, then its other arguments and its result are at depth $i+1$. We thus introduce a new combined datatype defined as $!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S$, for $k \geq 0$, containing a pair of a Church integer $\underline{n}$ and a Scott binary word $\widehat{w}$, where $\underline{n}$ is meant to represent the length of a list, whose content is described by $\widehat{w}$; so, in particular, we choose the datatype $!\mathrm{N} \otimes !^2\mathrm{W}_S$ for $k = 0$.

Notation 1 (Match)**.** *We use the following notation:*

$$\begin{array}{lll} \texttt{match } w \texttt{ with} & \texttt{cons}_0 u & \Rightarrow \mathcal{M}_0[u, \overline{x}] \\ & \texttt{cons}_1 u & \Rightarrow \mathcal{M}_1[u, \overline{x}] \\ & \texttt{nil} & \Rightarrow \mathcal{N}[\overline{x}] \end{array}$$

as syntactic sugar for $w\ \lambda u.\lambda\overline{x}.\mathcal{M}_0\ \lambda u.\lambda\overline{x}.\mathcal{M}_1\ \lambda\overline{x}.\mathcal{N}\ \overline{x}$*, where* $\overline{x}$ *stands for a sequence of variables* $x_1 ... x_n$*, while* $\lambda\overline{x}.\mathcal{M}$ *stands for the abstraction* $\lambda x_1.\ ...\lambda x_n.\mathcal{M}$*, for* $n \geq 1$*.*

Note that such notation is also equivalent to `case` $\lambda u.\lambda\overline{x}.\mathcal{M}_0\ \lambda u.\lambda\overline{x}.\mathcal{M}_1\ \lambda\overline{x}.\mathcal{N}\ w\ \overline{x}$.

When computing on elements $!^{k+1}\underline{n} \otimes !^{k+2}\widehat{w}$ of type $!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S$, we want to maintain the invariant that $\mathbf{length}(w) \leq n$, which is enforced by the following lemma:

Lemma 19 (Invariant of composite datatype)**.** *For any* $k \geq 0$*, there exists a term* $\vdash \mathcal{M} : !\mathrm{N} \multimap !^2\mathrm{W}_S \multimap !^{k+2}\mathrm{W}_S$ *for* $k \geq 0$ *such that, for any* $\underline{n}$ *and* $\widehat{w}$*,*

$$\mathcal{M}\, !\underline{n}\, !^2\widehat{w} \xrightarrow{*} !^{k+2}\widehat{w'}$$

where $w' = w$ *if* $\mathbf{length}(w) \leq n$*, otherwise* w' *is the prefix of* w *of length* n*.*

Proof. Let $\mathtt{A}^i = (!^i\mathrm{W} \multimap !^i\mathrm{W}) \otimes \mathrm{W} = \forall \mathtt{a}.((!^i\mathrm{W} \multimap !^i\mathrm{W}) \multimap \mathrm{W} \multimap \mathtt{a}) \multimap \mathtt{a}$ and consider the following terms:

$$
\begin{array}{lcll}
\vdash \mathrm{step}_0 & = & \lambda(f \otimes u). & \texttt{match } u \texttt{ with} \\
& & & \quad \texttt{cons}_0 v \Rightarrow \lambda y.(f\ (\texttt{cons}^0 y)) \otimes v \\
& & & \quad \texttt{cons}_1 v \Rightarrow \lambda y.(f\ (\texttt{cons}^1 y)) \otimes v \\
& & & \quad \texttt{nil} \Rightarrow f \otimes \texttt{nil} \\
& : & \mathtt{A}^0 \multimap \mathtt{A}^0 & \\
\vdash \mathrm{step}_{i+1} & = & \lambda(f \otimes u). & \texttt{match } u \texttt{ with} \\
& & & \quad \texttt{cons}_0 v \Rightarrow \lambda^! y.(f\ !(\texttt{cons}_i^0 y)) \otimes v \\
& & & \quad \texttt{cons}_1 v \Rightarrow \lambda^! y.(f\ !(\texttt{cons}_i^1 y)) \otimes v \\
& & & \quad \texttt{nil} \Rightarrow f \otimes \texttt{nil} \\
& : & \mathtt{A}^{i+1} \multimap \mathtt{A}^{i+1} &
\end{array}
$$

where $\vdash \texttt{cons}_i^0 : !^i \mathrm{W}_S \multimap !^i \mathrm{W}_S$ and $\vdash \texttt{cons}_i^1 : !^i \mathrm{W}_S \multimap !^i \mathrm{W}_S$ are obtained as usual by applying Property 1 to $\vdash \texttt{cons}^0 : \mathrm{W}_S \multimap \mathrm{W}_S$ and $\vdash \texttt{cons}^1 : \mathrm{W}_S \multimap \mathrm{W}_S$ respectively.

Then the desired program $\mathcal{M}$ is built as follows:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Sigma \quad \vdots \quad \emptyset \mid \emptyset \mid \Theta \vdash n!\mathrm{step}_k : !(\mathtt{A}^k \multimap \mathtt{A}^k)}{\emptyset \mid \emptyset \mid \Theta \vdash (\lambda^! y.\lambda^! z.!((\pi_1(y(I \otimes z)))!^k\widehat{\epsilon}))(n!\mathrm{step}_k) : !\mathrm{W}_S \multimap !^{k+1}\mathrm{W}_S}\ (\multimap E) \quad \dfrac{}{\emptyset \mid \emptyset \mid \Theta \vdash w : !\mathrm{W}_S}\ (Ax^P)}{\emptyset \mid \emptyset \mid \Theta \vdash (\lambda^! y.\lambda^! z.!((\pi_1(y(I \otimes z)))!^k\widehat{\epsilon}))(n!\mathrm{step}_k)w : !^{k+1}\mathrm{W}_S}\ (\multimap E)}{\emptyset \mid n : !\mathrm{N}, w : !^2\mathrm{W}_S \mid \emptyset \vdash !((\lambda^! y.\lambda^! z.!((\pi_1(y(I \otimes z)))!^k\widehat{\epsilon}))(n!\mathrm{step}_k)w) : !^{k+2}\mathrm{W}_S}\ (!)}{\vdash \lambda^! n.\lambda^! w.!((\lambda^! y.\lambda^! z.!((\pi_1(y(I \otimes z)))!^k\widehat{\epsilon}))(n!\mathrm{step}_k)w) : !\mathrm{N} \multimap !^2\mathrm{W}_S \multimap !^{k+2}\mathrm{W}_S}\ (\multimap I^I_{\otimes})
$$

where Π is:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\emptyset \mid \emptyset \mid \Theta' \vdash \pi_1 : \mathtt{A}^k \multimap (!^k\mathrm{W}_S \multimap !^k\mathrm{W}_S)} \quad \dfrac{\dfrac{}{\emptyset \mid \emptyset \mid \Theta' \vdash y : \mathtt{A}^k \multimap \mathtt{A}^k}\ (Ax^P) \quad \Sigma}{\emptyset \mid \emptyset \mid \Theta' \vdash y(I \otimes z) : \mathtt{A}^k}}{\emptyset \mid \emptyset \mid \Theta' \vdash \pi_1(y(I \otimes z)) : !^k\mathrm{W}_S \multimap !^k\mathrm{W}_S} \quad \dfrac{\vdots}{\emptyset \mid \emptyset \mid \Theta' \vdash !^k\widehat{\epsilon} : !^k\mathrm{W}_S}}{\emptyset \mid \emptyset \mid \Theta' \vdash (\pi_1(y(I \otimes z)))!^k\widehat{\epsilon} : !^k\mathrm{W}_S}\ (\multimap E)}{\emptyset \mid !\Theta' \mid \Theta \vdash !((\pi_1(y(I \otimes z)))!^k\widehat{\epsilon}) : !^{k+1}\mathrm{W}_S}\ (!)}{\emptyset \mid \emptyset \mid \Theta \vdash \lambda^! y.\lambda^! z.!((\pi_1(y(I \otimes z)))!^k\widehat{\epsilon}) : !(\mathtt{A}^k \multimap \mathtt{A}^k) \multimap !\mathrm{W}_S \multimap !^{k+1}\mathrm{W}_S}\ (\multimap I^L)
$$

and Σ is

$$
\dfrac{\dfrac{\dfrac{\vdots}{\emptyset \mid \emptyset \mid \Theta' \vdash I : \forall \mathtt{a}.\mathtt{a} \multimap \mathtt{a}}\ (\forall E)}{\emptyset \mid \emptyset \mid \Theta' \vdash I : !^k\mathrm{W}_S \multimap !^k\mathrm{W}_S} \quad \dfrac{}{\emptyset \mid \emptyset \mid \Theta' \vdash z : \mathrm{W}_S}\ (Ax^P)}{\emptyset \mid \emptyset \mid \Theta' \vdash I \otimes z : \mathtt{A}^k}\ (\otimes I)
$$

with $\Theta = n : \mathrm{N}, w : !\mathrm{W}_S$ and $\Theta' = y : \mathtt{A}^k \multimap \mathtt{A}^k, z : \mathrm{W}_S$.

It is easy to check that $\mathcal{M}!\underline{n}!^2\mathrm{W}_S$ and $!^2((\pi_1(\mathrm{step}_k^n(I \otimes \widehat{w})))!^k\widehat{\epsilon})$ reduce to the same normal form, which is the word $!^{k+2}\widehat{w'}$ such that w' is a prefix of w and $\mathbf{length}(w') \leq n$.

□

Retracing the reading properties of datatypes B and W_S, we need to examine under which conditions we can read out a value of the composite datatype. In order to do so, we need a few intermediary results about the shape of terms which are typed either with a ! type or with a type variable:

Lemma 20 (Term of ! type)**.** *Let* $\Pi \triangleright \Gamma \mid \emptyset \mid \Theta \vdash \mathcal{M} : !\tau$ *and* $\mathcal{M} \in \mathbf{nf}_0$*, such that* $\Gamma, \Theta \subseteq x : \sigma_1 \multimap ... \multimap \sigma_n \multimap \mathtt{a}, y_1 : \mathtt{a}_1, ..., y_k : \mathtt{a}_k$ *($n \geq 1, k \geq 0$): then* $\mathcal{M} = !\mathcal{N}$ *for some* $\mathcal{N}$*.*

Proof. By induction on Π.
By inspecting the rules of the system, the last application of Π is of either rule (!) or rule ($\multimap E$).

Let Π end with an application of rule (!): then the result follows trivially.

Otherwise, let Π be

$$\frac{\Pi_1 \triangleright \Gamma_1 \mid \emptyset \mid \Theta \vdash \mathcal{P} : \rho \multimap !\tau \quad \Pi_2 \triangleright \Gamma_2 \mid \emptyset \mid \Theta \vdash \mathcal{Q} : \rho}{\Gamma_1, \Gamma_2 \mid \emptyset \mid \Theta \vdash \mathcal{P}\mathcal{Q} : !\tau} \ (\multimap E)$$

where $\mathcal{M} = \mathcal{P}\mathcal{Q}$ and $\Gamma = \Gamma_1, \Gamma_2$. Let us check all three possible cases:

- let $\mathcal{M} = (\lambda z.\mathcal{R})^{\mathtt{A} \multimap !\tau} \mathcal{Q}^{\mathtt{A}}$, so $\mathcal{M}$ is be a β-redex, which contradicts the hypothesis that $\mathcal{M} \in \mathbf{nf}_0$: therefore this case is not possible;

- let $\mathcal{M} = (\lambda^! z.\mathcal{R})^{!\sigma \multimap !\tau} \mathcal{Q}^{!\sigma}$, so by inductive hypothesis $\mathcal{Q} = !\mathcal{Q}'$ for some $\mathcal{Q}'$, and $\mathcal{M}$ is a !-redex, which contradicts the hypothesis that $\mathcal{M} \in \mathbf{nf}_0$: therefore this case is not possible;

- let $\mathcal{M} = z^{\rho_1 \multimap ... \multimap \rho_m \multimap !\tau} \mathcal{P}_1^{\rho_1} ... \mathcal{P}_m^{\rho_m}$, where $z = x$ because x is the only variable to be assigned an arrow type in Γ, therefore $\Pi_1 \triangleright \Gamma_1 \mid \emptyset \mid \Theta \vdash x\mathcal{P}_1 ... \mathcal{P}_{m-1} : \sigma \multimap !\tau$ and $\Pi_2 \triangleright \Gamma_2 \mid \emptyset \mid \Theta \vdash \mathcal{P}_m : \sigma$: since every application of x to $m \leq n$ subterms produces a term having either an arrow type or a type variable, this case is not possible.

Therefore, Π cannot end with an application of rule ($\multimap E$).

□

Lemma 21 (Term of type variable)**.** *Let* $\Pi \triangleright \Gamma \mid \emptyset \mid \Theta \vdash \mathcal{M} : \mathtt{a}$*, such that* $\mathcal{M} \in \mathbf{nf}_0$ *and* $\Gamma, \Theta = x : \sigma_1 \multimap ... \multimap \sigma_n \multimap \mathtt{a}, y_1 : \mathtt{a}_1, ..., y_k : \mathtt{a}_k$ *($n \geq 1, k \geq 0$): then either* $\mathcal{M} = y_i$*, if* $\mathtt{a} = \mathtt{a}_i$*, or* $\mathcal{M} = x\mathcal{P}_1 ... \mathcal{P}_n$ *for some* $\mathcal{P}_1, ..., \mathcal{P}_n$*.*

Proof. By induction on Π.

Since $\mathtt{a}$ is not an arrow type, Π cannot end with rule ($\multimap I^L$) nor ($\multimap I^I$). Similarly, the last application cannot be of a non-constructive rule, since the assumption of Π being clean. Moreover, since the type is linear, Π cannot end with an application of rule (!).

Let Π end with an application of rule (Ax^L) (resp. (Ax^P)): then there is $y_i \in \mathrm{dom}(\Gamma)$ (resp. $y_i \in \mathrm{dom}(\Theta)$) such that $x = y_i$ and $\mathtt{a} = \mathtt{a}_i$, therefore the first point has been proved.

Otherwise, let Π be

$$\frac{\Pi_1 \triangleright \Gamma_1 \mid \emptyset \mid \Theta \vdash \mathcal{P} : \rho \multimap \mathtt{a} \quad \Pi_2 \triangleright \Gamma_2 \mid \emptyset \mid \Theta \vdash \mathcal{Q} : \rho}{\Gamma_1, \Gamma_2 \mid \emptyset \mid \Theta \vdash \mathcal{P}\mathcal{Q} : \mathtt{a}} \ (\multimap E)$$

where $\mathcal{M} = \mathcal{P}\mathcal{Q}$ and $\Gamma = \Gamma_1, \Gamma_2$. Since $\mathcal{M} \in \mathbf{nf}_0$, there are only two possibilities:

- let $\mathcal{M} = (\lambda^! y.\mathcal{P})^{!\tau \multimap \mathtt{a}} \mathcal{Q}^{!\tau}$ where $\mathcal{Q} \neq !\mathcal{Q}'$, namely $\mathcal{M}$ is a block; by applying Lemma 20 to $\Pi_2 \triangleright \Gamma_2 \mid \emptyset \mid \Theta \vdash \mathcal{Q} : !\tau$, we know that $\mathcal{Q} = !\mathcal{Q}'$, which contradicts the hypothesis that $\mathcal{M} \in \mathbf{nf}_0$: therefore $\mathcal{M}$ is not a block;

- let $\mathcal{M} = z^{\rho_1 \multimap \ldots \multimap \rho_m \multimap \mathtt{a}} \mathcal{P}_1^{\rho_1} \ldots \mathcal{P}_m^{\rho_m}$; since x is the only variable to be assigned an arrow type whose rightmost type is $\mathtt{a}$, we know that $\Pi \triangleright \Gamma \mid \emptyset \mid \Theta \vdash x\mathcal{P}_1 \ldots \mathcal{P}_n : \mathtt{a}$ and $m = n$, where x is introduced either through rule (Ax^L), if $x \in \mathrm{dom}(\Gamma)$, or through rule (Ax^P), if $x \in \mathrm{dom}(\Theta)$: therefore the second point has been proved.

□

Now we are able to prove a reading result, similar to the ones of Property 8, for both pairs and Church integers:

Property 9 (Reading property of pairs)**.** *If* $\vdash \mathcal{M} : (\sigma_1 \otimes \sigma_2)$ *and* $\mathcal{M} \in \mathbf{nf}_0$*, then there are* $\vdash \mathcal{M}_1 : \sigma_1$ *and* $\vdash \mathcal{M}_2 : \sigma_2$ *such that* $\mathcal{M} = \mathcal{M}_1 \otimes \mathcal{M}_2$*.*

Proof. Consider a clean derivation Π proving $\vdash \mathcal{M} : \forall \mathtt{a}.((\sigma_1 \multimap \sigma_2 \multimap \mathtt{a}) \multimap \mathtt{a})$. Since Π is clean, the derivation ends with an application of rule $(\forall I)$, whose premise is $\Pi' \triangleright \vdash \mathcal{M} : (\sigma_1 \multimap \sigma_2 \multimap \mathtt{a}) \multimap \mathtt{a}$.
By Lemma 16 and by the fact that all bases are empty, $\mathcal{M}$ is not an application nor a variable; then $\mathcal{M}$ is an abstraction, in particular a linear one since the type is of the shape $\mathtt{A} \multimap \mathtt{a}$, so $\mathcal{M} = \lambda x.\mathcal{N}$ and Π' ends with an application of rule $(\multimap I^L)$, whose premise is $\Pi'' \vdash x : \sigma_1 \multimap \sigma_2 \multimap \mathtt{a} \mid \emptyset \mid \emptyset \vdash \mathcal{N} : \mathtt{a}$.
By applying Lemma 21 to Π'', $\mathcal{N} = x\mathcal{M}_1\mathcal{M}_2$ for some terms $\mathcal{M}_1, \mathcal{M}_2$, so Π'' is

$$\dfrac{\dfrac{\overline{\Gamma \mid \emptyset \mid \emptyset \vdash x : \sigma_1 \multimap \sigma_2 \multimap \mathtt{a}} \ (Ax^L) \quad \Sigma_1 \triangleright \vdash \mathcal{M}_1 : \sigma_1}{\Gamma \mid \emptyset \mid \emptyset \vdash x\mathcal{M}_1 : \sigma_2 \multimap \mathtt{a}} \ (\multimap E) \quad \Sigma_2 \triangleright \vdash \mathcal{M}_2 : \sigma_2}{\Gamma \mid \emptyset \mid \emptyset \vdash x\mathcal{M}_1\mathcal{M}_2 : \mathtt{a}} \ (\multimap E)$$

where $\Gamma = x : \sigma_1 \multimap \sigma_2 \multimap \mathtt{a}$.
Since $\mathcal{M} = \lambda x.x\mathcal{M}_1\mathcal{M}_2 = \mathcal{M}_1 \otimes \mathcal{M}_2$, the desired derivations for $\mathcal{M}_1$ and $\mathcal{M}_2$ are respectively Σ_1 and Σ_2.

□

Property 10 (Reading property of N)**.** *If* $\vdash \mathcal{M} :!^k\mathrm{N}$ *for* $k \geq 0$ *and* $\mathcal{M} \in \mathbf{nf}_{k+1}$*, then there exists* $n \in \mathbb{N}$ *such that* $\mathcal{M} =!^k\underline{n}$*.*

Proof. Recall that $\underline{n} = \lambda^! f.!(\lambda x.f^n x)$ for any $n \in \mathbb{N}$. Let Π be a clean derivation for $\vdash \mathcal{M} :!^k\mathrm{N}$: we proceed by induction on k.

Let $k = 0$, so $\Pi\triangleright \vdash \mathcal{M} : \mathrm{N}$ and $\mathcal{M} \in \mathbf{nf}_1$.
Since Π is clean, the derivation ends with an application of rule $(\forall I)$, whose premise is $\Pi'\triangleright \vdash \mathcal{M} :!(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a})$. By Lemma 16 and since the term is closed, $\mathcal{M}$ is not an application nor a variable; then $\mathcal{M}$ is an abstraction, in particular $\mathcal{M} = \lambda^! f.\mathcal{N}$ since the type has the shape $!\sigma \multimap \tau$, so Π' ends with an application of rule $(\multimap I^I)$, whose premise is $\emptyset \mid f :!(\mathtt{a} \multimap \mathtt{a}) \mid \emptyset \vdash \mathcal{N} :!(\mathtt{a} \multimap \mathtt{a})$.
By Property 7.i, $\mathcal{N} =!\mathcal{P}$ for some $\mathcal{P}$: then $\mathcal{M} = \lambda^! f.!\mathcal{P}$ and Π'' ends with an application of rule (!), whose premise is $\Sigma \triangleright \emptyset \mid \emptyset \mid f : \mathtt{a} \multimap \mathtt{a} \vdash \mathcal{P} : \mathtt{a} \multimap \mathtt{a}$.
Again by Lemma 16, $\mathcal{P}$ is not an application. If $\mathcal{P} = f$, then $\mathcal{M} = \lambda^! f.!f$ and Σ is a parking axiom, so the proof is done. Otherwise $\mathcal{P}$ is an abstraction, in particular $\mathcal{P} = \lambda x.\mathcal{Q}$ since the type has the shape $\mathtt{A} \multimap \tau$: then Σ ends with an application of rule $(\multimap I^L)$, whose premise is $\Sigma' \triangleright x : \mathtt{a} \mid \emptyset \mid f : \mathtt{a} \multimap \mathtt{a} \vdash \mathcal{Q} : \mathtt{a}$. By Lemma 21 either $\mathcal{Q} = x$, so $\mathcal{M} = \underline{0}$, or $\mathcal{Q} = f\mathcal{Q}'$: in the second case, Σ' ends with an application of rule $(\multimap E)$, whose premises are the parking axiom $\emptyset \mid \emptyset \mid f : \mathtt{a} \multimap \mathtt{a} \vdash f : \mathtt{a} \multimap \mathtt{a}$ and $\Sigma'' \vdash x : \mathtt{a} \mid \emptyset \mid f : \mathtt{a} \multimap \mathtt{a} \vdash \mathcal{Q}' : \mathtt{a}$.
Finally, by repeating the same reasoning n times on the right premise of rule $(\multimap E)$, we obtain $\mathcal{M} = \lambda^! f.!(\lambda x.f^n x)$ for some $n \in \mathbb{N}$: therefore $\mathcal{M} = \underline{n}$.

Now consider the case $k + 1$. By repeatedly applying Property 7.i, $\mathcal{M} =!^{k+1}\mathcal{N}$ and $\Pi'\triangleright \vdash \mathcal{N} : \mathrm{N}$: then the proof follows by inductive hypothesis.

□

Such results can be combined in order to prove a reading result for pairs of datatypes:

Property 11 (Reading property of pairs of datatypes)**.** *If* $\vdash \mathcal{M} :!^k\mathrm{N}\otimes!^{k+1}\mathrm{W}_S$ *for* $k \geq 0$ *and* $\mathcal{M} \in \mathbf{nf}_{k+1}$*, then there exists* $m \in \mathbb{N}$ *and* $w \in \{0,1\}^\star$ *such that* $\mathcal{M} =!^k\underline{m}\otimes!^{k+1}\widehat{w}$*.*

Proof. By Property 9, $\vdash \mathcal{M} :!^k\mathrm{N}\otimes!^{k+1}\mathrm{W}_S$ and $\mathcal{M} \in \mathbf{nf}_0$ imply there are derivations $\vdash \mathcal{M}_1 :!^k\mathrm{N}$ and $\vdash \mathcal{M}_2 :!^{k+1}\mathrm{W}_S$ such that $\mathcal{M} = \mathcal{M}_1 \otimes \mathcal{M}_2$: then the result follows easily by Properties 10 and 8.ii.

□

It is possible to prove a result similar to the one of Property 3, where a stratified bound on the size of the reduction is given for programs taking combined data as input:

Property 12. *Given a program $\mathcal{P}$, for any $k \geq 2$, there exists a polynomial q such that, for any $m \in \mathbb{N}$, $w \in \{0,1\}^\star$, $\mathcal{P}(!\underline{m} \otimes !^2\widehat{w}) \overset{*}{\rightsquigarrow} \mathcal{M}_k^1 \in \mathbf{nf}_{k-1}$ in at most $2_{k-2}^{q(n)}$ steps, and $|\mathcal{M}_k^1| \leq 2_{k-2}^{q(n)}$, where $n = m + \mathbf{length}(w)$. In particular, in the case where $k = 2$ we have a polynomial bound $q(n)$.*

Proof. The statement can be proved in a way similar to Property 3. For $k = 2$, it is easy to check that the number of steps at depths 0 and 1 are bounded by a constant, since the size of the term $\mathcal{P}(!\underline{m} \otimes !^2\widehat{w})$ at depth lower than or equal to 1 does not depend on m nor w.

□

The complexity soundness result can be proved, in a similar way to that of Theorem 5, by combining the bound for the reduction in the untyped $\lambda^!$-calculus with the reading property of pairs:

Theorem 7 (Soundness for combined datatype)**.** *Let $\vdash \mathcal{P} : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S)$ where $\mathcal{P}$ is a program, then for any $\underline{m}$ and $\widehat{w}$ the reduction of $\mathcal{P}(!\underline{m} \otimes !^2\widehat{w})$ to its normal form can be computed in time $2_k^{p(n)}$, where p is a polynomial and $n = m + \mathbf{length}(w)$.*

Proof. Easy, by combining Property 12, Theorem 3 and Property 9.

□

Finally we examine the matter of expressivity with respect to the combined datatype:

Theorem 8 (Completeness for combined datatype)**.** *Let f be a function on binary words in k-*`FEXP`*, for $k \geq 0$; then there is a term $\mathcal{M}$ representing f such that $\vdash \mathcal{M} : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S)$.*

Proof. We show how to simulate a Turing machine for a function of `k-FEXP` through a term of type $!\mathrm{W} \multimap !^{k+2}\mathrm{W}_S$, in a way similar to the proof of Theorem 6.ii.

Consider a Turing machine $\mathfrak{M}$ of time $2_k^{q(n)}$ computing f, thus the size of the output is also bound by $2_k^{q(n)}$.
By Lemma 17 there is $\vdash \mathcal{Q} : !\mathrm{N} \multimap !^{k+1}\mathrm{N}$ such that $\mathcal{Q}$ is a term representing $2_k^{q(n)}$. Let $!\underline{n} \otimes !^2\widehat{w}$ be the input of $\mathcal{M}$, and let w' be the binary word represented by $\langle n, w\rangle$; by Lemma 19 there is $\vdash \mathcal{P} : !\mathrm{N} \multimap !^2\mathrm{W}_S \multimap !^{k+2}\mathrm{W}_S$ such that $\mathcal{P}!\underline{n}!^2\widehat{w} \overset{*}{\to} !^{k+2}\widehat{w'}$.
Let $\Delta = m : !\mathrm{N}, u : !^2\mathrm{W}_S$; it is easy to build the following derivations:

$$\begin{array}{lcl} \Pi & \triangleright & \emptyset \mid \Delta \mid \emptyset \vdash \mathtt{init}_{k+2}(\mathcal{P}!m!u) : !^{k+2}\mathrm{C}_S \\ \Sigma & \triangleright & \emptyset \mid \Delta \mid \emptyset \vdash !^{k+2}\mathtt{step} : !^{k+2}(\mathrm{C}_S \multimap \mathrm{C}_S) \\ \Phi & \triangleright & \emptyset \mid \Delta \mid \emptyset \vdash \mathcal{Q}\ !m : !^{k+1}\mathrm{N} \end{array}$$

Let $\mathcal{N} = \mathtt{iter}_2(\mathtt{init}_{k+2}(\mathcal{P}!m!u))(!^{k+2}\mathtt{step})(\mathcal{Q}\ !m)$ be obtained by applying $\mathtt{iter}_{k+2}$ to the typed terms above; by suitable applications of rule $(\multimap E)$ to

$\emptyset \mid \Delta \mid \emptyset \vdash \mathtt{iter}_{k+1} : !^{k+2}\mathrm{C}_S \multimap !^{k+2}(\mathrm{C}_S \multimap \mathrm{C}_S) \multimap !^{k+1}\mathrm{N} \multimap !^{k+2}\mathrm{C}_S$ and derivations Π, Σ, Φ, we can build the derivation $\Psi \triangleright \emptyset \mid \Delta \mid \emptyset \vdash \mathcal{N} : !^{k+2}\mathrm{C}_S$.
Then the desired term $\mathcal{M}$ is the subject of the following derivation:

$$\dfrac{\dfrac{\Psi \quad \dfrac{\vdots}{\emptyset \mid \Delta \mid \emptyset \vdash \mathtt{extract}_{k+2} : !^{k+2}\mathrm{C}_S \multimap !^{k+2}\mathrm{W}_S} \quad \Psi \triangleright \emptyset \mid \Delta \mid \emptyset \vdash \mathcal{N} : !^{k+2}\mathrm{C}_S}{\emptyset \mid \Delta \mid \emptyset \vdash \mathtt{extract}_{k+2}\mathcal{N} : !^{k+2}\mathrm{W}_S}\,(\multimap E)}{\emptyset \mid \Delta \mid \emptyset \vdash (\mathcal{Q}!m) \otimes (\mathtt{extract}_{k+2}\mathcal{N}) : !^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S}\,(\otimes I)}{\vdash \lambda^!(m \otimes u).(\mathcal{Q}!m) \otimes (\mathtt{extract}_{k+2}\mathcal{N}) : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S)}\,(\multimap I^I_\otimes)$$

where the derived rules $(\otimes I)$ and $(\multimap I^I_\otimes)$ are defined in Table 2.2.

Observe here that the size of the output word w'' is bounded by the running time $2_k^{q(n)}$, namely the clock of the Turing machine. Then the composite type $!^{k+1}n' \otimes !^{k+2}\widehat{w''}$ representing the output of the computation, where $n' = 2_k^{q(n)}$, respects the invariant stated at the beginning of Section 2.3 and thus is a correct representation of w''.

□

Note that, in particular, terms of type $(!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S)$ correspond to the class k-FEXP. Some aspects of this alternative characterization are worth mentioning.

- We can now compose two terms of type $(!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!\mathrm{N} \otimes !^2\mathrm{W}_S)$, mirroring the fact that FPTIME is closed by composition.

- If $f \in$ FPTIME and $g \in$ k-FEXP, then by Theorem 8 there are programs $\mathcal{M}$ and $\mathcal{N}$, representing f and g respectively, such that $\Pi \triangleright \vdash \mathcal{M} : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!\mathrm{N} \otimes !^2\mathrm{W}_S)$ and $\Sigma \triangleright \vdash \mathcal{N} : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S)$. Then we can compose these programs as follows:

$$\dfrac{\dfrac{\Pi \quad \dfrac{}{x : !\mathrm{N} \otimes !^2\mathrm{W}_S \mid \emptyset \mid \emptyset \vdash x : !\mathrm{N} \otimes !^2\mathrm{W}_S}\,(Ax^L)}{x : !\mathrm{N} \otimes !^2\mathrm{W}_S \mid \emptyset \mid \emptyset \vdash \mathcal{M}x : !\mathrm{N} \otimes !^2\mathrm{W}_S}\,(\multimap E) \quad \Sigma}{x : !\mathrm{N} \otimes !^2\mathrm{W}_S \mid \emptyset \mid \emptyset \vdash \mathcal{N}(\mathcal{M}x) : !^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S}\,(\multimap E)}{\vdash \lambda x.\mathcal{N}(\mathcal{M}x) : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S)}\,(\multimap I^L)$$

  which shows that $g \circ f \in$ k-FEXP.

- If $k \geq 1$, then the class k-FEXP is not closed by composition; nonetheless, the typing allows to show that $f \in k$-FEXP and $g \in k'$-FEXP imply $g \circ f \in (k + k')$-FEXP, for $k \geq 1$ and $k' \geq 1$. By Theorem 8 there are terms $\mathcal{M}$ and $\mathcal{N}$, representing functions f and g respectively, such that $\Pi \triangleright \vdash \mathcal{M} : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S)$ and $\Sigma \triangleright \vdash \mathcal{N} : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k'+1}\mathrm{N} \otimes !^{k'+2}\mathrm{W}_S)$.

Let $\Sigma' \triangleright \emptyset \mid n :!\mathrm{N}, w :!^2\mathrm{W}_S \mid \emptyset \vdash \mathcal{N} : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k'+1}\mathrm{N} \otimes !^{k'+2}\mathrm{W}_S)$ be obtained from Σ by Lemma 12; then we can build the following derivation Ψ

$$\dfrac{\Sigma' \quad \dfrac{\dfrac{\dfrac{}{\emptyset \mid \emptyset \mid n : \mathrm{N}, w :!\mathrm{W}_S \vdash n : \mathrm{N}}\,(Ax^P)}{\emptyset \mid n :!\mathrm{N}, w :!^2\mathrm{W}_S \mid \emptyset \vdash !n :!\mathrm{N}}\,(!) \quad \dfrac{\dfrac{}{\emptyset \mid \emptyset \mid n : \mathrm{N}, w :!\mathrm{W}_S \vdash w :!\mathrm{W}_S}\,(Ax^P)}{\emptyset \mid n :!\mathrm{N}, w :!^2\mathrm{W}_S \mid \emptyset \vdash !w :!^2\mathrm{W}_S}\,(!)}{\emptyset \mid n :!\mathrm{N}, w :!^2\mathrm{W}_S \mid \emptyset \vdash !n \otimes !w :!\mathrm{N} \otimes !^2\mathrm{W}_S}\,(\otimes I)}{\emptyset \mid n :!\mathrm{N}, w :!^2\mathrm{W}_S \mid \emptyset \vdash \mathcal{N}(!n \otimes !w) :!^{k'+1}\mathrm{N} \otimes !^{k'+2}\mathrm{W}_S}\,(\multimap E)$$

from which we obtain $\Psi_1 \triangleright \vdash \lambda^! n.\lambda^! w.\pi_1(\mathcal{N}(!n \otimes !w)) :!\mathrm{N} \multimap !^2\mathrm{W}_S \multimap !^{k'+1}\mathrm{N}$ and $\Psi_2 \triangleright \vdash \lambda^! n.\lambda^! w.\pi_2(\mathcal{N}(!n \otimes !w)) :!\mathrm{N} \multimap !^2\mathrm{W}_S \multimap !^{k'+2}\mathrm{W}_S$, by applying the respective projector and then abstracting over n and w.

By applying Property 1 to Ψ_1 and Ψ_2, there are $\vdash \mathcal{P}_1 :!^{k+1}\mathrm{N} \multimap !^{k+2}\mathrm{W}_S \multimap !^{k+k'+1}\mathrm{N}$ and $\vdash \mathcal{P}_2 :!^{k+1}\mathrm{N} \multimap !^{k+2}\mathrm{W}_S \multimap !^{k+k'+2}\mathrm{W}_S$, such that $\mathcal{P}_i$ represent the same function as $\lambda^! n.\lambda^! w.\pi_i(\mathcal{N}(!n \otimes !w))$ for $i \in \{1, 2\}$.

Let $\Delta = n :!^{k+1}\mathrm{N}, w :!^{k+2}\mathrm{W}_S$; then we can build the following derivation Φ:

$$\dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots \\ \emptyset \mid \Delta \mid \emptyset \vdash \mathcal{P}_1 !n !w :!^{k+k'+1}\mathrm{N} & \emptyset \mid \Delta \mid \emptyset \vdash \mathcal{P}_2 !n !w :!^{k+k'+2}\mathrm{W}_S\end{array}}{\emptyset \mid \Delta \mid \emptyset \vdash (\mathcal{P}_1 !n !w) \otimes (\mathcal{P}_2 !n !w) :!^{k+k'+1}\mathrm{N} \otimes !^{k+k'+2}\mathrm{W}_S}\,(\otimes I)}{\vdash \lambda^!(n \otimes w).(\mathcal{P}_1 !n !w) \otimes (\mathcal{P}_2 !n !w) : (!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S) \multimap (!^{k+k'+1}\mathrm{N} \otimes !^{k+k'+2}\mathrm{W}_S)}\,(\multimap I^I_\otimes)$$

where $\mathcal{P} = \lambda^!(n \otimes w).(\mathcal{P}_1 !n !w) \otimes (\mathcal{P}_2 !n !w)$ is a term representing the same function as $\mathcal{N}$, namely $g \in k'$-FEXP.

Finally we can compose the program $\mathcal{M}$ with the term obtained above:

$$\dfrac{\dfrac{\dfrac{\Phi \quad \dfrac{\Pi \quad \dfrac{}{x :!\mathrm{N} \otimes !^2\mathrm{W}_S \mid \emptyset \mid \emptyset \vdash x :!\mathrm{N} \otimes !^2\mathrm{W}_S}\,(Ax^L)}{x :!\mathrm{N} \otimes !^2\mathrm{W}_S \mid \emptyset \mid \emptyset \vdash \mathcal{M}x :!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S}\,(\multimap E)}{x :!\mathrm{N} \otimes !^2\mathrm{W}_S \mid \emptyset \mid \emptyset \vdash \mathcal{P}(\mathcal{M}x) :!^{k+k'+1}\mathrm{N} \otimes !^{k+k'+2}\mathrm{W}_S}\,(\multimap E)}{\vdash \lambda x.\mathcal{P}(\mathcal{M}x) : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k+k'+1}\mathrm{N} \otimes !^{k+k'+2}\mathrm{W}_S)}\,(\multimap I^L)$$

which shows that $g \circ f \in (k + k')$-FEXP.

While the previous characterization of k-FEXP, given in Section 2.2.1, offers the advantage of simplicity by employing classical datatypes (Church and Scott binary words), this alternative composite characterization offers a better account of the closure properties of the considered complexity classes, at the price of a slightly more involved representation of words.
Moreover, as shown in the next section, the flexibility of the latter choice allows also to tackle other less explored characterizations.

2.4 Other characterizations through composite datatypes

We give two examples of characterizations, beside that of the k-FEXP hierarchy, which can be achieved by carefully tuning the composite datatype introduced at the

beginning of Section 2.3.

2.4.1 Characterization of polyFEXP

We denote by *poly*FEXP the class of functions computable in exponential time on a deterministic Turing machine, whose output size is polynomially bounded with respect to the input. In order to obtain this effect, we choose the type $!\mathrm{N}\otimes!^3\mathrm{W}_S$ as the output type of the program: the intuition behind such choice is that, when looking at the components of the pair, we can actually see only a part of the binary word, since the tally integer represented by the first component is smaller than the second component. This general reasoning can be exploited in order to characterize classes of functions whose output's size is bounded by a function of the input:

Theorem 9 (Characterization of polyFEXP)**.**

i. Let $\vdash \mathcal{P} : !\mathrm{N}\otimes!^2\mathrm{W}_S \multimap !\mathrm{N}\otimes!^3\mathrm{W}_S$*; then there exists a polynomial* q *such that, for any* $n \in \mathbb{N}$ *and* $w \in \{0,1\}^\star$ *such that* $\mathbf{length}(w) \leq n$*,* $\mathcal{P}(!\underline{n}\otimes!^2\widehat{w}) \overset{*}{\rightsquigarrow} \mathcal{Q} \in \mathbf{nf}_3$ *can be computed in time* $O(2^{q(n)})$ *on a Turing machine and* $\mathcal{Q} = !\underline{n'}\otimes!^3\widehat{w'}$ *represents the word* w''*, where* $\mathbf{length}(w'') \leq q(n)$*.*

ii. Let f *be a function on binary words in* $\mathcal{P}$*-bound-*EXP*: then there is a term* $\mathcal{M}$ *representing* f *such that* $\vdash \mathcal{M} : (!\mathrm{N}\otimes!^2\mathrm{W}_S) \multimap (!\mathrm{N}\otimes!^3\mathrm{W}_S)$*.*

Proof.

i. Let $\mathcal{P} = \lambda^!(x_1 \otimes x_2).\mathcal{N} = \lambda x.x(\lambda^! x_1.\lambda^! x_2.\mathcal{N})$, so $x \notin \mathrm{FV}(\mathcal{N})$ and both x_1 and x_2 occur at depth 1 in $\mathcal{N}$: we examine the reduction of $\vdash \mathcal{P}(\lambda y.y\ !\underline{n}\ !^2\widehat{w}) : !\mathrm{N}\otimes!^3\mathrm{W}_S$ using the level-by-level reduction strategy.

Let us consider rounds 0 through 3, using the notation of Fig. 2.1

R-0. At round 0, the reduction sequence is the following:

$$\begin{array}{lll} (\lambda x.x(\lambda^! x_1.\lambda^! x_2.\mathcal{N}))(\lambda y.y\ !\underline{n}\ !^2\widehat{w}) & \rightsquigarrow_0 & (\lambda y.y\ !\underline{n}\ !^2\widehat{w})(\lambda^! x_1.\lambda^! x_2.\mathcal{N}) = \mathcal{M}_0^1 \\ & \rightsquigarrow_0 & (\lambda^! x_1.\lambda^! x_2.\mathcal{N})!\underline{n}\ !^2\widehat{w} = \mathcal{M}_0^2 \\ & \rightsquigarrow_0 & (\lambda^! x_2.\mathcal{N}[\underline{n}/x_1])!^2\widehat{w} = \mathcal{M}_0^3 \\ & \rightsquigarrow_0 & \mathcal{N}[\underline{n}/x_1, !\widehat{w}/x_2] = \mathcal{M}_0^4 \end{array}$$

where $\mathcal{N} = \mathcal{C}[x_1]_1...[x_1]_1[x_2]_1...[x_2]_1$ for some context $\mathcal{C}$. Let $c_i = n_0(x_i, \mathcal{M}_0^0)$, which depends only on the program $\mathcal{P}$ ($i \in \{1,2\}$).
By definition we have:

$$\begin{array}{rcl} \mathcal{M}_0^4 & = & \mathcal{C}[\underline{n}/x_1]_1...[\underline{n}/x_1]_1[!\widehat{w}/x_2]...[!\widehat{w}/x_2]_1 \\ & = & \mathcal{C}[\lambda^! f.!(\lambda x.f^n x)/x_1]_1...[\lambda^! f.!(\lambda x.f^n x)/x_1]_1[!\widehat{w}/x_2]...[!\widehat{w}/x_2]_1 \end{array}$$

so $\mathcal{M}_0^4 = \mathcal{M}_1^0 \in \mathbf{nf}_0$, since $\mathcal{N} \in \mathbf{nf}_0$ implies $\mathcal{N}[\underline{n}/x_1, !\widehat{w}/x_2] \in \mathbf{nf}_0$. By Lemma 2

$$\begin{array}{lcll} |\mathcal{M}_1^0|_0 & < & |\mathcal{P}|_0 & \\ |\mathcal{M}_1^0|_1 & \leq & |\mathcal{P}|_1 + c_1 & \\ |\mathcal{M}_1^0|_2 & \leq & |\mathcal{P}|_2 + 2 \cdot c_1 \cdot n + c_2 \cdot |\widehat{w}| & \\ |\mathcal{M}_1^0|_i & \leq & |\mathcal{P}|_i & \text{for } i \geq 3 \end{array}$$

therefore $|\mathcal{M}_1^0|$ is linear in n.

R-1. At round 1, by Property 1.i $\mathcal{M}_1^0 \overset{*}{\rightsquigarrow}_1 \mathcal{M}_2^0 \in \mathbf{nf}_1$. By Lemma 2,such reduction takes d_1 steps, where $d_1 \leq |\mathcal{M}_1^0|_1 \leq |\mathcal{P}|_1 + c_1$ and $|\mathcal{M}_2^0|_1 \leq |\mathcal{M}_1^0|_1$. By Corollary 2 $|\mathcal{M}_2^0| \leq |\mathcal{M}_1^0| \cdot (|\mathcal{M}_1^0| + 1)^{|\mathcal{P}|_1 + c_1}$: therefore the number of reduction steps is a constant and the size after round 1 is polynomial in n, namely $|\mathcal{M}_2^0| \leq p(n)$ for some polynomial p.

R-2. At round 2, by Property 1.i $\mathcal{M}_2^0 \overset{*}{\rightsquigarrow}_2 \mathcal{M}_3^0 \in \mathbf{nf}_2$. By Lemma 2 the reduction is done in $d_2 \leq |\mathcal{M}_2^0|_2 \leq p(n)$ steps and $|\mathcal{M}_3^0|_{2+} \leq |\mathcal{M}_2^0|_{2+}$.

Observe that, by Property 9, $\vdash \mathcal{M}_3^0 :!\mathrm{N} \otimes !^3\mathrm{W}_S$ and $\mathcal{M}_3^0 \in \mathbf{nf}_0$ imply $\mathcal{M}_3^0 = \mathcal{R} \otimes \mathcal{Q}$, for some $\vdash \mathcal{R} :!\mathrm{N}$ and $\vdash \mathcal{Q} :!^3\mathrm{W}_S$; moreover, by Property 10, $\vdash \mathcal{R} :!\mathrm{N}$ and $\mathcal{R} \in \mathbf{nf}_2$ imply $\mathcal{R} = !\underline{n'}$, for some $n' \in \mathbb{N}$. Since $|\mathcal{R}| \leq |\mathcal{M}_3^0|_2 \leq |\mathcal{M}_2^0|_2 \leq p(n)$ by Lemma 2, n' is polynomial in n, and by Corollary 2 we obtain

$$|\mathcal{M}_3^0| \leq p(n) \cdot (p(n)+1)^{p(n)} \leq p(n) \cdot (2 \cdot p(n))^{p(n)} \leq p(n) \cdot 2^{3 \cdot p(n)} \leq 2^{q(n)}$$

for some polynomial $q(n)$.

R-3. Since $!\underline{n'} \in \mathbf{nf}_3$, at round 3 we consider the reduction $\mathcal{Q} = \mathcal{Q}_3^0 \overset{*}{\rightsquigarrow}_3 \mathcal{Q}_4^0 \in \mathbf{nf}_3$, which follows from Property 1.i. By Lemma 2, the reduction is performed in $d_3 \leq |\mathcal{Q}_3^0| \leq 2^{q(n)}$ steps and $|\mathcal{Q}_4^0|_{3+} \leq |\mathcal{Q}_3^0|_{3+} \leq 2^{q(n)}$.

Observe that, by Lemma 8.ii, $\vdash \mathcal{Q}_4^0 :!^3\mathrm{W}_S$ and $\mathcal{Q}_4^0 \in \mathbf{nf}_3$ imply $\mathcal{Q}_4^0 = !^3\widehat{w'}$, for some Scott binary word w'. Then $\mathcal{M}_4^0 = !\underline{n'} \otimes !^3\widehat{w'} \in \mathbf{nf}_3$ represents a binary word w'', such that $\mathbf{length}(w'') \leq n' \leq q(n)$, and the reduction $\mathcal{M} \overset{*}{\rightsquigarrow} \mathcal{M}_4^0$ is done in at most $2^{q(n)}$ steps. Finally, by employing the approximations introduced in Section 2.1.6, a proper time bound can be obtained as in Property 4.

ii. Consider a Turing machine $\mathfrak{M}$ of time $2^{q(n)}$ computing f, so the size of the output is also bound by $2^{q(n)}$.

By Lemma 17, let $\vdash \mathcal{R} :!\mathrm{N} \multimap !\mathrm{N}$ and $\vdash \mathcal{Q} :!\mathrm{N} \multimap !^2\mathrm{N}$ represent the polynomial $q(n)$ and the exponential $2^{q(n)}$ respectively; moreover, let $!\underline{n} \otimes !^2\widehat{w}$ be the input of $\mathcal{M}$, such that $|\widehat{w}| \leq n$. By Lemma 19 there is $\vdash \mathcal{P} : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap !^3\mathrm{W}_S$ such

that $\mathcal{P}(!\underline{n}\otimes!^2\widehat{w}) \xrightarrow{*}!^3\widehat{w}$. Let $\Delta = m :!\mathrm{N}, u :!^2\mathrm{W}_S$; it is easy to build the following derivations:

$$\begin{array}{lcl} \Pi & \triangleright & \emptyset \mid \Delta \mid \emptyset \vdash \mathtt{init}_3(\mathcal{P}(!m\otimes!u)) :!^3\mathrm{C}_S \\ \Sigma & \triangleright & \emptyset \mid \Delta \mid \emptyset \vdash !^3\mathtt{step} :!^3(\mathrm{C}_S \multimap \mathrm{C}_S) \\ \Phi & \triangleright & \emptyset \mid \Delta \mid \emptyset \vdash \mathcal{Q}\ !m :!^2\mathrm{N} \end{array}$$

By applying $\emptyset \mid \Delta \mid \emptyset \vdash \mathtt{iter}_2 :!^3\mathrm{C}_S \multimap !^3(\mathrm{C}_S \multimap \mathrm{C}_S) \multimap !^2\mathrm{N} \multimap !^3\mathrm{C}_S$ to derivations Π, Σ, Φ, we obtain $\Psi \triangleright \emptyset \mid \Delta \mid \emptyset \vdash \mathtt{iter}_2(\mathtt{init}_3(\mathcal{P}(!m\otimes!u)))(!^3\mathtt{step})(\mathcal{Q}\ !m) :!^3\mathrm{C}_S$. For short, let $\mathcal{N}$ denote the subject of Ψ, so that $\Psi \triangleright \emptyset \mid \Delta \mid \emptyset \vdash \mathcal{N} :!^3\mathrm{C}_S$.

Then the desired term $\mathcal{M}$ is the subject of the following derivation:

$$\dfrac{\dfrac{\dfrac{\emptyset \mid \Delta \mid \emptyset \vdots \mathcal{R}!m :!\mathrm{N} \qquad \dfrac{\emptyset \mid \Delta \mid \emptyset \vdots \mathtt{extract}_3 :!^3\mathrm{C}_S \multimap !^3\mathrm{W}_S \qquad \Psi \triangleright \emptyset \mid \Delta \mid \emptyset \vdash \mathcal{N} :!^3\mathrm{C}_S}{\emptyset \mid \Delta \mid \emptyset \vdash \mathtt{extract}_3\mathcal{N} :!^3\mathrm{W}_S}\,(\multimap E)}{\emptyset \mid \Delta \mid \emptyset \vdash (\mathcal{R}!m)\otimes(\mathtt{extract}_3\mathcal{N}) :!\mathrm{N}\otimes!^3\mathrm{W}_S}\,(\otimes I)}{\vdash \lambda^!(m\otimes u).(\mathcal{R}!m)\otimes(\mathtt{extract}_3\mathcal{N}) :!\mathrm{N}\otimes!^2\mathrm{W}_S \multimap !\mathrm{N}\otimes!^3\mathrm{W}_S}\,(\multimap I^I_\otimes)$$

where the derived rules $(\otimes I)$ and $(\multimap I^I_\otimes)$ are defined in Table 2.2.

□

2.4.2 Characterization of NP problems

The class NP is known as the class of languages that have polynomial time verifiers [Sip97], that is, those functions for which we are able to check in polynomial time whether the input is in the language when given a polynomial size *witness* for it. More formally, given a language $\mathcal{L}$, we say that $\mathcal{L}$ is in NP if and only if there are two polynomials p, q and a deterministic Turing machine $\mathfrak{M}$ such that:

- for all u, w' the machine $\mathfrak{M}$ runs in time $p(|u|)$ on input (u, w');
- for all $u \in \mathcal{L}$, there is w' of length $q(|u|)$ such that $\mathfrak{M}$ accepts on input (u, w');
- for all $u \notin \mathcal{L}$, for all w' of length $q(|u|)$, $\mathfrak{M}$ rejects on input (u, w').

We want to show that it is possible, using the deterministic language of $\lambda^!$-calculus, to characterize also the class of functions for which there exists a polynomial verifier, namely, the class NP. We say that a predicate is *w-representable* if it is representable through a witness:

Definition 15. *Let $g : \{0,1\}^* \to \{0,1\}$ be a predicate: we say that g is w-represented by $\vdash \mathcal{M} :!\mathrm{N}\otimes!^2\mathrm{W}_S\otimes!^2\mathrm{W}_S \multimap !^2\mathrm{B}$ if, for any word u represented by $!\underline{n}\otimes!^2\widehat{w}$,*

$$g(u) = \begin{cases} 1 & \text{if } \exists w' \in \{0,1\}^*.\mathcal{M}\ (!\underline{n}\otimes!^2\widehat{w}\otimes!^2\widehat{w'}) \overset{*}{\rightsquigarrow} !^2\mathtt{true}; \\ 0 & \text{if } \forall w' \in \{0,1\}^*.\mathcal{M}\ (!\underline{n}\otimes!^2\widehat{w}\otimes!^2\widehat{w'}) \overset{*}{\rightsquigarrow} !^2\mathtt{false}. \end{cases}$$

We show that the predicates w-representable with such type are all and only those of NP:

Theorem 10 (Characterization of NP)**.**

i. If g is w-represented by $\vdash \mathcal{P} : !\mathrm{N} \otimes !^2\mathrm{W}_S \otimes !^2\mathrm{W}_S \multimap !^2\mathrm{B}$, then $g \in \mathtt{NP}$.

ii. If $g \in \mathtt{NP}$, then there is a program $\mathcal{P}$ such that g is w-represented by $\vdash \mathcal{P} : !\mathrm{N} \otimes !^2\mathrm{W}_S \otimes !^2\mathrm{W}_S \multimap !^2\mathrm{B}$.

Proof.

i. Let $\mathcal{P} = \lambda^!(x_1 \otimes x_2 \otimes x_3).\mathcal{N} = \lambda x.x(\lambda^! x_1.\lambda^! x_2.\lambda^! x_3.\mathcal{N})$: we study the reduction of $\vdash (\lambda^!(x_1 \otimes x_2 \otimes x_3).\mathcal{N})(!\underline{n} \otimes !^2\widehat{w} \otimes !^2\widehat{w'}) : !^2\mathrm{B}$ using the level-by-level reduction strategy, where $!\underline{n} \otimes !^2\widehat{w}$ represents the word w'' such that $\mathbf{length}(w'') \leq n$, and $\mathbf{length}(w') \leq p(n)$ for some polynomial p.

In the following we use the notation introduced in Fig. 2.1. Let us consider rounds 0 through 2; the proof is essentially the same of Theorem 9.i, except that here we have to consider the additional argument of the composite input.

R-0. At round 0, the reduction sequence is the following:

$$\begin{array}{rll}
\mathcal{P}(\lambda y.y!\underline{n}\ !^2\widehat{w}\ !^2\widehat{w'}) & \rightsquigarrow_0 & (\lambda y.y!\underline{n}\ !^2\widehat{w}\ !^2\widehat{w'})(\lambda^! x_1.\lambda^! x_2.\lambda^! x_3.\mathcal{N}) \quad = \mathcal{M}_0^1 \\
& \rightsquigarrow_0 & (\lambda^! x_1.\lambda^! x_2.\lambda^! x_3.\mathcal{N})!\underline{n}\ !^2\widehat{w}\ !^2\widehat{w'} \quad = \mathcal{M}_0^2 \\
& \rightsquigarrow_0 & (\lambda^! x_2.\lambda^! x_3.\mathcal{N}[\underline{n}/x_1])!^2\widehat{w}\ !^2\widehat{w'} \quad = \mathcal{M}_0^3 \\
& \rightsquigarrow_0 & (\lambda^! x_3.\mathcal{N}[\underline{n}/x_1, !\widehat{w}/x_2])!^2\widehat{w'} \quad = \mathcal{M}_0^4 \\
& \rightsquigarrow_0 & \mathcal{N}[\underline{n}/x_1, !\widehat{w}/x_2, !\widehat{w'}/x_3] \quad = \mathcal{M}_0^5
\end{array}$$

Let $\mathcal{N} = \mathcal{C}[x_1]_1 ... [x_1]_1 [x_2]_1 ... [x_2]_1 [x_3]_1 ... [x_3]_1$ for some context $\mathcal{C}$; note that $c_i = n_0(x_i, \mathcal{M})$ depends only on the program $\mathcal{P}$ ($i \in \{1, 2, 3\}$). By definition:

$$\begin{array}{rcl}
\mathcal{M}_0^5 & = & \mathcal{C}[\underline{n}/x_1]_1 ... [\underline{n}/x_1]_1 [!\widehat{w}/x_2] ... [!\widehat{w}/x_2]_1 [!\widehat{w'}/x_3] ... [!\widehat{w'}/x_3]_1 \\
& = & \mathcal{C}[\lambda^! f.!(\lambda x.f^n x)/x_1]_1 ... [\lambda^! f.!(\lambda x.f^n x)/x_1]_1 [!\widehat{w}/x_2] ... [!\widehat{w}/x_2]_1 [!\widehat{w'}/x_3] ... [!\widehat{w'}/x_3]_1
\end{array}$$

where $\mathcal{M}_0^5 = \mathcal{M}_1^0 \in \mathbf{nf}_0$, since $\mathcal{N} \in \mathbf{nf}_0$ implies $\mathcal{N}[\underline{n}/x_1, !\widehat{w}/x_2, !\widehat{w'}/x_3] \in \mathbf{nf}_0$. By Lemma 2:

$$\begin{array}{rcll}
|\mathcal{M}_1^0|_0 & < & |\mathcal{P}|_0 & \\
|\mathcal{M}_1^0|_1 & \leq & |\mathcal{P}|_1 + c_1 & \\
|\mathcal{M}_1^0|_2 & \leq & |\mathcal{P}|_2 + 2 \cdot c_1 \cdot n + c_2 \cdot |\widehat{w}| + c_3 \cdot |\widehat{w'}| & \\
|\mathcal{M}_1^0|_i & \leq & |\mathcal{P}|_i & \text{for } i \geq 3
\end{array}$$

so $|\mathcal{M}_1^0|$ is polynomial in n.

R-1. At round 1, by Property 1.i we have $\mathcal{M}_1^0 \rightsquigarrow_1^* \mathcal{M}_2^0 \in \mathbf{nf}_1$. By Lemma 2 the reduction is done in d_1 steps, where $d_1 \leq |\mathcal{M}_1^0|_1 \leq |\mathcal{P}|_1 + c_1$, and $|\mathcal{M}_2^0|_{1+} \leq |\mathcal{M}_1^0|_{1+}$. By Corollary 2 $|\mathcal{M}_2^0| \leq |\mathcal{M}_1^0| \cdot (|\mathcal{M}_1^0| + 1)^{|\mathcal{P}|_1 + c_1}$: therefore the number of reduction steps is a constant and the size after round 1 is polynomial in n, that is, $|\mathcal{M}_2^0| \leq q(n)$ for some polynomial q.

R-2. At round 2, by Property 1.i we have $\mathcal{M}_2^0 \rightsquigarrow_2^* \mathcal{M}_3^0 \in \mathbf{nf}_2$. By Lemma 2 the reduction is done in $d_2 \leq |\mathcal{M}_2^0|_2 \leq q(n)$ steps and $|\mathcal{M}_3^0|_{2+} \leq |\mathcal{M}_2^0|_{2+}$.
Observe that, by Property 8.i, $\vdash \mathcal{M}_3^0 :!^2\mathrm{B}$ and $\mathcal{M}_3^0 \in \mathbf{nf}_2$ imply either $\mathcal{M}_3^0 =!^2\mathtt{true}$ or $\mathcal{M}_3^0 =!^2\mathtt{false}$; moreover, by Lemma 2 $|\mathcal{M}_3^0| \leq |\mathcal{M}_2^0| \leq q(n)$: therefore $|\mathcal{M}_3^0|$ is polynomial in n.

Since we are able to verify in polynomial time if w' is a witness for g, that is, whether $g(w') = 0$ or $g(w') = 1$, we have that $g \in \mathtt{NP}$.

ii. Consider a Turing machine $\mathfrak{M}$ of time $p(n)$, such that $\mathfrak{M}$ is a verifier for g; then there exists a polynomial q such that:

- if $g(u) = 0$ then for all w' of size $q(n)$ the machine rejects on input (u, w');
- if $g(u) = 1$ then there is a w' of size $q(n)$ such that the machine accepts on input (u, w').

Let $\mathtt{init}^+$ be the program which initializes the tape with the input words, such that $\vdash \mathtt{init}^+ :!^2\mathrm{W_S} \multimap !^2\mathrm{W_S} \multimap !^2\mathrm{C}$; this program is easily adapted from the definition of $\mathtt{init}$.

By Lemma 17, let $\vdash \mathcal{R} :!\mathrm{N} \multimap !\mathrm{N}$ represent the polynomial $p(n)$; moreover, let $!\underline{n} \otimes !^2\widehat{w} \otimes !^2\widehat{w'}$ be the input of $\mathcal{M}$, such that $|\widehat{w}| \leq n$ and $|\widehat{w'}| \leq q(n)$. By Lemma 19 there is $\vdash \mathcal{P} : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap !^2\mathrm{W}_S$ such that $\mathcal{P}(!\underline{n} \otimes !^2\widehat{w}) \xrightarrow{*} !^2\widehat{w}$. Let $\Delta = m :!\mathrm{N}, u :!^2\mathrm{W}_S, v :!^2\mathrm{W}$; then we can easily build the following derivations:

$$\begin{array}{lcl} \Pi & \triangleright & \emptyset \mid \Delta \mid \emptyset \vdash \mathtt{init}^+(\mathcal{P}(!m \otimes !u))(\mathcal{P}((\mathcal{R}!m) \otimes !v)) :!^2\mathrm{C}_S \\ \Sigma & \triangleright & \emptyset \mid \Delta \mid \emptyset \vdash !^2\mathtt{step} :!^2(\mathrm{C}_S \multimap \mathrm{C}_S) \\ \Phi & \triangleright & \emptyset \mid \Delta \mid \emptyset \vdash \mathcal{R}\ !m :!\mathrm{N} \end{array}$$

Let $\mathcal{N} = \mathtt{iter}_1(\mathtt{init}^+(\mathcal{P}(!m \otimes !u))(\mathcal{P}((\mathcal{R}!m) \otimes !v)))(!^2\mathtt{step})(\mathcal{R}\ !m)$. By applying $\emptyset \mid \Delta \mid \emptyset \vdash \mathtt{iter}_1 :!^2\mathrm{C}_S \multimap !^2(\mathrm{C}_S \multimap \mathrm{C}_S) \multimap !\mathrm{N} \multimap !^2\mathrm{C}_S$ to derivations Π, Σ, Φ, we obtain $\Psi \triangleright \emptyset \mid \Delta \mid \emptyset \vdash \mathcal{N} :!^2\mathrm{C}_S$.
Then the desired term $\mathcal{M}$ is the subject of the following derivation:

$$\dfrac{\dfrac{\vdots}{\emptyset \mid \Delta \mid \emptyset \vdash \mathtt{accept}_2 :!^2\mathrm{C}_S \multimap !^2\mathrm{B} \quad \Psi \triangleright \emptyset \mid \Delta \mid \emptyset \vdash \mathcal{N} :!^2\mathrm{C}_S}\,(\multimap E)}{\dfrac{\emptyset \mid \Delta \mid \emptyset \vdash \mathtt{accept}_2\mathcal{N} :!^2\mathrm{B}}{\vdash \lambda^!(m \otimes u \otimes v).\mathtt{accept}_2\mathcal{N} : (!\mathrm{N} \otimes !^2\mathrm{W}_S \otimes !^2\mathrm{W}_S) \multimap !^2\mathrm{B}}\,(\multimap I^I_{\otimes 3})}$$

where the derived rule $(\multimap I^{I}_{\otimes n})$ is the extension of the second rule of Table 2.2 to the n-ary case.

□

Chapter 3

Characterizing FPTIME and strong normalization through stratified types

In Chapter 2 we explored the widely known *ML-like* approach to ICC, based on the use of λ-calculus as paradigmatic programming language and on the design of a type assignment system for λ-terms based on Linear Logic, where types guarantee both the functional correctness and the desired complexity bound. A common drawback of all such systems can be found in their lack of typability: indeed, they all give type to a proper subset of the strongly normalizing terms, so basically cutting out a big slice of admissible algorithms.

In order to break such boundary, we propose a system of *stratified types*, which are clearly inspired by non-associative intersection types. This system, called STR, is correct and complete for polynomial time computations; moreover, all the strongly normalizing terms are typed in it, thus increasing the typing power with respect to the previous proposals based on subsystems of Linear Logic, while maintaining the language of pure λ-calculus. In particular, STR enjoys a stronger expressivity with respect to the previous system STA [GR07], since it allows to type a restricted version of iteration on binary words.

The aim is to design a type assignment system for λ-calculus, where types enforce both the functional correctness and the polynomial bound of terms. Earlier proposals along this line include the systems DLAL [BT04] and STA [GR07], both based on Light Logics, whose design is based on the inheritance of the complexity properties from the logic to the typed terms, according to the proofs-as-programs approach: more precisely, types of DLAL are a proper subset of formulae of LAL [AR02], while types of STA are a proper subset of formulae of SLL [Laf04]. The above mentioned systems both characterize the polynomial time functions, in the sense that all and only the

polynomial time functions can be encoded by typed terms in such systems, according to the standard coding of functions as λ-terms. Here, the logical inspiration is at the same time the key ingredient of their correctness and the reason for their weak expressivity, since it limits the set of algorithms which can be typed: indeed, from a typability point of view, both systems give types to a proper subset of the strongly normalizing terms.

While a stronger expressive power can be achieved by endowing the basic language with some stronger features [CS12; BGM10], here we wish to explore a different direction: namely, we want to maintain pure λ-calculus as a programming language, but on the other hand we aim to design a system with a stronger typability power in order to obtain, as a side effect, also a gain in expressivity, while at the same time preserving the desired polynomial bound.

The system we design, called `STR`, is polynomial -in the previous sense- and it allows to type all the strongly normalizing terms, thus increasing in a significant way the typability power with respect to both `DLAL` and `STA`; in particular `STR` is shown to be more expressive than `STA`, since a restricted form of iteration, which cannot be expressed in the latter, is typable in the former system.

From a logical point of view, while in `STA` the promotion is a sort of multiple contraction for different copies of the same premise, here we are able to contract also premises having different types. Observe that this feature can no longer be expressed in a logical way: indeed, `STR` is introduced directly as a type assignment system, without a pure logical counterpart.
In such design we were inspired by intersection types [CD80]; indeed the relation between `STA` and `STR` recalls, in a shallow way, the relation between simple types and intersection types assignment system, the latter being derived from the former by allowing a variable to be assigned different types.

Surprisingly enough, `STR` preserves the polynomial bound; indeed, the introduction of the intersection increases the typability power without increasing the computability power, as shown in [BPS03], in the sense that more programs are typed, but the class of functions characterized by such programs is exactly the same as `STA`, i.e. `FPTIME`.

Outline of this chapter We begin with a quick overview of `STA` and its properties (Section 3.1). We then introduce the type assignment system for λ-calculus (Section 3.2), where *stratified types* are essentially intersection types not enjoying the associative property, for which we prove the usual properties of generation and subject reduction.
We proceed to prove that `STR` characterizes strong normalization of λ-terms (Section 3.3). In order to prove that typed $\lambda^!$-terms of `STR` are strongly normalizing, we define a measure of weight on derivations, which is then proved to decrease after

reduction simply by retracing the proof of subject reduction and endowing derivations with their respective weights; this allows also to prove a bound on both the number of reduction steps and the size of the normal form with respect to the initial term. For the opposite direction, we start by proving an expansion lemma, through which we show that the three rules defining the set of strongly normalizing terms are derivable in STR.
In order to prove the polynomial characterization (Section 3.4), we show that STR and STA can compute the same class of functions: the completeness of STR with respect to FPTIME comes easily by giving a translation from the types of STR to the types of STA, while the soundness is shown by examining the reduction of a term representing the application of a function to its arguments.
Finally we comment on the choice of stratified types with respect to intersection types and we offer some technical observations on the use of intersection types for quantitative purposes (Section 3.5).
The current chapter is presented in [DR14].

3.1 An overview of the Soft Type Assignment system

The Soft Type Assignment system of [GR07] is a typing system for pure λ-calculus in (almost) natural deduction style, where types are a subset of the formulae of SLL [Laf04]. The set $\mathcal{TS}$ of STA-types is defined by the following syntax:

$$\begin{array}{lcll} \mathtt{U} & ::= & \mathtt{a} \mid \mu \multimap \mathtt{U} \mid \forall \mathtt{a}.\mathtt{U} & \text{(linear types)} \\ \mu, \nu & ::= & \mathtt{U} \mid !\mu & \text{(modal types)} \end{array}$$

Note that only strict types having linear types on the right-hand side of the arrow are considered, in order to design a system for which the subject reduction property holds.

A judgment of STA has the shape $\Theta \vdash_{\mathtt{STA}} \mathcal{M} : \mu$, where Θ is a partial function from term variables to types of $\mathcal{TS}$ (basis), $\mathcal{M}$ is a pure λ-term and μ is a type of $\mathcal{TS}$. The typing rules are given in Table 3.1.

In order to prove the complexity bound, some notions of measure are defined:

Definition 16 (Measures of STA)**.**

*i. The **rank** of a multiplexor rule* (m)

$$\frac{\Theta, x_1 : \mu, \ldots, x_n : \mu \vdash_{\mathtt{STA}} \mathcal{M} : \nu}{\Theta, x :! \mu \vdash_{\mathtt{STA}} \mathcal{M}[x/x_1, ..., x_n] : \nu}\ (m)$$

is the cardinality of the set $\{x_1, ..., x_n\} \cap \mathrm{FV}(\mathcal{M})$*. Let* r *be the maximum rank of all rules* (m) *in* Π*; then the rank* $\mathtt{rk_S}(\Pi)$ *of* Π *is the maximum between* 1 *and* r*.*

$$\frac{}{x : \mathtt{U} \vdash_{\mathtt{STA}} x : \mathtt{U}}\ (Ax) \qquad \frac{\Theta \vdash_{\mathtt{STA}} \mathcal{M} : \mu \quad x \notin dom\Theta}{\Theta, x : \mathtt{U} \vdash_{\mathtt{STA}} \mathcal{M} : \mu}\ (w)$$

$$\frac{\Theta, x : \mu \vdash_{\mathtt{STA}} \mathcal{M} : \mathtt{U}}{\Theta \vdash_{\mathtt{STA}} \lambda x.\mathcal{M} : \mu \multimap \mathtt{U}}\ (\multimap I) \qquad \frac{\Theta \vdash_{\mathtt{STA}} \mathcal{M} : \mu \multimap \mathtt{U} \quad \Xi \vdash_{\mathtt{STA}} \mathcal{N} : \mu \quad \Theta \# \Xi}{\Theta, \Xi \vdash_{\mathtt{STA}} \mathcal{M}\mathcal{N} : \mathtt{U}}\ (\multimap E)$$

$$\frac{\Theta, x_1 : \mu, \ldots, x_n : \mu \vdash_{\mathtt{STA}} \mathcal{M} : \nu}{\Theta, x :! \mu \vdash_{\mathtt{STA}} \mathcal{M}[x/x_1, ..., x_n] : \nu}\ (m) \qquad \frac{\Theta \vdash \mathcal{M} : \mu}{!\Theta \vdash_{\mathtt{STA}} \mathcal{M} :! \mu}\ (sp)$$

$$\frac{\Theta \vdash_{\mathtt{STA}} \mathcal{M} : \mathtt{U} \quad \mathtt{a} \notin FV(\Theta)}{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \forall \mathtt{a}.\mathtt{U}}\ (\forall I) \qquad \frac{\Theta \vdash_{\mathtt{STA}} \mathcal{M} : \forall \mathtt{a}.\mathtt{B}}{\Theta \vdash_{\mathtt{STA}} \mathcal{M} : \mathtt{B}[\mathtt{U}/\mathtt{a}]}\ (\forall E)$$

Table 3.1: The Soft Type Assignment (STA) system.

ii. *The* ***degree*** *of a proof* Π*, denoted by* $\mathtt{d_S}(\Pi)$*, is the maximal nesting of applications of the* (sp) *rule in* Π*, namely the maximal number of applications of the* (sp) *rule in any path connecting the conclusion with some axiom of* Π*.*

iii. *Let* r *be a positive integer. The* ***weight*** $\mathtt{W_S}(\Pi, r)$ *of* Π *with respect to* r *is defined inductively as follows:*

- *if* Π *ends with an application of rule* (Ax)*, then* $\mathtt{W_S}(\Pi, r) = 1$*;*
- *if* Π *ends with an application of rule* $(\multimap I)$ *with premise* Π'*, then* $\mathtt{W_S}(\Pi, r) = \mathtt{W_S}(\Pi', r) + 1$*;*
- *if* Π *ends with an application of rule* $(\multimap E)$ *with premises* Π_1 *and* Π_2*, then* $\mathtt{W_S}(\Pi, r) = \mathtt{W_S}(\Pi_1, r) + \mathtt{W_S}(\Pi_2, r) + 1$*;*
- *if* Π *ends with an application of rule* (sp) *with premise* Π'*, then* $\mathtt{W_S}(\Pi, r) = r \cdot \mathtt{W_S}(\Pi', r)$
- *in every other case,* $\mathtt{W_S}(\Pi, r) = \mathtt{W_S}(\Pi', r)$ *where* Π' *is the premise of the rule.*

Here we recall only the key technical property of STA, which is very similar to the property of STR which will be shown in Theorem 15:

Property 13 (Proved in [GR07])**.** *Let* $\Pi \triangleright \Theta \vdash_{\mathtt{STA}} \mathcal{M} : \mu$ *and* $\mathcal{M} \xrightarrow{*} \mathcal{M}'$ *in* m *steps; then:*

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

By employing Property 13 and the usual simulation of polytime Turing machines through λ-terms, the authors then show that:

Theorem 11 (Proved in [GR07]). STA *characterizes* FPTIME.

3.2 The STR typing system

In this section we introduce the STR type assignment system for pure λ-calculus, based on the notion of *stratification* of types, for which we prove that subject reduction holds.

3.2.1 Definition of the typing system

Calculus

As usual, the set Λ of λ-terms is defined by the following syntax:

$$\mathcal{M}, \mathcal{N} ::= x \mid \lambda x.\mathcal{M} \mid \mathcal{M}\mathcal{N}$$

where x ranges over a countable set of term variables Var.
The same assumptions made in Chapter 2 hold, namely we consider terms modulo α-equivalence and up to the variable convention. Free variables and substitution are defined as in Definitions 1 and 2, where the substitution $\mathcal{M}[\mathcal{N}_1/x_1, ..., \mathcal{N}_n/x_n]$, also denoted by $\mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n$, is the capture-free substitution of $\mathcal{N}_i$ to all the free occurrences of x_i in $\mathcal{M}$ $(1 \leq i \leq n)$. The symbol $=$ denotes the identity on terms, modulo renaming of bound variables.

A *term context* is generated by the same grammar, starting from a constant $\Box$ (the hole), in addition to variables. Term contexts are denoted by $\mathcal{C}$, while $\mathcal{C}[\mathcal{M}]$ denotes the result of plugging $\mathcal{M}$ into every occurrence of $\Box$ in $\mathcal{C}$; observe that, as usual, the plugging operation does not forbid the capture of free variables.

The reduction relation $\to$ is the contextual closure of the following rule:

$$(\lambda x.\mathcal{M})\mathcal{N} \to \mathcal{M}[\mathcal{N}/x]$$

and the relation $\overset{*}{\to}$ is the reflexive and transitive closure of $\to$.

Types

In Chapter 2 we considered an extended version of $\lambda^!$-calculus, where the stratification of the computation was directly readable on the calculus. As already mentioned earlier, here we have a different aim, namely using pure λ-calculus as a typed functional language: indeed, all information on the stratified structure of the computation is now stored in the types assigned to λ-terms.

To this aim, in addition to the usual notion of linear type, we specify *stratified types*. A stratified type is essentially a *set*, where the depth of each component is memorized through a stratified structure, so remembering the construction steps of the type. We start by giving a notion of *pre-types*:

Definition 17 (Pre-types)**.** *The set of pre-types is defined by the following syntax:*

$$\begin{array}{lcll} \mathtt{A} & ::= & \mathtt{a} \mid \sigma \multimap \mathtt{A} \mid \forall \mathtt{a}.\mathtt{A} & \textit{(linear pre-types)} \\ \sigma & ::= & \mathtt{A} \mid \{\underbrace{\sigma, ..., \sigma}_{n}\} & \textit{(stratified pre-types)} \end{array}$$

where $\mathtt{a}$ *ranges over a countable set of type variables and* $n \geq 1$*.*

Let $\sim$ denote the syntactical equality between (stratified) pre-types. On pre-types we define the following equivalence $=$, modulo renaming of bound variables:

- $\mathtt{A} \sim \mathtt{B}$ implies $\mathtt{A} = \mathtt{B}$;
- $\mathtt{A} = \mathtt{B}$ implies $\forall \mathtt{a}.\mathtt{A} = \forall \mathtt{a}.\mathtt{B}$;
- $\sigma = \tau$ and $\mathtt{A} = \mathtt{B}$ imply $\sigma \multimap \mathtt{A} = \tau \multimap \mathtt{B}$;
- $\{\sigma_1, ..., \sigma_n\} = \{\tau_1, ..., \tau_m\}$ if and only if $\forall i.\exists j.\sigma_i = \tau_j$ and $\forall j.\exists i.\sigma_i = \tau_j$ $(1 \leq i \leq n, 1 \leq j \leq m)$, namely, a stratified pre-type represents a set.

Example 8. *Consider the following pre-types:* $\sigma \sim \{\mathtt{a}, \mathtt{a}, \{\mathtt{b}\}\}$*,* $\tau \sim \{\{\mathtt{b}\}, \mathtt{a}\}$ *and* $\rho \sim \{\{\mathtt{a}\}, \mathtt{a}, \{\mathtt{b}\}\}$*. It is easy to check that* $\sigma = \tau$*, since both pre-types represent the same set, while both* $\sigma \neq \rho$ *and* $\tau \neq \rho$*.*

Among pre-types, we then isolate the class of types satisfying the above equivalence relation:

Definition 18 (Types)**.** *The types of* $\mathtt{STR}$*, whose set is denoted by* $\mathcal{T}$*, are pre-types modulo the equivalence relation* $=$*.*

In order to avoid reasoning modulo $=$, when writing $\{\sigma_1, ..., \sigma_n\}$ we assume that $\sigma_i \neq \sigma_j$, for $i \neq j$ $(1 \leq i, j \leq n)$, where $\sigma_1, ..., \sigma_n$ are the *components* of $\{\sigma_1, ..., \sigma_n\}$. Operations on sets are naturally extended to stratified types; in particular, we denote by $\cup_{i=1}^{n}\{\sigma_i\}$ the stratified type obtained by unifying the singletons $\{\sigma_1\}, ..., \{\sigma_n\}$. In order to avoid unnecessary parentheses, we assume that $\multimap$ takes precedence over $\forall$, that is $\forall \mathtt{a}.\sigma \multimap \mathtt{A}$ is equivalent to $\forall \mathtt{a}.(\sigma \multimap \mathtt{A})$.

A multiset over $\mathcal{T}$ is an unordered list $[\sigma_1, ..., \sigma_n]$, where the number of occurrences of σ_i is its *multiplicity*, and the multiset union $\uplus$ is the concatenation of lists. It is useful to be able to identify the elements of a stratified type:

Definition 19 (Linear components)**.** *The multiset of the linear components of σ, denoted by $\overline{\sigma}$, is defined inductively as*

$$\overline{\mathtt{A}} = [\mathtt{A}] \qquad \overline{\{\sigma_1, \dots, \sigma_k\}} = \overline{\sigma}_1 \uplus \dots \uplus \overline{\sigma}_k.$$

Example 9. *Let $\sigma = \{\mathtt{A}, \{\mathtt{A}, \mathtt{B}\}\}$: then $\overline{\sigma} = [\mathtt{A}, \mathtt{A}, \mathtt{B}]$.*

Bases

A basis is a partial function from variables to types, whose finite domain is represented by $\mathrm{dom}(\Gamma)$ as usual. The empty context is denoted by $\emptyset$.

Let $\{\sigma\}^n$ be a shorthand for $\underbrace{\{\dots\{}_{n}\,\sigma\,\underbrace{\}\dots\}}_{n}$, for every $n \geq 0$.

We introduce a few notations that are used throughout the chapter:

Notation 2 (Bases)**.**

- *Let Γ be a basis: then $\{\Gamma\}^n$ is the basis such that $\{\Gamma\}^n(x) = \{\Gamma(x)\}^n$.*
- *The condition $\Gamma_1 \# \dots \# \Gamma_n$, also written $\#_{i=1}^n \Gamma_i$, holds if and only if $j \neq h$ implies $\mathrm{dom}(\Gamma_j) \cap \mathrm{dom}(\Gamma_h) = \emptyset$, for $1 \leq j, h \leq n$.*
- *The condition $\Gamma_1 \boxminus \dots \boxminus \Gamma_n$, also written $\boxminus_{i=1}^n \Gamma_i$, holds if and only if $j \neq h$ implies $\mathrm{dom}(\Gamma_j) = \mathrm{dom}(\Gamma_h)$, for $1 \leq j, h \leq n$.*
- *If $\#_{i=1}^n \Gamma_i$, then $\Gamma_1, \dots, \Gamma_n$ is the basis such that $(\Gamma_1, \dots, \Gamma_n)(x) = \Gamma_i(x)$, where $x \in \mathrm{dom}(\Gamma_i)$ $(1 \leq i \leq n)$.*
- *If $\boxminus_{i=1}^n \Gamma_i$, then $\cup_{i=1}^n \{\Gamma_i\}$ is the basis such that $(\cup_{i=1}^n \{\Gamma_i\})(x) = \cup_{i=1}^n \{\Gamma_i(x)\}$.*

3.2.2 Properties of STR

System STR proves judgments of the kind $\Gamma \vdash \mathcal{M} : \sigma$, where Γ is a basis, $\mathcal{M}$ is a term and σ is a type; if the basis of a judgement is empty, then we abbreviate it by $\vdash \mathcal{M} : \sigma$. The rules of the system are shown in Table 3.2. Observe that the premises of rule $(\multimap E)$ must have disjoint sets of free variables; however, more general applications can be built by renaming term variables through the *multiplexor* rule (m). Moreover, retracing the behavior of rule (!) in SLL, the *stratification* rule (st) introduces the stratification on both the left and the right hand side. Finally, rule $(\forall E)$ allows to replace type variables only with linear types, in order to preserve the syntax.

Among the rules we can again distinguish between some *constructive rules*, namely (Ax), $(\multimap I)$ and $(\multimap E)$, which contribute to building the subject, and non-constructive ones. The latter can be further classified into *quantifier rules*,

$$\frac{}{x : \mathtt{A} \vdash x : \mathtt{A}}\ (Ax) \qquad \frac{\Gamma \vdash \mathcal{M} : \sigma \quad (x \notin \mathrm{dom}(\Gamma))}{\Gamma, x : \mathtt{A} \vdash \mathcal{M} : \sigma}\ (w)$$

$$\frac{\Gamma, x : \sigma \vdash \mathcal{M} : \mathtt{B}}{\Gamma \vdash \lambda x.\mathcal{M} : \sigma \multimap \mathtt{B}}\ (\multimap I)$$

$$\frac{\Gamma_1 \vdash \mathcal{M} : \sigma \multimap \mathtt{A} \quad \Gamma_2 \vdash \mathcal{N} : \sigma \quad (\Gamma_1 \# \Gamma_2)}{\Gamma_1, \Gamma_2 \vdash \mathcal{M}\mathcal{N} : \mathtt{A}}\ (\multimap E)$$

$$\frac{\Gamma, x_1 : \sigma_1, ..., x_n : \sigma_n \vdash \mathcal{M} : \tau}{\Gamma, x : \cup_{i=1}^{n}\{\sigma_i\} \vdash \mathcal{M}[x/x_1, ..., x/x_n] : \tau}\ (m)$$

$$\frac{(\Gamma_i \vdash \mathcal{M} : \sigma_i)_{1 \leq i \leq n} \quad \boxminus_{i=1}^{n}\Gamma_i}{\cup_{i=1}^{n}\{\Gamma_i\} \vdash \mathcal{M} : \{\sigma_1, ..., \sigma_n\}}\ (st)$$

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

Table 3.2: Derivation rules of system STR.

namely $(\forall I)$ and $(\forall E)$, which modify the types but do not affect the subject, and *renaming rules*, namely (w) and (m), renaming term variables or introducing new variables in the basis.
A sequence of applications of renaming (resp. quantifier) rules is called a *renaming* (resp. *quantifier*) *sequence*.

For renaming rules, we adopt the following notation:

Notation 3 (Renaming rules)**.** *The domain and the range of an application of rule* (m) *are respectively the set of variables contracted by it and the singleton of the new introduced variable.*
The domain and the range of an application of rule (w) *are respectively the empty set and the singleton of the new introduced variable.*

Considering the rules of Table 3.2, the domain and range of rule (m) are respectively $\{x_1, ..., x_n\}$ and $\{x\}$, while the range of rule (w) is $\{x\}$.

Based on the above notation, two applications of renaming rules are said to be *disjoint* if and only if both their domains and their ranges are disjoint.

Type derivations are denoted by Σ, Π. When writing $\Gamma \vdash \mathcal{M} : \sigma$ we mean that

there exists a derivation proving such statement, whereas $\Pi \triangleright \Gamma \vdash \mathcal{M} : \sigma$ identifies a particular derivation Π.
Given the application of a rule in a derivation, the derivations to which it is applied are its *premises*.

The following notations apply to derivations:

Notation 4 (Derivations)**.**

- *Given a derivation* Σ, $\mathrm{dom}(\Sigma)$ *represents the set of term variables* $\cup_{\Gamma \in \Sigma} \mathrm{dom}(\Gamma)$, *where* $\Gamma \in \Sigma$ *means that* Γ *is a basis occurring in the application of a rule in* Σ.

- *By abuse of notation, we denote by* $\Sigma_1 \# ... \# \Sigma_n$, *also written* $\#_{i=1}^{n} \Sigma_i$, *the fact that* $j \neq h$ *implies* $\mathrm{dom}(\Sigma_j) \cap \mathrm{dom}(\Sigma_h) = \emptyset$, *for* $1 \leq j, h \leq n$.

It is easy to show that a more general weakening rule is derivable:

Property 14 (Weakening)**.** $\Gamma \vdash \mathcal{M} : \sigma$ *and* $x \notin \mathrm{dom}(\Gamma)$ *imply* $\Gamma, x : \tau \vdash \mathcal{M} : \sigma$, *for every* τ.

Proof. By induction on τ.

If τ is linear, it is sufficient to apply rule (w).

Otherwise, let $\overline{\tau} = [\mathtt{A}_1, ..., \mathtt{A}_n]$: then we can build the following derivation:

$$\dfrac{\dfrac{\Gamma \vdash \mathcal{M} : \sigma}{\Gamma, x_1 : \mathtt{A}_1, ..., x_n : \mathtt{A}_n \vdash \mathcal{M} : \sigma}\ (w)}{\Gamma, x : \tau \vdash \mathcal{M} : \sigma}\ \delta$$

where δ is a suitable sequence of applications of rule (m). □

Example 10. *Let* $\Gamma \vdash \mathcal{M} : \sigma$ *and* $\tau = \{\{\mathtt{A}\}, \{\mathtt{A}, \mathtt{B}\}\}$, *so* $\overline{\tau} = [\mathtt{A}, \mathtt{A}, \mathtt{B}]$*: then we can build the following derivation:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash \mathcal{M} : \sigma}{\Gamma, y_1 : \mathtt{A}, y_2 : \mathtt{A}, y_3 : \mathtt{B} \vdash \mathcal{M} : \sigma}\ (w)}{\Gamma, x_1 : \{\mathtt{A}\}, y_2 : \mathtt{A}, y_3 : \mathtt{B} \vdash \mathcal{M} : \sigma}\ (m)}{\Gamma, x_1 : \{\mathtt{A}\}, x_2 : \{\mathtt{A}, \mathtt{B}\} \vdash \mathcal{M} : \sigma}\ (m)}{\Gamma, x : \{\{\mathtt{A}\}, \{\mathtt{A}, \mathtt{B}\}\} \vdash \mathcal{M} : \sigma}\ (m)$$

Note that, as a result of the previous property, the condition on the contexts in rule (st) is not at all restrictive.

In order to build a derivation for a given term, we first need to type a *linear* version of it, that is, a term where each free variable occurs exactly once. In order to better formalize such operation, we give the following notion of *instance* of a term:

Definition 20 (Instance of a term)**.** *A term $\mathcal{M}$ is an instance of $\mathcal{N}$ if and only if there are $\mathcal{X}_1, ..., \mathcal{X}_n$ disjoint subsets of $\mathrm{FV}(\mathcal{N})$ and fresh variables $y_1, ..., y_n$, for $n \geq 0$, such that $\mathcal{M}$ is obtained from $\mathcal{N}$ by replacing all variables in $\mathcal{X}_i$ with y_i.*
If all $\mathcal{X}_i$ are singletons and $\cup_{i=1}^{n}\mathcal{X}_\rangle = \mathrm{FV}(\mathcal{N})$, then $\mathcal{M}$ is a copy of $\mathcal{N}$; in particular, if $\cup_{i=1}^{n}\mathcal{X}_i$ and $\{y_1, ..., y_n\}$ are disjoint, then $\mathcal{M}$ is a disjoint copy of $\mathcal{N}$.

We supply an example in order to clarify the previous definition.

Example 11. *Let $\mathcal{N} = f_1(f_2(f_3x))$.*

- *The term $\mathcal{M} = y(y(zx))$ is an instance of $\mathcal{N}$, since there are $\mathcal{Y} = \{f_1, f_2\} \subseteq \mathrm{FV}(\mathcal{N})$, $\mathcal{Z} = \{f_3\} \subseteq \mathrm{FV}(\mathcal{N})$ and fresh variables y, z such that $\mathcal{M}$ is obtained from $\mathcal{N}$ by replacing variables in $\mathcal{Y}$ and $\mathcal{Z}$ with y and z respectively.*

- *The term $g_1(g_2(g_3y))$ is a disjoint copy of $\mathcal{N}$, obtained by replacing variables in the singletons $\{f_1\}$, $\{f_2\}$, $\{f_3\}$, $\{x\}$ with fresh variables g_1, g_2, g_3, y respectively.*

- *Consider the instance $f(f(fx))$ of $\mathcal{N}$: by abstracting over x and f, we obtain the Church representation of the integer 3. In general, the Church integer $\lambda f.\lambda x.f(f...(fx)...)$, for $n \geq 0$, can be assigned in* STR *both the uniform type $\mathrm{N} = \forall \mathtt{a}.\{\mathtt{a} \multimap \mathtt{a}\} \multimap \mathtt{a} \multimap \mathtt{a}$ and a parametric type $\mathrm{N}_m = \forall \mathtt{a}.\{\mathtt{a} \multimap \mathtt{a}\}^m \multimap \mathtt{a} \multimap \mathtt{a}$ for every $m \geq 1$.*
  The latter then allows to type the following programs on Church integers:

$$\begin{array}{lll}
\vdash \lambda y.\lambda f.\lambda x.f(yfx) : & \mathrm{N}_i \multimap \mathrm{N}_{i+1} & \textit{(successor)} \\
\vdash \lambda y.\lambda z.\lambda f.\lambda x.yf(zfx) : & \mathrm{N}_i \multimap \mathrm{N}_j \multimap \mathrm{N}_{\max\{i,j\}+1} & \textit{(addition)} \\
\vdash \lambda y.\lambda z.\lambda f.y(zf) : & \mathrm{N}_i \multimap \{\mathrm{N}_j\}^i \multimap \mathrm{N}_{i+j} & \textit{(multiplication)}
\end{array}$$

The notion of (disjoint) copy can be extended to bases and derivations in a straightforward way: in the former case, Γ is a copy of Δ (disjoint, if $\Gamma\#\Delta$) if and only if one is obtained from the other by renaming variables in it; in the latter, Π is a copy of Σ (disjoint, if $\Pi\#\Sigma$) if and only if both their bases and their subject are copies of each other.

The following lemma follows easily from the above definition:

Lemma 22 (Instance of a derivation)**.** *Let $\Pi \triangleright \Gamma \vdash \mathcal{M} : \sigma$ and let $\mathcal{N}$ be an instance of $\mathcal{M}$: then there is Δ such that $\Sigma \triangleright \Delta \vdash \mathcal{N} : \sigma$, where Σ is obtained from Π by a renaming sequence.*

Proof. By definition 20, there are $\mathcal{X}_1, ..., \mathcal{X}_n$ subsets of $\mathrm{FV}(\mathcal{M})$ and fresh variables $y_1, ..., y_n$, for some $n \geq 0$, such that $\mathcal{N}$ is obtained from $\mathcal{M}$ by renaming variables in $\mathcal{X}_i$ by y_i: then the desired derivation is obtained by n applications of rule (m), the i-th application having domain $\mathcal{X}_i$ and range $\{y_i\}$, for $1 \leq i \leq n$.

□

Due to the presence of many non-constructive rules, and in order to reason more easily about derivations in proofs, we again introduce a notion of *clean* derivation: in this context, a derivation is *clean* if, in every application of rule $(\multimap E)$ with premises Π_1 and Π_2, it holds that $\Pi_1 \# \Pi_2$.
The proof of the following property is then obvious, thanks to the renaming rule (m):

Property 15 (Clean derivation)**.** *For every derivation* Π*, there is a clean derivation* Σ *proving the same statement.*

Proof. By induction on Π.

The only interesting case is that of Π ending with an application of rule $(\multimap E)$, whose premises are Π_1 and Π_2. By inductive hypothesis there are clean derivations Σ_1 and Σ_2 proving the same statements. Since the bases of Σ_1 and Σ_2 are disjoint by definition of the rule, Σ is obtained by replacing the clean derivation Σ_1 with a copy of Σ_1 which is disjoint from Σ_2.

□

From now on, we implicitly assume that all derivations are clean.

The following is a key property of the system, which allows to deconstruct a derivation assigning a stratified type into its immediate premises assigning simpler types:

Property 16 (Subject with stratified type)**.** *Let* $\Pi \triangleright \Gamma \vdash \mathcal{M} : \{\sigma_1, \dots, \sigma_n\}$*; then there are* $\Pi_i \triangleright \Gamma_i \vdash \mathcal{N} : \sigma_i$ $(1 \leq i \leq n)$ *such that* Π *consists of an application of rule* (st) *with premises* $(\Pi_i)_{1 \leq i \leq n}$*, followed by a renaming sequence.*

Proof. By induction on Π. Observe that the last application of Π is either rule (w), (m) or (st).

Let Π end with an application of rule (w) or (m): then the proof follows easily by induction.

Otherwise, let Π be

$$\frac{(\Gamma_i \vdash \mathcal{M} : \sigma_i)_{1 \leq i \leq n}}{\cup_{i=1}^{n} \{\Gamma_i\} \vdash \mathcal{M} : \{\sigma_1, \dots, \sigma_n\}} \ (st)$$

Then the proof is trivial, since $\mathcal{M} = \mathcal{N}$ and the renaming sequence is empty.

□

Corollary 12. *Let* $\Gamma \vdash \mathcal{M} : \{\sigma_1, \dots, \sigma_n\}$ *and* $\overline{\sigma} = [\mathtt{A}_1, \dots, \mathtt{A}_n]$*; then there is* Γ_i *such that* $\Gamma_i \vdash \mathcal{M} : \mathtt{A}_i$ $(1 \leq i \leq n)$*.*

Proof. Easy, by applying repeatedly Property 16.

□

Observe that, by the previous Corollary, we can consider without loss of generality derivations assigning linear types to terms, since $\mathcal{M}$ being an instance of $\mathcal{N}$ implies that both terms have the same structure.
As usual, the generation lemma connects the shape of a term with its possible typings:

Lemma 23 (Generation)**.** *Let* $\Pi \rhd \Gamma \vdash \mathcal{M} : \mathtt{A}$ *and let* $\Sigma \rhd \Delta \vdash \mathcal{N} : \mathtt{B}$ *be the smallest subderivation of* Π *such that* Π *is obtained from* Σ *by a (possibly empty) renaming and quantifier sequence:*

i. if $\mathcal{M} = x$*, then* Σ *ends with an application of rule* (Ax)*;*

ii. if $\mathcal{M} = \lambda x.\mathcal{P}$*, then* Σ *ends with an application of rule* $(\multimap I)$ *and* $\mathtt{A} = \forall \overline{\mathtt{a}}.\sigma \multimap \mathtt{C}$ *for some* $\overline{\mathtt{a}}$*,* σ *and* $\mathtt{C}$*;*

iii. if $\mathcal{M} = \mathcal{P}\mathcal{Q}$*, then* Σ *ends with an application of rule* $(\multimap E)$*.*

Proof. Easy by considering each constructive rule. □

The following technical lemma shows that it is possible to mimic a sequence of applications of rule (st) through a renaming sequence, thus recovering the same basis:

Lemma 24 (From stratification to renaming)**.** *Let* $\Sigma_i \rhd \Theta_i \vdash \mathcal{N}_i : \sigma_i$ *be a copy of* $\Pi_i \rhd \Delta_i \vdash \mathcal{N} : \sigma_i$ *such that* $\#_{i=1}^n \Theta_i$ *and* $\boxminus_{i=1}^n \Delta_i$*, with* $\Delta = \cup_{i=1}^n \{\Delta_i\}$ $(1 \leq i \leq n)$*. Then, from any* $\Gamma, \Theta_1, ..., \Theta_n \vdash \mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n : \tau$ *such that* $\Gamma \# \Delta$*, we obtain* $\Gamma, \Delta \vdash \mathcal{M}[\mathcal{N}/x_i]_{i=1}^n : \tau$ *by a renaming sequence.*

Proof. By induction on Δ.

If $\Delta = \emptyset$, then $\mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n = \mathcal{M}[\mathcal{N}/x_i]_{i=1}^n$, so the renaming sequence is empty.

Otherwise, let $\Delta_i = \Delta_i', y : \rho_i$ and $\Delta = \Delta_i', y : \rho = \cup_{i=1}^n \{\Delta_i'\}, \cup_{i=1}^n \{\rho_i\}$. Moreover, let $\Theta_i', y_i : \rho_i \vdash \mathcal{N}_i[y_i/y] : \sigma_i$ be a copy of $\Delta_i', y : \rho_i \vdash \mathcal{N} : \sigma_i$ such that $\#_{i=1}^n \Theta_i'$ and $\boxminus_{i=1}^n \Delta_i'$, with $\Delta' = \cup_{i=1}^n \{\Delta_i'\}$ $(1 \leq i \leq n)$. By inductive hypothesis, from $\Gamma, \Theta_1', ..., \Theta_n', y_1 : \rho_1, ..., y_n : \rho_n \vdash \mathcal{M}[\mathcal{N}_i[y_i/y]/x_i]_{i=1}^n : \tau$ such that $\Gamma \# \Delta'$, we can derive $\Gamma, \Delta', y_1 : \rho_1, ..., y_n : \rho_n \vdash \mathcal{M}[\mathcal{N}[y_i/y]/x_i]_{i=1}^n : \tau$ by a renaming sequence: from this derivation we obtain $\Gamma, \Delta', y : \rho \vdash \mathcal{M}[\mathcal{N}/x_i]_{i=1}^n : \tau$ by one application of rule (m) with domain $\{y_1, ..., y_n\}$ and range $\{y\}$.

□

We supply an example in order to make the previous lemma clearer:

Example 12. *Let*

$$\frac{\Sigma_1 \rhd x : \sigma_1, y : \tau_1, z : \rho_1 \vdash \mathcal{N} : \phi_1 \quad \Sigma_2 \rhd x : \sigma_2, y : \tau_2, z : \rho_2 \vdash \mathcal{N} : \phi_2}{x : \{\sigma_1, \sigma_2\}, y : \{\tau_1, \tau_2\}, z : \{\rho_1, \rho_2\} \vdash \mathcal{N} : \{\phi_1, \phi_2\}} \ (st)$$

and consider

$$\Sigma_1' \triangleright x_1 : \sigma_1, y_1 : \tau_1, z_1 : \rho_1 \vdash \mathcal{N}_1 : \phi_1$$

$$\Sigma_2' \triangleright x_2 : \sigma_2, y_2 : \tau_2, z_2 : \rho_2 \vdash \mathcal{N}_2 : \phi_2$$

where $\mathcal{N}_1 \equiv \mathcal{N}[x_1/x, y_1/y, z_1/z]$ *and* $\mathcal{N}_2 \equiv \mathcal{N}[x_2/x, y_2/y, z_2/z]$*, as copies of* Σ_1 *and* Σ_2 *respectively.*

Then, from $\Gamma, x_1 : \sigma_1, y_1 : \tau_1, z_1 : \rho_1, x_2 : \sigma_2, y_2 : \tau_2, z_2 : \rho_2 \vdash \mathcal{M}[\mathcal{N}_1/w_1, \mathcal{N}_2/w_2] : \phi$ *such that* $\{x, y, z\} \cap \mathrm{dom}(\Gamma) = \emptyset$*, we obtain the following derivation simply by applying some renaming rules:*

$$\dfrac{\dfrac{\dfrac{\Gamma, x_1 : \sigma_1, y_1 : \tau_1, z_1 : \rho_1, x_2 : \sigma_2, y_2 : \tau_2, z_2 : \rho_2 \vdash \mathcal{M}[\mathcal{N}_1/w_1, \mathcal{N}_2/w_2] : \phi}{\Delta, x : \{\sigma_1, \sigma_2\}, y_1 : \tau_1, z_1 : \rho_1, y_2 : \tau_2, z_2 : \rho_2 \vdash \mathcal{M}[\mathcal{N}_1[x/x_1]/w_1, \mathcal{N}_2[x/x_2]/w_2] : \phi}\ (m)}{\Delta, x : \{\sigma_1, \sigma_2\}, y : \{\tau_1, \tau_2\}, z_1 : \rho_1, z_2 : \rho_2 \vdash \mathcal{M}[\mathcal{N}_1[x/x_1, y/y_1]/w_1, \mathcal{N}_2[x/x_2, y/y_2]/w_2] : \phi}\ (m)}{\Delta, x : \{\sigma_1, \sigma_2\}, y : \{\tau_1, \tau_2\}, z : \{\rho_1, \rho_2\} \vdash \mathcal{M}[\mathcal{N}/w_1, \mathcal{N}/w_2] : \phi}\ (m)$$

As usual, the substitution lemma is the prelude to the more important subject reduction property:

Lemma 25 (Substitution)**.** *Let* $\Pi \triangleright \Gamma, x_1 : \sigma_1, ..., x_n : \sigma_n \vdash \mathcal{M} : \tau$ *and* $\Sigma_i \triangleright \Delta_i \vdash \mathcal{N}_i : \sigma_i$*, for* $1 \leq i \leq n$*, such that* $\Gamma \# \Delta_1 \# ... \# \Delta_n$*; then there is a clean derivation*

$$\Phi \triangleright \Gamma, \Delta_1, ..., \Delta_n \vdash \mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n : \tau.$$

Proof. By induction on the shape of Π. Note that, since the hypothesis $\Gamma \# \Delta_1 \# ... \# \Delta_n$, it is immediate to see that Φ is clean.

Let Π be

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

Then $n = 1$, $\Sigma_1 \triangleright \Delta_1 \vdash \mathcal{N}_1 : \mathtt{A}$, and $\Phi = \Sigma_1$.

Let Π end with an application of rule (w) having range $\{y\}$. If $y \notin \{x_1, ..., x_n\}$, then the proof follows by induction. Otherwise, let $y \equiv x_1$ and let Π be

$$\dfrac{\Pi' \triangleright \Gamma, x_2 : \sigma_2, ..., x_n : \sigma_n \vdash \mathcal{M} : \tau \quad x_1 \notin \mathrm{dom}(\Gamma)}{\Gamma, x_1 : \mathtt{A}, x_2 : \sigma_2, ..., x_n : \sigma_n \vdash \mathcal{M} : \tau}\ (w)$$

so $\Sigma_1 \triangleright \Delta_1 \vdash \mathcal{N}_1 : \mathtt{A}$ and $\Sigma_i \triangleright \Delta_i \vdash \mathcal{N}_i : \sigma_i$ $(2 \leq i \leq n)$.
By inductive hypothesis, there is $\Phi' \triangleright \Gamma, \Delta_2, ..., \Delta_n \vdash \mathcal{M}[\mathcal{N}_2/x_2, ..., \mathcal{N}_n/x_n] : \tau$; then the desired proof is obtained from Φ' by Property 14.

Let Π end with an application of rule $(\multimap I)$: then the result follows easily by induction.

Let Π be

$$\dfrac{\Pi_1 \triangleright \Gamma_1 \vdash \mathcal{P} : \tau \multimap \mathtt{B} \quad \Pi_2 \triangleright \Gamma_2 \vdash \mathcal{Q} : \tau \quad \Gamma_1 \# \Gamma_2}{\Gamma_1, \Gamma_2 \vdash \mathcal{P}\mathcal{Q} : \mathtt{B}}\ (\multimap E)$$

where $\Gamma = \Gamma_1, \Gamma_2$. Since $\Gamma_1 \# \Gamma_2$, we can consider without loss of generality a partition of n such that $\Gamma_1 = \Gamma_1', x_1 : \sigma_1, ..., x_k : \sigma_k$ and $\Gamma_2 = \Gamma_2', x_{k+1} : \sigma_{k+1}, ..., x_n : \sigma_n$. By

inductive hypothesis there are derivations $\Phi_1 \triangleright \Gamma'_1, \Delta_1, ..., \Delta_k \vdash \mathcal{P}[\mathcal{N}/x] : \tau \multimap \mathtt{B}$ and $\Phi_2 \triangleright \Gamma'_2, \Delta_{k+1}, ..., \Delta_n \vdash \mathcal{Q}[\mathcal{N}/x] : \tau$: then the result follows by one application of rule $(\multimap E)$ to Φ_1 and Φ_2.

Let Π end with an application of rule (m) having range $\{y\}$.
If $y \notin \{x_1, ..., x_n\}$, then the proof follows by induction.
Otherwise, let $y \equiv x_1$ and let Π be

$$\frac{\Pi' \triangleright \Gamma, y_1 : \rho_1, \dots, y_h : \rho_h, x_2 : \sigma_2, \dots, x_n : \sigma_n \vdash \mathcal{P} : \tau}{\Gamma, x_1 : \sigma_1, x_2 : \sigma_2, \dots, x_n : \sigma_n \vdash \mathcal{M} : \tau}\ (m)$$

where $\mathcal{M} \equiv \mathcal{P}[x_1/y_k]_{k=1}^h$ and $\sigma_1 = \cup_{k=1}^h\{\rho_k\}$. Let $\sigma_1 = \{\mu_1, \dots, \mu_m\}$, for some $m \leq h$.
By Property 16, $\Sigma_1 \triangleright \Delta_1 \vdash \mathcal{N}_1 : \{\mu_1, \dots, \mu_m\}$ implies there are $\Psi_s \triangleright \Theta_s \vdash \mathcal{Q} : \mu_s$ $(1 \leq s \leq m)$ such that Σ_1 is obtained by an application of rule (st) to $(\Psi_s)_{s=1}^m$, followed by a renaming sequence δ.
Observe that, for every k $(1 \leq k \leq h)$, there is s_k such that $\rho_k = \mu_{s_k}$; moreover $\rho_k = \rho_{k'}$ implies $\Psi_{s_k} = \Psi_{s_{k'}}$. Let $\Psi'_{s_k} \triangleright \Theta'_{s_k} \vdash \mathcal{Q}_{s_k} : \mu_{s_k}$ be a copy of Ψ_{s_k}, for $1 \leq k \leq h$, such that $(\Gamma, \Delta_2, ..., \Delta_n)\#\Theta'_{s_1}\#...\#\Theta'_{s_h}$.
By inductive hypothesis $\Psi \triangleright \Gamma, \Theta'_{s_1}, ..., \Theta'_{s_h}, \Delta_2, ..., \Delta_n \vdash \mathcal{P}[\mathcal{Q}_{s_k}/y_k]_{k=1}^h[\mathcal{N}_i/x_i]_{i=2}^n : \tau$, from which we derive $\Pi'' \triangleright \Gamma, \cup_{k=1}^h\{\Theta_{s_k}\}, \Delta_2, ..., \Delta_n \vdash \mathcal{P}[\mathcal{Q}/y_k]_{k=1}^h[\mathcal{N}_i/x_i]_{i=2}^n : \tau$ by Lemma 24.
Note that $\Pi'' \triangleright \Gamma, \cup_{s=1}^m\{\Theta_s\}, \Delta_2, ..., \Delta_n \vdash \mathcal{P}[\mathcal{Q}/y_k]_{k=1}^h[\mathcal{N}_i/x_i]_{i=2}^n : \tau$ also holds, since $\cup_{k=1}^h\{\Theta_{s_k}\} = \cup_{s=1}^m\{\Theta_s\}$: then the desired derivation Φ is obtained by applying renaming sequence δ to Π''.

Let Π be

$$\frac{(\Pi_k \triangleright \Gamma_k, x_1 : \sigma_1^k, ..., x_n : \sigma_n^k \vdash \mathcal{M} : \tau_k)_{1 \leq k \leq h}}{\Gamma, x_1 : \sigma_1, ..., x_n : \sigma_n \vdash \mathcal{M} : \tau}\ (st)$$

where $\Gamma = \cup_{k=1}^h\{\Gamma_k\}$, $\tau = \{\tau_1, \dots, \tau_h\}$ and $\sigma_i = \cup_{k=1}^h\{\sigma_i^k\}$.
Let $\sigma_i = \{\rho_i^1, ..., \rho_i^{h_i}\}$, where $h_i \leq h$. By Property 16, $\Sigma_i \triangleright \Delta_i \vdash \mathcal{N}_i : \{\rho_i^1, ..., \rho_i^{h_i}\}$ implies there are $\Sigma_i^s \triangleright \Delta_i^s \vdash \mathcal{P}_i : \rho_i^s$ $(1 \leq s \leq h_i)$ such that Σ_i is obtained by an application of rule (st) to $(\Sigma_i^s)_{s=1}^{h_i}$, followed by a renaming sequence δ_i $(1 \leq i \leq n)$. Since by hypothesis $\Gamma\#\Delta_1\#...\#\Delta_n$, we can safely assume $\Pi\#\Sigma_1\#...\#\Sigma_n$. For $1 \leq i \leq n$ and $1 \leq k \leq h$, there is s_k such that $\sigma_i^k = \rho_i^{s_k}$; moreover, $\sigma_i^k = \sigma_i^{k'}$ implies $\Sigma_i^{s_k} = \Sigma_i^{s_{k'}}$.
By inductive hypothesis there are $\Phi_k \triangleright \Gamma_k, \Delta_1^{s_k}, ..., \Delta_n^{s_k} \vdash \mathcal{M}[\mathcal{P}_i/x_i]_{i=1}^n$, for $1 \leq k \leq h$; then we build $\Pi' \triangleright \Gamma, \cup_{k=1}^h\{\Theta_1^{s_k}\}, ..., \cup_{k=1}^h\{\Theta_n^{s_k}\} \vdash \mathcal{M}[\mathcal{P}_i/x_i]_{i=1}^n : \tau$ by applying rule (st) to $(\Phi_k)_{1 \leq k \leq h}$. Note that $\Pi' \triangleright \Gamma, \cup_{s=1}^{h_1}\{\Theta_1^s\}, ..., \cup_{s=1}^{h_n}\{\Theta_n^s\} \vdash \mathcal{M}[\mathcal{P}_i/x_i]_{i=1}^n : \tau$ because $\cup_{k=1}^h\{\Delta_i^{s_k}\} = \cup_{s=1}^{h_i}\{\Delta_i^s\}$: then, since by hypothesis $\#_{i=1}^n\Delta_i$, the desired derivation Φ is obtained by applying renaming sequences $\delta_1, ..., \delta_n$ consecutively to Π'.

The cases of rules $(\forall I)$ and $(\forall E)$ follow directly by induction, since the subject is not affected by the quantifier rules.

□

Another crucial ingredient for proving the subject reduction property is, as usual, is elimination of *detours* in a type derivation:

Definition 21 (Detour elimination)**.**

i. A $\forall$-detour is a derivation ending with an application of rule $(\forall I)$, immediately followed by an application of rule $(\forall E)$, which is erased by the following rule:

$$\dfrac{\dfrac{\Pi \triangleright \Gamma \vdash \mathcal{M} : \mathtt{B} \quad \mathtt{a} \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash \mathcal{M} : \forall \mathtt{a}.\mathtt{B}}\ (\forall I)}{\Gamma \vdash \mathcal{M} : \mathtt{B}[\mathtt{A}/\mathtt{a}]}\ (\forall E) \quad \mapsto \quad \Pi[\mathtt{A}/\mathtt{a}] \triangleright \Gamma \vdash \mathcal{M} : \mathtt{B}[\mathtt{A}/\mathtt{a}]$$

where $\Pi[\mathtt{A}/\mathtt{a}]$ denotes the derivation obtained from Π by replacing every occurrence of $\mathtt{a}$ by $\mathtt{A}$.

ii. A $\multimap$-detour is a derivation ending with an application of rule $(\multimap I)$, immediately followed by an application of rule $(\multimap E)$, which is erased by the following rule:

$$\dfrac{\dfrac{\Pi \triangleright \Gamma, x : \sigma \vdash \mathcal{M} : \mathtt{A}}{\Gamma \vdash \lambda x.\mathcal{M} : \sigma \multimap \mathtt{A}}\ (\multimap I) \qquad \Sigma \triangleright \Delta \vdash \mathcal{N} : \sigma}{\Gamma \vdash (\lambda x.\mathcal{M})\mathcal{N} : \mathtt{A}}\ (\multimap E) \quad \mapsto \quad \Phi \triangleright \Gamma, \Delta \vdash \mathcal{M}[\mathcal{N}/x] : \mathtt{A}$$

where Φ is defined in Lemma 25.

Observe that the operation of $\multimap$-detour elimination, if merely performed as shown above, is not guaranteed to produce a correct derivation. Consider for example a derivation ending with an application of rule (st) to $n \geq 1$ subderivations, whose subject contains a β-redex: in this case, a $\multimap$-detour having the same subject appears in n different subderivations, but eliminating only one of such $\multimap$-detours would result in an incorrect derivation.
Here the intuition is that one β-reduction may correspond to many detour eliminations; in practice, this happens in two cases: on one hand, when there are applications of quantifier rules in between the introduction and the elimination of the $\multimap$; on the other hand, when there is an application of rule (st) to $n \geq 1$ subderivations whose subject contains the current β-redex. In such cases, the reduction is performed by first erasing all $\forall$-detours in sequence, followed by the elimination of all $\multimap$-detours *simultaneously* in every premise of the application of rule (st).
In order to achieve this result, we need to be able to shift the applications of quantifier rules above a renaming sequence:

Lemma 26 (Lifting of quantifier rules)**.** *A sequence of applications of renaming and quantifier rules can be rearranged in such a way that the applications of the quantifier rules precede the applications of the renaming rules.*

Proof. Observe that quantifier rules only affect the type, while renaming rules deal with both the subject and the variables in the context. Let (R) and (Q) be respectively a renaming and a quantifier rule: it is sufficient to prove that the sequence of applications of rules $(R)(Q)$ can be replaced by the sequence of applications $(Q)(R)$.

If $(R) = (m)$ the proof is obvious; otherwise, if $(R) = (w)$ then the constraints for the application of rule $(\forall I)$ are obviously preserved, since the applications of rule (w) may introduce new type variables.

□

Finally the main result of the present section is shown:

Theorem 13 (Subject Reduction)**.** $\Gamma \vdash \mathcal{P} : \sigma$ *and* $\mathcal{P} \to \mathcal{Q}$ *implies* $\Gamma \vdash \mathcal{Q} : \sigma$.

Proof. If $\mathcal{P} \to \mathcal{Q}$, then there is a term context $\mathcal{C}$ such that $\mathcal{P} \equiv \mathcal{C}[(\lambda x.\mathcal{M})\mathcal{N}]$ and $\mathcal{Q} \equiv \mathcal{C}[\mathcal{M}[\mathcal{N}/x]]$: the proof is by induction on $\mathcal{C}$ and then by induction on σ.

Let $\mathcal{C} = \Box$, so $\Gamma \vdash (\lambda x.\mathcal{M})\mathcal{N} : \sigma$, and let σ be a linear type $\mathtt{A}$. By Lemma 23, Π is

$$\dfrac{\dfrac{\dfrac{\Pi' \triangleright \Theta'', x : \sigma'' \vdash \mathcal{M}'' : \mathtt{A}''}{\Theta'' \vdash \lambda x.\mathcal{M}'' : \sigma'' \multimap \mathtt{A}''}\,(\multimap I)}{\Theta' \vdash \lambda x.\mathcal{M}' : \sigma' \multimap \mathtt{A}'}\,\delta_1 \qquad \Sigma \triangleright \Delta' \vdash \mathcal{N}' : \sigma' \quad \Theta' \# \Delta'}{\Theta', \Delta' \vdash (\lambda x.\mathcal{M}')\mathcal{N}' : \mathtt{A}'}\,(\multimap E)}{\Theta, \Delta \vdash (\lambda x.\mathcal{M})\mathcal{N} : \mathtt{A}}\,\delta_2$$

where $\mathcal{M}$ is an instance of $\mathcal{M}'$ (which in turn is an instance of $\mathcal{M}''$) and $\mathcal{N}$ is an instance of $\mathcal{N}'$, $\Gamma = \Theta, \Delta$ and δ_1, δ_2 are renaming and quantifier sequences.
By Lemma 26, δ_1 can be replaced by δ', δ'', where δ' contains only quantifier rules and δ'' contains only renaming rules. Moreover, $\Theta'' \# \Delta'$ by the assumption that all derivations are clean. Then we can rewrite Π in the following way:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi' \triangleright \Theta'', x : \sigma'' \vdash \mathcal{M}'' : \mathtt{A}''}{\Theta'' \vdash \lambda x.\mathcal{M}'' : \sigma'' \multimap \mathtt{A}''}\,(\multimap I)}{\Theta'' \vdash \lambda x.\mathcal{M}'' : \sigma' \multimap \mathtt{A}'}\,\delta' \qquad \Sigma \triangleright \Delta' \vdash \mathcal{N}' : \sigma' \quad \Theta'' \# \Delta'}{\Theta'', \Delta' \vdash (\lambda x.\mathcal{M}'')\mathcal{N}' : \mathtt{A}'}\,(\multimap E)}{\Theta', \Delta' \vdash (\lambda x.\mathcal{M}')\mathcal{N}' : \mathtt{A}'}\,\delta''}{\Theta, \Delta \vdash (\lambda x.\mathcal{M})\mathcal{N} : \mathtt{A}}\,\delta_2$$

Let us assume the non-trivial case in which the sequence δ' is not empty; then, since $\sigma' \multimap \mathtt{A}'$ is a $\multimap$ type, sequence δ' must end with one application of $(\forall E)$ rule; however, since $\sigma'' \multimap \mathtt{A}''$ is also a $\multimap$ type, sequence δ' must contain a matching application of $(\forall I)$ rule: therefore, δ' contains a $\forall$-detour, which can be erased as shown in Definition 21.i. At this point, sequence δ_1 decreases by two applications of quantifier rules. By retracing the same step and thus erasing all $\forall$-detours in

sequence δ', we obtain the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi'' \triangleright \Theta'', x : \sigma' \vdash \mathcal{M}'' : \mathtt{A}'}{\Theta'' \vdash \lambda x.\mathcal{M}'' : \sigma' \multimap \mathtt{A}'}\,(\multimap I) \quad \Sigma \triangleright \Delta' \vdash \mathcal{N}' : \sigma' \quad \Theta' \# \Delta'}{\Theta'', \Delta' \vdash (\lambda x.\mathcal{M}'')\mathcal{N}' : \mathtt{A}'}\,(\multimap E)}{\Theta', \Delta' \vdash (\lambda x.\mathcal{M}')\mathcal{N}' : \mathtt{A}'}\,\delta''}{\Theta, \Delta \vdash (\lambda x.\mathcal{M})\mathcal{N} : \mathtt{A}}\,\delta_2$$

Finally, by applying Lemma 25 and substituting Σ in Π'' as in Definition 21.ii, the resulting derivation is

$$\dfrac{\dfrac{\Phi \triangleright \Theta'', \Delta' \vdash \mathcal{M}''[\mathcal{N}'/x] : \mathtt{A}'}{\Theta', \Delta' \vdash \mathcal{M}'[\mathcal{N}'/x] : \mathtt{A}'}\,\delta''}{\Theta, \Delta \vdash \mathcal{M}[\mathcal{N}/x] : \mathtt{A}}\,\delta_2$$

Notice that, since the property of a derivation being clean is preserved by substitution, the resulting proof is clean.

Now let $\mathcal{C} = \Box$ and $\sigma = \{\sigma_1, ..., \sigma_n\}$ for some $n \geq 1$.
By Property 16 there are $\Pi_i \triangleright \Gamma_i \vdash (\lambda x.\mathcal{M}')\mathcal{N}' : \sigma_i$ $(1 \leq i \leq n)$ and a renaming sequence δ such that Π has the following shape:

$$\dfrac{\dfrac{(\Pi_i \triangleright \Gamma_i \vdash (\lambda x.\mathcal{M}')\mathcal{N}' : \sigma_i)_{1 \leq i \leq n}}{\cup_{i=1}^{n}\{\Gamma_i\} \vdash (\lambda x.\mathcal{M}')\mathcal{N}' : \sigma}\,(st)}{\Gamma \vdash (\lambda x.\mathcal{M})\mathcal{N} : \sigma}\,\delta$$

By inductive hypothesis there are $\Phi_i \triangleright \Gamma_i \vdash \mathcal{M}'[\mathcal{N}'/x] : \sigma_i$, for $1 \leq i \leq n$; then the result follows by applying rule (st) to $\Phi_1, ..., \Phi_n$, followed by sequence δ.

The inductive cases of $\mathcal{C} \neq \Box$ are straightforward, since both the basis and the type are not affected by the subject reduction.

□

3.3 Strong normalization

In this section we show that system STR characterizes strong normalization, namely, that $\Gamma \vdash \mathcal{M} : \sigma$, for some Γ and σ, if and only if $\mathcal{M}$ is strongly normalizing; as usual, the result is proved by examining both directions of the implication.

3.3.1 Typability implies strong normalization

We first prove that, if a term $\mathcal{M}$ is typable in STR, then $\mathcal{M}$ is strongly normalizing.

The intuition is that, given a typed term $\mathcal{M}$, the stratified structure of its derivation allows to give a bound on the number of β-reduction steps necessary to reach the normal form of $\mathcal{M}$, which depends both on the size of $\mathcal{M}$ and on the nesting of applications of rule (st) in Π: namely, the number of reduction steps is

bounded by a polynomial in the size of the term, whose degree depends on the degree of the underlying type derivation. As a consequence, the reduction of a typed term to its normal form is performed in a finite number of step; therefore all typable terms in STR are strongly normalizing.

We start by giving a few necessary definitions of measure:

Definition 22 (Measures)**.**

i. The ***size*** $|\mathcal{M}|$ *of a term* $\mathcal{M}$ *is defined inductively as follows:*

$$|x| = 1 \qquad |\lambda x.\mathcal{M}| = |\mathcal{M}| + 1 \qquad |\mathcal{M}\mathcal{N}| = |\mathcal{M}| + |\mathcal{N}| + 1$$

ii. The ***size*** $|\Pi|$ *of a derivation* Π *is defined inductively as follows:*

- *if* Π *ends with an application of rule* (Ax)*, then* $|\Pi| = 1$*;*
- *if* Π *ends with an application of any other rule with* $n \geq 1$ *premises* $\Pi_1 ... \Pi_n$*, then* $|\Pi| = \sum_{i=1}^{n} |\Pi_i|$*.*

iii. The ***rank*** *of an application of rule* (m) *with domain* $\mathcal{X}$ *is the cardinality of the set* $\mathcal{X} \cap \mathrm{FV}(\mathcal{M})$*, that is, the number of variables in the domain of the rule appearing free in* $\mathcal{M}$*. Let* r *be the maximum rank of all applications of rule* (m) *in* Π*: then the rank of* Π*, denoted by* $\mathtt{rk}(\Pi)$*, is equal to* $\max\{1, r\}$*.*

iv. The ***degree*** *of a derivation* Π*, denoted by* $\mathtt{d}(\Pi)$*, is defined inductively as follows:*

- *if* Π *ends with an application of rule* (Ax)*, then* $\mathtt{d}(\Pi) = 0$*;*
- *if* Π *ends with an application of rule* $(\multimap\ I)$ *with premise* Π'*, then* $\mathtt{d}(\Pi) = \mathtt{d}(\Pi')$*;*
- *if* Π *ends with an application of rule* $(\multimap E)$ *with premises* Π_1 *and* Π_2*, then* $\mathtt{d}(\Pi) = \max\{\mathtt{d}(\Pi_1), \mathtt{d}(\Pi_2)\}$*;*
- *if* Π *ends with an application of rule* (st) *with premises* $(\Pi_i)_{i=1}^{n}$*, then* $\mathtt{d}(\Pi)r = \max_{i=1}^{n} \mathtt{d}(\Pi_i) + 1$*;*
- *if* Π *ends with an application of a renaming or quantifier rule with premise* Π'*, then* $\mathtt{d}(\Pi) = \mathtt{d}(\Pi')$*.*

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

- *if* Π *ends with an application of rule* (Ax)*, then* $\mathtt{W}(\Pi, r) = 1$*;*
- *if* Π *ends with an application of rule* $(\multimap\ I)$ *with premise* Π'*, then* $\mathtt{W}(\Pi, r) = \mathtt{W}(\Pi', r) + 1$*;*

- *if* Π *ends with an application of rule* $(\multimap E)$ *with premises* Π_1 *and* Π_2*, then* $\mathtt{W}(\Pi, r) = \mathtt{W}(\Pi_1, r) + \mathtt{W}(\Pi_2, r) + 1$*;*
- *if* Π *ends with an application of rule* (st) *with premises* $(\Pi_i)_{i=1}^n$*, then* $\mathtt{W}(\Pi, r) = r \cdot \max_{i=1}^n \mathtt{W}(\Pi_i, r)$*;*
- *if* Π *ends with an application of a renaming or quantifier rule with premise* Π'*, then* $\mathtt{W}(\Pi, r) = \mathtt{W}(\Pi', r)$.

Such measures are related to each other as shown by the following lemma:

Lemma 27 (Relations between measures)**.** *If* $\Pi \triangleright \Gamma \vdash \mathcal{M} : \sigma$*, then:*

i. $\mathtt{rk}(\Pi) \leq |\mathcal{M}| \leq |\Pi|$*;*

ii. $\mathtt{W}(\Pi, r) \leq r^{\mathtt{d}(\Pi)} \cdot \mathtt{W}(\Pi, 1)$*;*

iii. $\mathtt{W}(\Pi, 1) = |\mathcal{M}|$.

Proof. All three points are proved by induction on Π.

i. Let Π end with an application of rule (Ax): then $\mathtt{rk}(\Pi) = 0$ and $|x| = |\Pi| = 1$, so $0 \leq 1 \leq 1$ and the inequality is satisfied.

Let Π end with an application of either rule (w), rule $(\forall I)$ or rule $(\forall E)$, where Π' is the premise of the rule. By inductive hypothesis $\mathtt{rk}(\Pi') \leq |\mathcal{M}| \leq |\Pi'|$: then $\mathtt{rk}(\Pi) = \mathtt{rk}(\Pi') \leq |\mathcal{M}| \leq |\Pi'| < |\Pi|$.

Let Π end with an application of rule $(\multimap I)$, where Π' is the premise of the rule and $\mathcal{M} = \lambda x.\mathcal{P}$. By inductive hypothesis $\mathtt{rk}(\Pi') \leq |\mathcal{P}| \leq |\Pi'|$: then $\mathtt{rk}(\Pi) = \mathtt{rk}(\Pi') \leq |\mathcal{P}| < |\mathcal{M}| \leq |\Pi'| + 1 = |\Pi|$.

Let Π end with an application of rule $(\multimap E)$, where Π_1, Π_2 are the premises of the rule and $\mathcal{M} = \mathcal{P}\mathcal{Q}$. By inductive hypothesis $\mathtt{rk}(\Pi_1) \leq |\mathcal{P}| \leq |\Pi_1|$ and $\mathtt{rk}(\Pi_2) \leq |\mathcal{Q}| \leq |\Pi_2|$: then $\mathtt{rk}(\Pi) = \max\{\mathtt{rk}(\Pi_1), \mathtt{rk}(\Pi_2)\} < \mathtt{rk}(\Pi_1) + \mathtt{rk}(\Pi_2) + 1 \leq |\mathcal{P}| + |\mathcal{Q}| + 1 = |\mathcal{M}| \leq |\Pi_1| + |\Pi_2| + 1 = |\Pi|$.

Let Π end with an application of rule (m) of domain $\{x_1, ..., x_n\}$ and range $\{x\}$, where Π' is the premise of the rule and $\mathcal{M} = \mathcal{P}[x/x_i]_{i=1}^n$. By inductive hypothesis $\mathtt{rk}(\Pi') \leq |\mathcal{P}| \leq |\Pi'|$, with $n \leq |\mathcal{P}| = |\mathcal{M}|$: then

$$\mathtt{rk}(\Pi) = \begin{cases} \mathtt{rk}(\Pi') \leq |\mathcal{P}| = |\mathcal{M}| \leq |\Pi| & \text{if } \max\{\mathtt{rk}(\Pi'), n)\} = \mathtt{rk}(\Pi'); \\ n \leq |\mathcal{P}| = |\mathcal{M}| \leq |\Pi| & \text{otherwise.} \end{cases}$$

Finally, let Π end with an application of rule (st), where $\Pi_1, ..., \Pi_n$ are the premises of the rule. By inductive hypothesis $\mathtt{rk}(\Pi_i) \leq |\mathcal{M}| \leq |\Pi_i|$ for $1 \leq i \leq n$: then $\mathtt{rk}(\Pi) = \max_{i=1}^n \mathtt{rk}(\Pi_i) \leq |\mathcal{M}| \leq |\Pi_i| < |\Pi|$.

ii. Observe that $r^{\mathtt{d}(\Pi)} \geq 1$ for any $r \geq 1$.

Let Π end with an application of rule (Ax): then by definition $\mathtt{W}(\Pi, r) = \mathtt{W}(\Pi, 1) = 1$ and $\mathtt{d}(\Pi) = 0$, so $1 \leq r^0 \cdot 1$ and the inequality is satisfied.

Let Π end with an application of either rule (w), rule $(\forall I)$, rule $(\forall E)$ or rule (m), where Π' is the premise of the rule. By inductive hypothesis $\mathtt{W}(\Pi', r) \leq r^{\mathtt{d}(\Pi')} \cdot \mathtt{W}(\Pi', 1)$, and by definition $\mathtt{d}(\Pi) = \mathtt{d}(\Pi')$: then $\mathtt{W}(\Pi, r) = \mathtt{W}(\Pi', r) \leq r^{\mathtt{d}(\Pi)} \cdot \mathtt{W}(\Pi', 1) = r^{\mathtt{d}(\Pi)} \cdot \mathtt{W}(\Pi, 1)$.

Let Π end with an application of rule $(\multimap I)$, where Π' is the premise of the rule. By inductive hypothesis $\mathtt{W}(\Pi', r) \leq r^{\mathtt{d}(\Pi')} \cdot \mathtt{W}(\Pi', 1)$: then $\mathtt{W}(\Pi, r) = \mathtt{W}(\Pi', r) + 1 \leq r^{\mathtt{d}(\Pi)} \cdot \mathtt{W}(\Pi', 1) + 1 \leq r^{\mathtt{d}(\Pi)} \cdot \mathtt{W}(\Pi', 1) + r^{\mathtt{d}(\Pi)} < r^{\mathtt{d}(\Pi)} \cdot \mathtt{W}(\Pi, 1)$.

Let Π end with an application of rule $(\multimap E)$, where Π_1 and Π_2 are the premises of the rule. By inductive hypothesis $\mathtt{W}(\Pi_i, r) \leq r^{\mathtt{d}(\Pi_i)} \cdot \mathtt{W}(\Pi_i, 1)$ for $i \in \{1, 2\}$: then $\mathtt{W}(\Pi, r) = \mathtt{W}(\Pi_1, r) + \mathtt{W}(\Pi_2, r) + 1 \leq r^{\mathtt{d}(\Pi_1)} \cdot \mathtt{W}(\Pi_1, 1) + r^{\mathtt{d}(\Pi_2)} \cdot \mathtt{W}(\Pi_2, 1) + 1 \leq r^{\mathtt{d}(\Pi)} \cdot \mathtt{W}(\Pi_1, 1) + r^{\mathtt{d}(\Pi)} \cdot \mathtt{W}(\Pi_2, 1) + r^{\mathtt{d}(\Pi)} = r^{\mathtt{d}(\Pi)} \cdot \mathtt{W}(\Pi, 1)$.

Finally, let Π end with an application of rule (st), where $\Pi_1, ..., \Pi_n$ are the premises of the rule. By inductive hypothesis $\mathtt{W}(\Pi_i, r) \leq r^{\mathtt{d}(\Pi_i)} \cdot \mathtt{W}(\Pi_i, 1)$ for $1 \leq i \leq n$, and in particular $\max_{i=1}^n \mathtt{W}(\Pi_i, r) \leq r^{\max_{i=1}^n \mathtt{d}(\Pi_i)} \cdot \max_{i=1}^n \mathtt{W}(\Pi_i, 1)$: then $\mathtt{W}(\Pi, r) = r \cdot \max_{i=1}^n \mathtt{W}(\Pi_i, r) \leq r \cdot r^{\max_{i=1}^n \mathtt{d}(\Pi_i)} \cdot \max_{i=1}^n \mathtt{W}(\Pi_i, 1) = r^{\max_{i=1}^n \mathtt{d}(\Pi_i)+1} \cdot \max_{i=1}^n \mathtt{W}(\Pi_i, 1) = r^{\mathtt{d}(\Pi)} \cdot \mathtt{W}(\Pi, 1)$.

iii. Let Π end with an application of rule (Ax): then $\mathtt{W}(\Pi, 1) = 1$ and $|x| = 1$, so the equality is satisfied.

Let Π end with an application of either rule (w), rule $(\forall I)$ or rule $(\forall E)$, where Π' is the premise of the rule. By inductive hypothesis $\mathtt{W}(\Pi', 1) = |\mathcal{M}|$: then $\mathtt{W}(\Pi, 1) = \mathtt{W}(\Pi', 1) = |\mathcal{M}|$.

Let Π end with an application of rule $(\multimap I)$, where Π' is the premise of the rule. By inductive hypothesis $\mathtt{W}(\Pi', 1) = |\mathcal{M}|$: then $\mathtt{W}(\Pi, 1) = \mathtt{W}(\Pi', 1) + 1 = |\mathcal{M}| + 1 = |\lambda x.\mathcal{M}|$.

Let Π end with an application of rule $(\multimap E)$, where Π_1 and Π_2 are the premises of the rule. By inductive hypothesis $\mathtt{W}(\Pi_1, 1) = |\mathcal{M}|$ and $\mathtt{W}(\Pi_2, 1) = |\mathcal{N}|$: then $\mathtt{W}(\Pi, 1) = \mathtt{W}(\Pi_1, 1) + \mathtt{W}(\Pi_2, 1) + 1 = |\mathcal{M}| + |\mathcal{N}| + 1 = |\mathcal{M}\mathcal{N}|$.

Let Π end with an application of rule (m) of domain $\{x_1, ..., x_n\}$ and range $\{x\}$, where Π' is the premise of the rule. By inductive hypothesis $\mathtt{W}(\Pi', 1) = |\mathcal{M}|$: then $\mathtt{W}(\Pi, 1) = \mathtt{W}(\Pi', 1) = |\mathcal{M}| = |\mathcal{M}[x_i/x]_{i=1}^n|$.

Finally, let Π end with an application of rule (st), where $\Pi_1, ..., \Pi_n$ are the premises of the rule. By inductive hypothesis $\mathtt{W}(\Pi_i, 1) = |\mathcal{M}|$ for $1 \leq i \leq n$, and in particular $\max_{i=1}^n \mathtt{W}(\Pi_i, 1) = |M|$: then $\mathtt{W}(\Pi, 1) = 1 \cdot \max_{i=1}^n \mathtt{W}(\Pi_i, 1) = |\mathcal{M}|$.

□

Note that copies of a derivation have the same measures:

Property 17 (Size of a copy)**.** *Let* Π' *be a copy of* Π*: then* $\mathtt{rk}(\Pi') = \mathtt{rk}(\Pi)$*,* $\mathtt{d}(\Pi') = \mathtt{d}(\Pi)$ *and* $\mathtt{W}(\Pi', r) = \mathtt{W}(\Pi, r)$ *for every* $r \geq 1$*.*

Proof. Easy, since Π and Π' have exactly the same structure.

□

Using the results obtained so far, we are able to state a sort of weighted version of Lemma 25, where the substitution is shown not to increase the overall weight of the derivation:

Lemma 28 (Weighted substitution)**.** *Let* $\Pi \triangleright \Gamma, x_1 : \sigma_1, ..., x_n : \sigma_n \vdash \mathcal{M} : \tau$ *and* $\Sigma_i \triangleright \Delta_i \vdash \mathcal{N}_i : \sigma_i$*, for* $1 \leq i \leq n$*, such that* $\{x_1, ..., x_n\} \cap \mathrm{FV}(\mathcal{M}) = \{x_{i_1}, ..., x_{i_p}\}$ *and* $\Gamma \# \Delta_1 \# ... \# \Delta_n$*; then there is* $\Phi \triangleright \Gamma, \Delta_1, ..., \Delta_n \vdash \mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n : \sigma$ *such that*

$$\mathtt{W}(\Phi, r) \leq \mathtt{W}(\Pi, r) + \sum_{s=1}^{p} \mathtt{W}(\Sigma_{i_s}, r)$$

for every $r \geq \max\{\mathtt{rk}(\Sigma_1), ..., \mathtt{rk}(\Sigma_n), \mathtt{rk}(\Pi)\}$*.*

Proof. By induction on Π. Since the first part of the statement was already shown to be valid in the proof of Lemma 25, we refer to the notation used in that proof in order to prove that the given inequality holds.

Let Π end with an application of rule (Ax): then $\mathtt{W}(\Phi, r) = \mathtt{W}(\Sigma_1, r)$, so the proof is obvious.

Let Π end with an application of rule (w), having range $\{y\}$, to the premise Π'. If $y \notin \{x_1, ..., x_n\}$, then the proof follows by induction. Otherwise, let $y = x_1$, so $x_1 \notin \{x_{i_1}, ..., x_{i_p}\}$. By inductive hypothesis $\mathtt{W}(\Phi', r) \leq \mathtt{W}(\Pi', r) + \sum_{s=1}^{p} \mathtt{W}(\Sigma_{i_s}, r)$; then

$$\begin{aligned} \mathtt{W}(\Phi, r) = \mathtt{W}(\Phi', r) &\leq \mathtt{W}(\Pi', r) + \sum_{s=1}^{p} \mathtt{W}(\Sigma_{i_s}, r) \\ &= \mathtt{W}(\Pi, r) + \sum_{s=1}^{p} \mathtt{W}(\Sigma_{i_s}, r) \end{aligned}$$

Let Π end with an application of rule $(\multimap I)$: then the result follows easily by induction.

Let Π end with an application of rule $(\multimap E)$ to Π_1 and Π_2; moreover, let $\{x_1, ..., x_k\} \cap \mathrm{FV}(\mathcal{M}) = \{x_{i_1}, ..., x_{i_q}\}$ and $\{x_{k+1}, ..., x_n\} \cap \mathrm{FV}(\mathcal{P}) = \{x_{i_{q+1}}, ..., x_{i_p}\}$. By

inductive hypothesis on Π_1 and Π_2 respectively, $\mathtt{W}(\Phi_1, r) \leq \mathtt{W}(\Pi_1, r) + \sum_{s=1}^{q} \mathtt{W}(\Sigma_{i_s}, r)$ and $\mathtt{W}(\Phi_2, r) \leq \mathtt{W}(\Pi_2, r) + \sum_{s=q+1}^{p} \mathtt{W}(\Sigma_{i_s}, r)$; then

$$\begin{aligned}\mathtt{W}(\Phi, r) &= \mathtt{W}(\Phi_1, r) + \mathtt{W}(\Phi_2, r) + 1 \\ &\leq \mathtt{W}(\Pi_1, r) + \mathtt{W}(\Pi_2, r) + 1 + \sum_{s=1}^{q} \mathtt{W}(\Sigma_{i_s}, r) + \sum_{s=q+1}^{p} \mathtt{W}(\Sigma_{i_s}, r) \\ &= \mathtt{W}(\Pi, r) + \sum_{s=1}^{p} \mathtt{W}(\Sigma_{i_s}, r)\end{aligned}$$

Let Π end with an application of rule (m), having range $\{y\}$, to Π'.
If $y \notin \{x_{i_1}, ..., x_{i_p}\}$, then the proof follows by induction.
Otherwise let $y = x_1$. Let us assume that $\{y_1, ..., y_h\} \cap \mathrm{FV}(\mathcal{P}) = \{y_{j_1}, ..., y_{j_q}\}$ and $\{x_2, ..., x_n\} \cap \mathrm{FV}(\mathcal{M}) = \{x_{i_1}, ..., x_{i_p}\}$. By inductive hypothesis on the premise Π', $\mathtt{W}(\Psi, r) \leq \mathtt{W}(\Pi', r) + \sum_{l=1}^{q} \mathtt{W}(\Psi'_{s_{j_l}}, r) + \sum_{s=1}^{p} \mathtt{W}(\Sigma_{i_s}, r)$.
Note that $\mathtt{W}(\Sigma_1, r) = r \cdot \max_{s=1}^{m} \mathtt{W}(\Psi_s, r) = r \cdot \max_{k=1}^{h} \mathtt{W}(\Psi'_{s_k}, r)$ by Property 17. Moreover, $\mathtt{W}(\Pi, r) = \mathtt{W}(\Pi', r)$ and $q \leq \mathtt{rk}(\Pi) \leq r$.
Since δ is a renaming sequence, by definition $\mathtt{W}(\Phi, r) = \mathtt{W}(\Psi, r)$; then

$$\begin{aligned}\mathtt{W}(\Phi, r) &= \mathtt{W}(\Phi, r) \\ &\leq \mathtt{W}(\Pi', r) + \sum_{l=1}^{q} \mathtt{W}(\Psi'_{s_{j_l}}, r) + \sum_{s=1}^{p} \mathtt{W}(\Sigma_{i_s}, r) \\ &\leq \mathtt{W}(\Pi', r) + r \cdot \max_{l=1}^{q} \mathtt{W}(\Psi'_{s_{j_l}}, r) + \sum_{s=1}^{p} \mathtt{W}(\Sigma_{i_s}, r) \\ &= \mathtt{W}(\Pi, r) + r \cdot \max_{k=1}^{h} \mathtt{W}(\Psi'_{s_k}, r) + \sum_{s=1}^{p} \mathtt{W}(\Sigma_{i_s}, r) \\ &= \mathtt{W}(\Pi, r) + \mathtt{W}(\Sigma_1, r) + \sum_{s=1}^{p} \mathtt{W}(\Sigma_{i_s}, r)\end{aligned}$$

Let Π end with an application of rule (st) to the premises $(\Pi_k)_{1 \leq k \leq h}$. Note that $\mathtt{W}(\Sigma_i, r) = r \cdot \max_{s=1}^{h_i} \mathtt{W}(\Sigma_i^s, r) = r \cdot \max_{k=1}^{h} \mathtt{W}(\Sigma_i^{s_k}, r)$ $(1 \leq i \leq n)$. By induction hypothesis $\mathtt{W}(\Phi_k, r) \leq \mathtt{W}(\Pi_k, r) + \sum_{j=1}^{p} \mathtt{W}(\Sigma_{i_j}^{s_k}, r)$, for every $1 \leq k \leq h$; then

$$\begin{aligned}\mathtt{W}(\Phi, r) &= r \cdot \max_{k=1}^{h} \mathtt{W}(\Phi_k, r) \\ &\leq r \cdot \max_{k=1}^{h} \left(\mathtt{W}(\Pi_k, r) + \sum_{j=1}^{p} \mathtt{W}(\Sigma_{i_j}^{s_k}, r) \right) \\ &< r \cdot \max_{k=1}^{h} \mathtt{W}(\Pi_k, r) + r \cdot \max_{k=1}^{h} \sum_{j=1}^{p} \mathtt{W}(\Sigma_{i_j}^{s_k}, r) \\ &\leq r \cdot \max_{k=1}^{h} \mathtt{W}(\Pi_k, r) + \sum_{j=1}^{p} \left(r \cdot \max_{k=1}^{h} \mathtt{W}(\Sigma_{i_j}^{s_k}, r) \right) \\ &= \mathtt{W}(\Pi, r) + \sum_{j=1}^{p} \mathtt{W}(\Sigma_{i_j}, r)\end{aligned}$$

Let Π end with an application of a quantifier rule: then the result follows by induction.

□

Observe that erasing a $\forall$-detour does not change the weight of the proof; then we can easily prove that the weight of a proof strictly decreases with each normalization step:

Theorem 14 (Weighted subject reduction)**.** *$\Pi \triangleright \Gamma \vdash \mathcal{P} : \sigma$ and $\mathcal{P} \to \mathcal{Q}$ implies there is $\Psi \triangleright \Gamma \vdash \mathcal{Q} : \sigma$, such that $\mathtt{W}(\Psi, r) < \mathtt{W}(\Pi, r)$ for every $r \geq \mathtt{rk}(\Pi)$.*

Proof. Let $\mathcal{P} = \mathcal{C}[(\lambda x.\mathcal{M})\mathcal{N}]$ and $\mathcal{Q} = \mathcal{C}[\mathcal{M}[\mathcal{N}/x]]$. We proceed by induction on the term context and then by induction on σ. We already proved in Theorem 13 that the subject reduction holds; we refer to the notation used in that proof in order to prove that the stated inequality holds.

Let $\mathcal{C} = \Box$ and let σ be a linear type. Since δ_1, δ_2 are sequences of renaming and quantifier rules, $\mathtt{W}(\Pi, r) = \mathtt{W}(\Pi', r) + \mathtt{W}(\Sigma, r) + 2 = \mathtt{W}(\Pi'', r) + \mathtt{W}(\Sigma, r) + 2$: then $\mathtt{W}(\Psi, r) = \mathtt{W}(\Phi, r) \leq \mathtt{W}(\Sigma, r) + \mathtt{W}(\Pi'', r) < \mathtt{W}(\Pi, r)$ by Lemma 28.

Let $\mathcal{C} = \Box$ and let $\sigma = \{\sigma_1, ..., \sigma_n\}$. By inductive hypothesis $\mathtt{W}(\Phi_i, r) < \mathtt{W}(\Pi_i, r)$, for $1 \leq i \leq n$: then $\mathtt{W}(\Psi, r) = r \cdot \max_{i=1}^{n} \mathtt{W}(\Phi_i, r) < r \cdot \max_{i=1}^{n} \mathtt{W}(\Pi_i, r) = \mathtt{W}(\Pi, r)$.

If $\mathcal{C} \neq \Box$, the proof follows easily by induction.

□

From such results we can easily derive both the complexity bound and the strong normalization property:

Theorem 15. *If $\Pi \triangleright \Gamma \vdash \mathcal{M} : \sigma$ and let $\mathcal{M} \xrightarrow{*} \mathcal{M}'$ in m steps, then:*

i. $m \leq |\mathcal{M}|^{\mathtt{d}(\Pi)+1}$;

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

Proof. Let $r = \mathtt{rk}(\Pi)$ and let $\Pi' \triangleright \Gamma \vdash \mathcal{M}' : \sigma$.

i. By Lemma 27, $r \leq |\mathcal{M}|$ and $\mathtt{W}(\Pi, r) \leq r^{\mathtt{d}(\Pi)} \cdot \mathtt{W}(\Pi, 1) = r^{\mathtt{d}(\Pi)} \cdot |\mathcal{M}| \leq |\mathcal{M}|^{\mathtt{d}(\Pi)} \cdot |\mathcal{M}|$. Let $\mathcal{M} \to \mathcal{M}_1 \to \mathcal{M}_2 \to ... \to \mathcal{M}_m = \mathcal{M}'$, with $m \geq 1$. By Theorem 14, if $\mathcal{M}$ reduces to $\mathcal{M}_1$ in one β-reduction step, then there is $\Pi_1 \triangleright \Gamma \vdash \mathcal{M}_1 : \sigma$ such that $\mathtt{W}(\Pi_1, r) \leq \mathtt{W}(\Pi, r) - 1$; if $\mathcal{M}_1$ reduces to $\mathcal{M}_2$ in one β-reduction step, then there is $\Pi_2 \triangleright \Gamma \vdash \mathcal{M}_2 : \sigma$ such that $\mathtt{W}(\Pi_2, r) \leq \mathtt{W}(\Pi_1, r) - 1 \leq \mathtt{W}(\Pi, r) - 2$, and so on. After m reduction steps, we obtain $\mathtt{W}(\Pi', r) \leq \mathtt{W}(\Pi, r) - m$, so $m \leq \mathtt{W}(\Pi, r)$: by substituting $\mathtt{W}(\Pi, r) \leq |\mathcal{M}|^{\mathtt{d}(\Pi)+1}$ in the above expression, we finally obtain $m \leq |\mathcal{M}|^{\mathtt{d}(\Pi)+1}$.

$$\frac{\mathcal{M}_1 \in \mathrm{SN} \quad \ldots \quad \mathcal{M}_n \in \mathrm{SN}}{x\mathcal{M}_1 \ldots \mathcal{M}_n \in \mathrm{SN}} \tag{3.1}$$

$$\frac{\mathcal{M} \in \mathrm{SN}}{\lambda x.\mathcal{M} \in \mathrm{SN}} \tag{3.2}$$

$$\frac{\mathcal{M}[\mathcal{N}/x]\mathcal{M}_1 \ldots \mathcal{M}_n \in \mathrm{SN} \quad \mathcal{N} \in \mathrm{SN}}{(\lambda x.\mathcal{M})\mathcal{N}\mathcal{M}_1 \ldots \mathcal{M}_n \in \mathrm{SN}} \tag{3.3}$$

Table 3.3: Rules defining the set SN.

ii. By Theorem 14, $\mathtt{W}(\Pi', r) < \mathtt{W}(\Pi, r)$. Since $\mathtt{W}(\Pi', 1) \leq \mathtt{W}(\Pi', r)$ and by Lemma 27 $|\mathcal{M}'| = \mathtt{W}(\Pi', 1)$, we obtain

$$|\mathcal{M}'| \leq \mathtt{W}(\Pi, r) \leq r^{\mathtt{d}(\Pi)} \cdot \mathtt{W}(\Pi, 1) = r^{\mathtt{d}(\Pi)} \cdot |\mathcal{M}| \leq |\mathcal{M}|^{\mathtt{d}(\Pi)+1}.$$

□

Since the number of steps in the reduction path for a given term is a finite number, we also obtain a proof of strong normalization:

Theorem 16. *If a term $\mathcal{M}$ is typed in* STR*, then $\mathcal{M}$ is strongly normalizing.*

Proof. Easy by Theorem 15.

□

Note that, as usual, a typed term can be assigned an infinite number of types by fiddling with non-constructive rules; in particular, since stratifications can be added ad libitum, different derivations of the same term may supply a different bound on the number of its normalization steps.
Nonetheless, a minimal bound on the normalization time of a term can be easily obtained, simply by choosing a derivation with *minimal* depth in which "useless" applications of non-constructive rules are avoided.

3.3.2 Strong normalization implies typability

We now show the converse implication, namely that all strongly normalizing terms are typed in STR, by following the technique of [Val01; RSSX99; BDS13].
The set of strongly normalizing terms, denoted by SN, is the smallest set of terms closed under the rules depicted in Table 3.3.

For each of these rules, we prove that if the premises of the rule are typable then the conclusion is also typable; in particular, in order to prove such property for Rule (3.3), we need to show that the expansion holds for typed terms of STR.
To this aim we start by giving a sort of inversion of Lemma 25:

Lemma 29 (Inverted substitution)**.** *Let* $\Gamma, \Delta \vdash \mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n : \tau$*, where* $\mathcal{N}_1, ..., \mathcal{N}_n$ *are typable in* STR*,* $\mathrm{dom}(\Gamma) = \mathrm{FV}(\mathcal{M}) \setminus \{x_1, ..., x_n\}$ *and* $\{x_1, ..., x_n\} \cap \mathrm{dom}(\Delta) = \emptyset$*; then there are* $\Delta_i \vdash \mathcal{N}_i : \sigma_i$*, for* $1 \leq i \leq n$*, such that* $\Gamma, x_1 : \sigma_1, .., x_n : \sigma_n \vdash \mathcal{M} : \tau$*.*

Proof. Let Π be a derivation proving $\Gamma, \Delta \vdash \mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n : \tau$. The proof is by induction on Π. The key cases, to be considered more carefully, are the ones of rule $(\multimap E)$ and (m).

Let Π be

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

so $\mathcal{M}$ is a variable. By hypothesis $\mathcal{N}_i$ is typable: then there is $\Sigma_i \triangleright \Delta_i \vdash \mathcal{N}_i : \mathtt{A}_i$ for some linear type $\mathtt{A}_i$, for $1 \leq i \leq n$.
If $\mathcal{M} = x_k$, for $1 \leq k \leq n$, we have $\Gamma = \emptyset$ and $\mathcal{N}_k = x$: then $x_1 : \mathtt{A}_1, ..., x_n : \mathtt{A}_n \vdash x_k : \mathtt{A}_k$ follows from axiom $x_k : \mathtt{A}_k \vdash x_k : \mathtt{A}_k$ by Property 14. Otherwise we have $\mathcal{M} = x$ and $x \notin \{x_1, ..., x_n\}$, so $\Gamma = x : \mathtt{A}$: then $x : \mathtt{A}, x_1 : \mathtt{A}_1, ..., x_n : \mathtt{A}_n \vdash x : \mathtt{A}$ follows from axiom $x : \mathtt{A} \vdash x : \mathtt{A}$ by Property 14.

Let Π be

$$\frac{\Gamma, \Delta \vdash \mathcal{P} : \tau}{\Gamma, \Delta, y : \mathtt{B} \vdash \mathcal{P} : \tau} \ (w)$$

Since $y \notin \mathrm{FV}(\mathcal{M})$, the proof follows by induction.

Let Π be

$$\frac{\Gamma, \Delta, y : \rho \vdash \mathcal{P} : \mathtt{A}}{\Gamma, \Delta \vdash \lambda y.\mathcal{P} : \rho \multimap \mathtt{A}} \ (\multimap I)$$

where $\lambda y.\mathcal{P} = \mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n$, so either $\mathcal{M} = \lambda y.\mathcal{P}'$ or $\mathcal{M} = x_k$, for some $k \in \{1, ..., n\}$.
Let $\mathcal{M} = \lambda y.\mathcal{P}'$. Since $(\lambda y.\mathcal{P}')[\mathcal{N}_i/x_i]_{i=1}^n = \lambda y.(\mathcal{P}'[\mathcal{N}_i/x_i]_{i=1}^n)$, by induction there are $\Delta_i \vdash \mathcal{N}_i : \sigma_i$ such that $\Gamma, y : \rho, x_1 : \sigma_1, ..., x_n : \sigma_n \vdash \mathcal{P} : \mathtt{A}$ $(1 \leq i \leq n)$: then the proof follows by applying rule $(\multimap I)$ to this derivation.
Otherwise, let $\mathcal{M} = x_k$. Since $\mathcal{N}_k = \lambda y.\mathcal{P}$, $\mathtt{A}_k = \rho \multimap \mathtt{A}$ and $\Gamma = \emptyset$, by one application of rule (Ax) $x_k : \rho \multimap \mathtt{A} \vdash x_k : \rho \multimap \mathtt{A}$: then $x_1 : \sigma_1, .., x_k : \rho \multimap \mathtt{A}, ...x_n : \sigma_n \vdash x_k : \rho \multimap \mathtt{A}$ follows by Property 14.

Let Π be

$$\frac{\Gamma', \Delta' \vdash \mathcal{P} : \rho \multimap \mathtt{A} \quad \Gamma'', \Delta'' \vdash \mathcal{Q} : \rho}{\Gamma, \Delta \vdash \mathcal{P}\mathcal{Q} : \mathtt{A}} \ (\multimap E)$$

where $\Gamma = \Gamma', \Gamma''$, so either $\mathcal{M} = \mathcal{P}'\mathcal{Q}'$ or $\mathcal{M} = x_k$, for some $k \in \{1, ..., n\}$.
Let $\mathcal{M} = \mathcal{P}'\mathcal{Q}'$, so $\mathcal{P}\mathcal{Q} = \mathcal{P}'[\mathcal{N}_i/x_i]_{i=1}^n \mathcal{Q}'[\mathcal{N}_i/x_i]_{i=1}^n$.
By inductive hypothesis, since $\mathcal{P} = \mathcal{P}'[\mathcal{N}_i/x_i]_{i=1}^n$ and $\mathcal{Q} = \mathcal{Q}'[\mathcal{N}_i/x_i]_{i=1}^n$, there are $\Delta_i' \vdash \mathcal{N}_i : \sigma_i'$ and $\Delta_i'' \vdash \mathcal{N}_i : \sigma_i''$ $(1 \leq i \leq n)$ such that $\Gamma', x_1 : \sigma_1', ..., x_n : \sigma_n' \vdash \mathcal{P}' : \rho \multimap \mathtt{A}$ and $\Gamma'', x_1 : \sigma_1'', ..., x_n : \sigma_n'' \vdash \mathcal{Q}' : \rho$.
From such derivations we obtain $\Sigma' \triangleright \Gamma', x_1' : \{\sigma_1'\}, ..., x_n' : \{\sigma_n'\} \vdash \mathcal{P}'[x_i'/x_i]_{i=1}^n : \rho \multimap \mathtt{A}$ and $\Sigma'' \triangleright \Gamma'', x_1'' : \{\sigma_1''\}, ..., x_n'' : \{\sigma_n''\} \vdash \mathcal{Q}'[x_i''/x_i]_{i=1}^n : \rho$ respectively, by a renaming sequence such that $\{x_1', ..., x_n'\} \cap \{x_1'', ..., x_n''\} = \emptyset$.

For every $1 \leq i \leq n$, let $\sigma_i = \{\{\sigma_i'\}\} \cup \{\{\sigma_i''\}\}$: if $\sigma_i' \neq \sigma_i''$, then we can build

$$\dfrac{\dfrac{\Delta_i' \vdash \mathcal{N}_i : \sigma_i'}{\{\Delta_i'\} \vdash \mathcal{N}_i : \{\sigma_i'\}}\ (st) \quad \dfrac{\Delta_i'' \vdash \mathcal{N}_i : \sigma_i''}{\{\Delta_i''\} \vdash \mathcal{N}_i : \{\sigma_i''\}}\ (st)}{\Delta_i \vdash \mathcal{N}_i : \sigma_i}\ (st)$$

where $\Delta_i = \{\{\Delta_i'\}\} \cup \{\{\Delta_i''\}\}$; otherwise, if $\sigma_i' = \sigma_i''$, then we build the following

$$\dfrac{\dfrac{\Delta_i' \vdash \mathcal{N}_i : \sigma_i'}{\{\Delta_i'\} \vdash \mathcal{N}_i : \{\sigma_i'\}}\ (st)}{\Delta_i \vdash \mathcal{N}_i : \sigma_i}\ (st)$$

where $\Delta_i = \{\{\Delta_i'\}\}$. Then there are $\Delta_i \vdash \mathcal{N}_i : \sigma_i$, for $1 \leq i \leq n$ and a renaming sequence δ such that the desired derivation is

$$\dfrac{\dfrac{\Sigma' \qquad \Sigma''}{\Gamma, x_1' : \{\sigma_1'\}, ..., x_n' : \{\sigma_n'\}, x_1'' : \{\sigma_1''\}, ..., x_n'' : \{\sigma_n''\} \vdash \mathcal{P}'[x_i'/x_i]_{i=1}^n \mathcal{Q}'[x_i''/x_i]_{i=1}^n : \mathtt{A}}\ (\multimap E)}{\Gamma, x_1 : \sigma_1, ..., x_n : \sigma_n \vdash \mathcal{P}'\mathcal{Q}' : \mathtt{A}}\ \delta$$

If $\mathcal{M} = x_k$, then the proof follows easily as in the previous case $(\multimap I)$.

Let Π be

$$\dfrac{\Gamma, \Delta, y_1 : \rho_1, ..., y_m : \rho_m : \tau_m \vdash \mathcal{P} : \tau}{\Gamma, \Delta, y : \cup_{j=1}^m \{\rho_j\} \vdash \mathcal{P}[y/y_j]_{j=1}^m : \tau}\ (m)$$

where $\mathcal{P}[y/y_j]_{j=1}^m = \mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n$.
Let $y \notin \cup_{i=1}^n \mathrm{FV}(\mathcal{N}_i)$, so $\mathcal{M} = \mathcal{M}'[y/y_j]_{j=1}^m$ and $\mathcal{P} = \mathcal{M}'[\mathcal{N}_i/x_i]_{i=1}^n$. If $y \notin \mathrm{FV}(\mathcal{M})$, then $\mathcal{P} = \mathcal{P}[y/y_j]_{j=1}^m = \mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n$ and the proof comes by induction. Otherwise, let $y \in \mathrm{FV}(\mathcal{M})$ and $\{y_{s_1}, ..., y_{s_p}\} = \{y_1, ..., y_m\} \cap \mathrm{FV}(\mathcal{M})$. By inductive hypothesis there are $\Delta_i \vdash \mathcal{N}_i : \sigma_i$ such that $\Gamma, y_{s_1} : \rho_{s_1}, ..., y_{s_p} : \rho_{s_p}, x_1 : \sigma_1, ..., x_n : \sigma_n \vdash \mathcal{M} : \tau$; then, from such derivation, we obtain the desired result by Property 14 and by one application of rule (m) with domain $\{y_1, ..., y_n\}$ and range $\{y\}$.
Now let us consider $y \in \cup_{i=1}^n \mathrm{FV}(\mathcal{N}_i)$, so $\mathcal{M} = \mathcal{M}'[y/y_j]_{j=1}^m[x_i/x_i^1, ..., x_i/x_i^{r_i}]_{i=1}^n$ and $\mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n = (\mathcal{M}'[\mathcal{N}_i^1/x_i^1, ..., \mathcal{N}_i^{n_i}/x_i^{r_i}]_{i=1}^n)[y/y_j]_{j=1}^m$.
Let $\{y_{s_1}, ..., y_{s_p}\} = \{y_1, ..., y_m\} \cap \mathrm{FV}(\mathcal{M}')$. Since $\mathcal{P} = \mathcal{M}'[\mathcal{N}_i^1/x_i^1, ..., \mathcal{N}_i^{r_i}/x_i^{r_i}]_{i=1}^n$, by inductive hypothesis there are $\Delta_i^h \vdash \mathcal{N}_i^h : \sigma_i^h$, for $1 \leq i \leq n$ and $1 \leq h \leq r_i$, such that

$$\Psi \triangleright \Gamma, y_{s_1} : \rho_{s_1}, ..., y_{s_p} : \rho_{s_p}, x_1^1 : \sigma_1^1, ..., x_1^{r_1} : \sigma_1^{r_1}, ..., x_n^1 : \sigma_n^1, ..., x_n^{r_n} : \sigma_n^{r_n} \vdash \mathcal{M}' : \tau$$

Note that $\mathcal{N}_i = \mathcal{N}_i^h[y/y_m]_{j=1}^m$, so we can build $\Delta_i'^h \vdash \mathcal{N}_i^h[y/y_m]_{j=1}^m : \sigma_i^h$ by applying rule (m) with domain $\mathrm{FV}(\mathcal{N}_i^h) \cap \{y_1, ..., y_m\}$ and range $\{y\}$ to $\Delta_i^h \vdash \mathcal{N}_i^h : \sigma_i^h$, for $1 \leq i \leq n$ and $1 \leq h \leq r_i$. Then, for every $1 \leq i \leq n$, if $\sigma_i = \cup_{h=1}^{r_i}\{\sigma_i^h\} = \{\sigma_i^{s_1^i}, ..., \sigma_i^{s_q^i}\}$ we can build

$$\dfrac{(\Delta_i'^h \vdash \mathcal{N}_i^h[y/y_m]_{j=1}^m : \sigma_i^h)_{s_1^i \leq h \leq s_q^i}}{\Delta_i \vdash \mathcal{N}_i : \sigma_i}\ (st)$$

Let δ be a renaming sequence containing n applications of rule (m), the i-th one having domain $\{x_i^1, ..., x_i^{r_i}\}$ and range $\{x_i\}$, for $1 \leq i \leq n$. If $y \in \mathrm{FV}(\mathcal{M})$, then from Ψ we derive

$$\Gamma, y : \cup_{j=1}^{m}\{\rho_j\}, x_1 : \sigma_1, ..., x_n : \sigma_n \vdash \mathcal{M}'[y/y_j]_{j=1}^{m}[x_i/x_i^1, ..., x_i/x_i^{r_i}]_{i=1}^{n} : \tau$$

by Property 14 and by one application of rule (m) with domain $\{y_1, ..., y_n\}$ and range $\{y\}$, followed by sequence δ. Otherwise, if $y \notin \mathrm{FV}(\mathcal{M})$, then $\mathcal{M}' = \mathcal{M}$ and $\{y_{s_1}, ..., y_{s_p}\} = \emptyset$, so we obtain the desired result by applying renaming sequence δ to $\Gamma, x_1^1 : \sigma_1^1, ..., x_1^{r_1} : \sigma_1^{r_1}, ..., x_n^1 : \sigma_n^1, ..., x_n^{r_n} : \sigma_n^{r_n} \vdash \mathcal{M} : \tau$.

Let Π be

$$\frac{(\Gamma_j, \Theta_j \vdash \mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^{n} : \tau_j)_{1 \leq j \leq m}}{\Gamma, \Delta \vdash \mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^{n} : \tau} \ (st)$$

where $\Gamma = \cup_{j=1}^{m}\{\Gamma_j\}$ and $\tau = \{\tau_1, ... , \tau_m\}$. For $1 \leq j \leq m$, by inductive hypothesis there are $\Delta_i^j \vdash \mathcal{N}_i : \sigma_i^j$, for $1 \leq i \leq n$, such that $\Gamma_j, x_1 : \sigma_1^j, ..., x_n : \sigma_n^j \vdash \mathcal{M} : \tau_j$. Then, for $1 \leq i \leq n$, if $\sigma_i = \cup_{j=1}^{m}\{\sigma_i^j\} = \{\sigma_i^{s_1^i}, ..., \sigma_i^{s_p^i}\}$ there are

$$\frac{(\Delta_i^j \vdash \mathcal{N}_i : \sigma_i^j)_{s_1^i \leq j \leq s_p^i}}{\Delta_i \vdash \mathcal{N}_i : \sigma_i} \ (st)$$

such that the desired derivation is

$$\frac{(\Gamma_j, x_1 : \sigma_1^j, ..., x_n : \sigma_n^j \vdash \mathcal{M} : \tau_j)_{1 \leq j \leq m}}{\Gamma, x_1 : \sigma_1, ..., x_n : \sigma_n \vdash \mathcal{M} : \tau} \ (st)$$

Finally, let Π end with an application of a quantifier rule: then the proof follows easily by induction.

□

Then we are able to prove a particular case of the (typed) subject expansion, where only the case of the subject being a redex is considered:

Lemma 30 (Subject expansion)**.** *$\Theta \vdash \mathcal{M}[\mathcal{N}/x] : \sigma$ and $\mathcal{N}$ typable in* STR *imply there exists Θ' such that $\Theta' \vdash (\lambda x.\mathcal{M})\mathcal{N} : \sigma$.*

Proof. By induction on σ.

Let $\sigma = \mathtt{A}$. Consider $\Theta = \Gamma, \Xi$ where $\mathrm{dom}(\Gamma) = \mathrm{FV}(\mathcal{M}) \setminus \{x\}$. By Lemma 29, there is $\Delta \vdash \mathcal{N} : \tau$ such that $\Gamma, x : \tau \vdash \mathcal{M} : \mathtt{A}$; let $\Delta' \vdash \mathcal{N}' : \tau$ be a copy of $\Delta \vdash \mathcal{N} : \tau$, such that $\Gamma \# \Delta'$: then we can build the following derivation

$$\frac{\dfrac{\Gamma, x : \tau \vdash \mathcal{M} : \mathtt{A}}{\Gamma \vdash \lambda x.\mathcal{M} : \tau \multimap \mathtt{A}} \ (\multimap I) \qquad \Delta' \vdash \mathcal{N}' : \tau}{\Gamma, \Delta' \vdash (\lambda x.\mathcal{M})\mathcal{N}' : \mathtt{A}} \ (\multimap E)$$

and, since $(\lambda x.\mathcal{M})\mathcal{N}$ is an instance of $(\lambda x.\mathcal{M})\mathcal{N}'$, the result follows by Lemma 22.
Otherwise, let $\sigma = \{\sigma_1, \dots, \sigma_n\}$. By Property 16, $\Pi \rhd \Gamma \vdash \mathcal{M}[\mathcal{N}/x] : \{\sigma_1, \dots, \sigma_n\}$ implies there are derivations $\Pi_i \rhd \Gamma_i \vdash \mathcal{P} : \sigma_i$ $(1 \leq i \leq n)$ such that Π is obtained by an application of rule (st) to $(\Pi_i)_{i=1}^n$, followed by a renaming sequence δ. By applying sequence δ to Π_i we obtain $\Gamma_i'' \vdash \mathcal{M}[\mathcal{N}/x] : \sigma_i$, for $1 \leq i \leq n$.
By inductive hypothesis there are Γ_i' such that $\Gamma_i' \vdash (\lambda x.\mathcal{M})\mathcal{N} : \sigma_i$, for $1 \leq i \leq n$: then by applying rule (st) to such derivations we obtain $\Gamma' \vdash (\lambda x.\mathcal{M})\mathcal{N} : \sigma$, where $\Gamma' = \cup_{i=1}^n \{\Gamma_i'\}$.

□

Finally we can prove the desired implication, namely that all strongly normalizing terms are typable in STR:

Theorem 17. *If $\mathcal{M}$ is strongly normalizing, then $\mathcal{M}$ is typable in* STR.

Proof. For each of the three rules of Table 3.3 we show that, if the premises of the rule are typable, then the conclusion is typable as well.

Let us consider Rule (3.1), so $\mathcal{M} = x\mathcal{N}_1 \dots \mathcal{N}_n$. Let $x', \mathcal{N}_1', \dots, \mathcal{N}_n'$ be instances of $x, \mathcal{N}_1, \dots, \mathcal{N}_n$ respectively, such that $j \neq h$ implies $\mathrm{FV}(\mathcal{N}_j') \cap \mathrm{FV}(\mathcal{N}_h') = \emptyset$, for $1 \leq j, h \leq n$, and $x' \notin \cup_{i=1}^n \mathrm{FV}(\mathcal{N}_i')$: we prove that there is a derivation, assigning to $x'\mathcal{N}_1' \dots \mathcal{N}_n'$ a linear type, from which we can derive a typing for $x\mathcal{N}_1 \dots \mathcal{N}_n$.
If $n = 0$, then the proof is obvious.
Otherwise, by inductive hypothesis on $\mathcal{N}_1', \dots, \mathcal{N}_n'$ and by Corollary 12 there are $\Gamma_i' \vdash \mathcal{N}_i' : \mathtt{A}_i$, for $1 \leq i \leq n$. Let $\mathtt{A} = \mathtt{A}_1 \multimap \mathtt{A}_2 \multimap \dots \multimap \mathtt{A}_n \multimap \mathtt{B}$; then we can build

$$\dfrac{\dfrac{\dfrac{}{x' : \mathtt{A} \vdash x' : \mathtt{A}}\,(Ax) \quad \Gamma_1' \vdash \mathcal{N}_1' : \mathtt{A}_1}{x' : \mathtt{A}, \Gamma_1' \vdash x'\mathcal{N}_1' : \mathtt{A}_2 \multimap \dots \multimap \mathtt{A}_n \multimap \mathtt{B}}\,(\multimap E) \quad \Gamma_2' \vdash \mathcal{N}_2' : \mathtt{A}_2}{\vdots}\,(\multimap E)}{x' : \mathtt{A}, \Gamma_1', \dots, \Gamma_n' \vdash x'\mathcal{N}_1' \dots \mathcal{N}_n' : \mathtt{B}}\,(\multimap E)$$

and, since $x\mathcal{N}_1 \dots \mathcal{N}_n$ is an instance of $x'\mathcal{N}_1' \dots \mathcal{N}_n'$, the desired derivation follows by Lemma 22.

Now let us consider Rule (3.2), so $\mathcal{M} = \lambda x.\mathcal{N}$. Since $\mathcal{N} \in \mathrm{SN}$, by inductive hypothesis on $\mathcal{N}$ there is a derivation $\Gamma \vdash \mathcal{N} : \mathtt{B}$. If $x \in \mathrm{FV}(\mathcal{N})$, then $\Gamma = \Gamma', x : \tau$, so the desired derivation follows by one application of rule $(\multimap I)$ to Π'; otherwise, we first apply a rule (w) with range $\{x\}$, followed by an application of rule $(\multimap I)$ to abstract over x.

Finally, let us consider the crucial case of Rule (3.3), so $\mathcal{M} = (\lambda x.\mathcal{M})\mathcal{N}\mathcal{M}_1 \dots \mathcal{M}_n$. By induction, both $\mathcal{M}[\mathcal{N}/x]\mathcal{M}_1 \dots \mathcal{M}_n$ and $\mathcal{N}$ are typable. Moreover, by Lemma 23

the derivation proving $\Gamma \vdash \mathcal{M}[\mathcal{N}/x]\mathcal{M}_1...\mathcal{M}_n : \mathtt{A}$ is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Theta_1 \vdash \mathcal{P}_1 : \sigma_1^1 \multimap ... \multimap \sigma_n^1 \multimap \mathtt{B}_1 \quad \Delta_1 \vdash \mathcal{M}_1^1 : \sigma_1^1}{\Theta_1, \Delta_1 \vdash \mathcal{P}_1\mathcal{M}_1^1 : \sigma_2^1 \multimap ... \multimap \sigma_n^1 \multimap \mathtt{B}_1}\,(\multimap E)}{\Theta_2 \vdash \mathcal{P}_2\mathcal{M}_1^2 : \sigma_2^2 \multimap ... \multimap \sigma_n^2 \multimap \mathtt{B}_2}\,\delta_1 \qquad \Delta_2 \vdash \mathcal{M}_2^2 : \sigma_2^2}{\Theta_2, \Delta_2 \vdash \mathcal{P}_2\mathcal{M}_1^2\mathcal{M}_2^2 : \sigma_3^2 \multimap ... \multimap \sigma_n^2 \multimap \mathtt{B}_2}\,(\multimap E)}{\Theta_3 \vdash \mathcal{P}_3\mathcal{M}_1^3\mathcal{M}_2^3 : \sigma_3^3 \multimap ... \multimap \sigma_n^3 \multimap \mathtt{B}_3}\,\delta_2}{\vdots}}{\Theta_n, \Delta_n \vdash \mathcal{P}_n\mathcal{M}_1^n...\mathcal{M}_n^n : \mathtt{B}_n}\,\delta_n}{\Gamma \vdash \mathcal{M}[\mathcal{N}/x]\mathcal{M}_1...\mathcal{M}_n : \mathtt{A}}$$

where $\mathcal{M}[\mathcal{N}/x]$ is an instance of $\mathcal{P}_i$, $\mathcal{M}_i$ is an instance of $\mathcal{M}_i^j$ and δ_i is a renaming and quantifier sequence, for $1 \leq i \leq n$ and $i \leq j \leq n$.
Without loss of generality, we can safely assume that $\mathrm{FV}(\mathcal{M}) \cap \cup_{i=1}^n \mathrm{FV}(\mathcal{M}_i^i) = \emptyset$ and $\mathrm{FV}(\mathcal{N}) \cap \cup_{i=1}^n \mathrm{FV}(\mathcal{M}_i^i) = \emptyset$. By Lemma 26, each sequence δ_i $(1 \leq i \leq n)$ can be rearranged in such a way that the applications of quantifier rules precede the applications of renaming rules; let $\delta_1'...\delta_n'$ be such quantifier sequences.
Since $\mathcal{M}[\mathcal{N}/x]$ is an instance of $\mathcal{P}_1$, by Lemma 22 $\Theta \vdash \mathcal{M}[\mathcal{N}/x] : \sigma_1^1 \multimap ... \multimap \sigma_n^1 \multimap \mathtt{B}_1$ can be obtained from $\Theta_1 \vdash \mathcal{P}_1 : \sigma_1^1 \multimap ... \multimap \sigma_n^1 \multimap \mathtt{B}_1$ by a renaming sequence. Moreover, since by hypothesis $\mathcal{N}$ is typable in STR, by Lemma 30 there is Θ' such that $\Theta' \vdash (\lambda x.\mathcal{M})\mathcal{N} : \sigma_1^1 \multimap ... \multimap \sigma_n^1 \multimap \mathtt{B}_1$: then we can build

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Theta' \vdash (\lambda x.\mathcal{M})\mathcal{N} : \sigma_1^1 \multimap ... \multimap \sigma_n^1 \multimap \mathtt{B}_1 \quad \Delta_1 \vdash \mathcal{M}_1^1 : \sigma_1^1}{\Theta', \Delta_1 \vdash (\lambda x.\mathcal{M})\mathcal{N}\mathcal{M}_1^1 : \sigma_2^1 \multimap ... \multimap \sigma_n^1 \multimap \mathtt{B}_1}\,(\multimap E)}{\Theta', \Delta_1 \vdash (\lambda x.\mathcal{M})\mathcal{N}\mathcal{M}_1^1 : \sigma_2^2 \multimap ... \multimap \sigma_n^2 \multimap \mathtt{B}_2}\,\delta_1' \qquad \Delta_2 \vdash \mathcal{M}_2^2 : \sigma_2^2}{\Theta', \Delta_1, \Delta_2 \vdash (\lambda x.\mathcal{M})\mathcal{N}\mathcal{M}_1^1\mathcal{M}_2^2 : \sigma_3^2 \multimap ... \multimap \sigma_n^2 \multimap \mathtt{B}_2}\,(\multimap E)}{\Theta', \Delta_1, \Delta_2 \vdash (\lambda x.\mathcal{M})\mathcal{N}\mathcal{M}_1^1\mathcal{M}_2^2 : \sigma_3^3 \multimap ... \multimap \sigma_n^3 \multimap \mathtt{B}_3}\,\delta_2'}{\vdots}}{\Theta', \Delta_1, ..., \Delta_n \vdash (\lambda x.\mathcal{M})\mathcal{N}\mathcal{M}_1^1...\mathcal{M}_n^n : \mathtt{B}_n}\,\delta_n'}{\Theta', \Delta_1, ..., \Delta_n \vdash (\lambda x.\mathcal{M})\mathcal{N}\mathcal{M}_1^1...\mathcal{M}_n^n : \mathtt{A}}$$

Since $(\lambda x.\mathcal{M})\mathcal{N}\mathcal{M}_1...\mathcal{M}_n$ is an instance of $(\lambda x.\mathcal{M})\mathcal{N}\mathcal{M}_1^1...\mathcal{M}_n^n$, the desired derivation follows by applying Lemma 22.

□

3.4 Polynomial characterization

From the point of view of implicit complexity, we are interested in proving that STR characterizes exactly the functions of FPTIME; therefore, while being considerably more expressive, STR characterizes exactly the same functions as the Soft Type Assignment system STA [GR07], which was proved to be sound and complete with

respect to FPTIME. This is somehow in line with the result obtained in [BPS03], where a typing system assigning simple types to λ-terms is proved to represent exactly the same class of functions as one endowed with intersection types, by giving a translation of terms typed in the former system into terms typed in the latter one.

3.4.1 Translation from STA to STR

In order to prove the FPTIME characterization, we consider the relation between STA and STR. We start by showing that if a term is typable in STA, then it is also typed by STR. Such translation is easy to define on both types and bases, since STA can be seen as a restriction of STR, where only variables with the same type can be contracted by rule (m), while rule (!) is the equivalent of rule (st) where only one premise is allowed and stratification is replaced by the ! marker:

Definition 23 (Translation from STA to STR)**.**

- *The translation* $(.)^\circ$ *from* $\mathcal{TS}$ *to* $\mathcal{T}$ *is defined inductively as follows::*

$$(\mathtt{a})^\circ = \mathtt{a}; \quad (\mu \multimap \mathtt{U})^\circ = (\mu)^\circ \multimap (\mathtt{U})^\circ; \quad (!\mu)^\circ = \{(\mu)^\circ\}$$

- *Let* Θ *be a basis in* STA*: then* $(\Theta)^\circ$ *is the basis such that* $(\Theta)^\circ(x) = (\Theta(x))^\circ$.

By the previous definition, the translation is obtained in a straightforward way:

Lemma 31. $\Theta \vdash_{\mathtt{STA}} \mathcal{M} : \mu$ *implies* $(\Theta)^\circ \vdash \mathcal{M} : (\mu)^\circ$.

Proof. Let $\Phi \triangleright \Theta \vdash_{\mathtt{STA}} \mathcal{M} : \mu$. We proceed by induction on Φ.

Let Φ end with an application of rule (Ax): then the proof is trivial.

Let Φ end with an application of rule (sp) to Π'. By inductive hypothesis $(\Pi')^\circ \triangleright (\Theta)^\circ \vdash_{\mathtt{STA}} \mathcal{M} : (\nu)^\circ$: then $(\Pi)^\circ$ is obtained by applying rule (st) to the unique premise $(\Pi')^\circ$.

All the other cases follow easily by induction and by applying the respective rule in STR.

□

Observe that the converse translation from STR to STA is not at all straightforward, since STR is much more expressive than STA, in which only a strict subset of the strongly normalizing terms can be typed. In particular, the technique of [BPS03] can hardly be adapted to our setting, since both typing systems employ the renaming of variables in order to control the duplication of terms.
The reason of this difficulty lays in the fact that a translation from $\mathcal{T}$ to $\mathcal{TS}$ cannot be designed simply by induction on the type, but it should depend on the whole derivation, since in principle only a *partial linearization* of a term typed in STR could be typed in STA. Luckily, there is an easier way to prove the soundness of STR with respect to FPTIME, by using the result of Theorem 15.

3.4.2 Polynomial time soundness and completeness

As in Section 2.1.5, we need to define an encoding of data and its respective datatype, in order to represent functions. We choose the Church representation of binary words:

$$w \in \{0,1\}^\star, \quad w = \langle i_1, \ldots, i_n \rangle, \qquad \underline{w} \;=\; \lambda f_0.\lambda f_1.\lambda x.f_{i_1}\ (f_{i_2}\ \ldots (f_{i_n}\ x)\ldots)$$

where, by abuse of notation, we also denote by $\underline{1}$ the identity $\lambda f.f$. The size of the binary word w is denoted by $|w|$; it is easy to check that $|\underline{w}|$ is linear with respect to $|w|$.

Church binary words can be assigned in $\mathtt{STA}$ both a uniform and a parametric type, defined respectively as $\mathrm{W} = \forall \mathtt{a}.!(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a} \multimap \mathtt{a}$ and $\mathbf{W}_{n,m} = \forall \mathtt{a}.!^n(\mathtt{a} \multimap \mathtt{a}) \multimap !^m(\mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a} \multimap \mathtt{a}$, for every $n \geq 1$ and $m \geq 1$; note that $\mathrm{W} = \mathbf{W}_{1,1}$.
It is easy to check that every derivation $\Pi \triangleright \vdash_{\mathtt{STA}} \underline{w} : \mathbf{W}_{n,m}$, for $n \geq 1$ and $m \geq 1$, is such that $\mathtt{d_S}(\Pi) = 0$: indeed, the polynomiality of $\mathtt{STA}$ depends on this very property.

The parametric datatypes play an essential role in bounding the running time of the computation of a program in $\mathtt{STA}$, since a term representing a numerical function can have different parameters for its input(s) and output types: because of the mismatch between such types, the iteration of such functions is thus forbidden, with the result that terms representing non-polynomial functions (such as exponentiation) cannot be typed.

In a very similar way, Church binary words can be assigned in $\mathtt{STR}$ both the uniform type $\mathbf{WI} = \forall \mathtt{a}.\{\mathtt{a} \multimap \mathtt{a}\} \multimap \{\mathtt{a} \multimap \mathtt{a}\} \multimap \mathtt{a} \multimap \mathtt{a}$ and a parametric type $\mathbf{WI}_{n,m} = \forall \mathtt{a}.\{\mathtt{a} \multimap \mathtt{a}\}^n \multimap \{\mathtt{a} \multimap \mathtt{a}\}^m \multimap \mathtt{a} \multimap \mathtt{a}$, for any $n \geq 1$ and $m \geq 1$. Again, it is easy to check that the depth of any derivation $\Pi \triangleright \vdash_{\mathtt{STA}} \underline{w} : \mathbf{WI}_{n,m}$ is equal to 0, for $n \geq 1$ and $m \geq 1$.

Now we are able to formally define the representation of functions in both $\mathtt{STA}$ and $\mathtt{STR}$. Such definition is a straightforward extension of the classical definition of λ-representation of functions [Bar84; RP04] in a typed setting [BPS03; GR07].
The additional power of $\mathtt{STR}$ with respect to $\mathtt{STA}$ is illustrated by the fact that, while in $\mathtt{STA}$ every input data must be assigned a parametric type for natural numbers, in $\mathtt{STR}$ we allow each input number to be assigned a stratified set of types, with the proviso that all its linear components are numerical types:

Definition 24 (Representation of functions)**.** *Let $\phi : \mathbb{N}^p \longrightarrow \mathbb{N}$ be a function of arity p.*

i. A program $\mathcal{M} = \lambda x_1 ... x_p.\mathcal{P}$ represents ϕ in $\mathtt{STA}$ if and only if, for any binary word $w_1, ..., w_p$:

- $\mathcal{M}\underline{w_1}...\underline{w_p} \xrightarrow{*} \underline{\phi(w_1, ..., w_p)}$;

- $\vdash_{\mathtt{STA}} \mathcal{M} : !^{i_1}\mathbf{W}_{j_1,k_1} \multimap ... \multimap !^{i_p}\mathbf{W}_{j_p,k_p} \multimap \mathbf{W}_{j,k}$, *for some* j, k, j_h, k_h $(1 \leq h \leq p)$.

ii. A program $\mathcal{M} = \lambda x_1...x_p.\mathcal{P}$ *represents* ϕ *in* STR *if and only if:*

- $\mathcal{M}\underline{w_1}...\underline{w_p} \xrightarrow{*} \underline{\phi(w_1, ..., w_p)}$;
- $\vdash \mathcal{M} : \sigma_1 \multimap ... \multimap \sigma_p \multimap \mathbf{WI}_{h,k}$ *and* $\overline{\sigma_i} = [\mathbf{WI}_{h_1^i,k_1^i}, ..., \mathbf{WI}_{h_{q_i}^i,k_{q_i}^i}]$, *for some* $h, k, h_i, k_i, q_i, h_r^i, k_r^i$ $(1 \leq i \leq p, 1 \leq r \leq q_i)$.

In [GR07], the authors prove that STA is sound and complete with respect to FPTIME; here we exploit the translation from STA to STR, given in Lemma 31, in order to extend the completeness result to STR:

Lemma 32 (FPTIME completeness)**.** *Let* $\phi : \mathbb{N}^p \longrightarrow \mathbb{N}$ *be a polynomial time function, for* $p \geq 1$*: then there is a typed term of* STR *representing it.*

Proof. By Definition 24.i, let $\mathcal{M}$ be a term representing function ϕ in STA, such that $\mathcal{M}\underline{w_1}...\underline{w_p} \xrightarrow{*} \underline{\phi(w_1, ..., w_p)}$ and $\vdash_{\mathtt{STA}} \mathcal{M} : !^{i_1}\mathbf{W}_{j_1,k_1} \multimap ... \multimap !^{i_p}\mathbf{W}_{j_p,k_p} \multimap \mathbf{W}_{j,k}$, for any binary word $w_1, ..., w_p$ and some j, k, j_h, k_h $(1 \leq h \leq p)$.
By Definition 23, we know that $(!^{i_s}\mathbf{W}_{j_s,k_s})^\circ = \{\mathbf{WI}_{j_s,k_s}\}^{i_s}$, for $1 \leq s \leq p$, and $(\mathbf{W}_{j,k})^\circ = \mathbf{WI}_{j,k}$. By Lemma 31, the above STA derivation typing $\mathcal{M}$ is translated into $\vdash \mathcal{M} : \{\mathbf{WI}_{j_1,k_1}\}^{i_1} \multimap ... \multimap \{\mathbf{WI}_{j_p,k_p}\}^{i_p} \multimap \mathbf{WI}_{j,k}$: then $\mathcal{M}$ represents ϕ in STR by Definition 24.ii.

□

Let $\mathcal{M}$ be a term representing a numerical function $\phi : \mathbb{N}^p \longrightarrow \mathbb{N}$ in STR; in order to prove the soundness of STR with respect to FPTIME, we show that the reduction of $\mathcal{M}\underline{w_1}...\underline{w_p}$ to its normal form can be performed on a Turing machine of time polynomial in the size of the input.
Consider the application of a program to a given argument. Our aim is to bound the number of duplications (namely, substitutions) performed by a reduction step; therefore, we need to know how many application of rule (Ax) play a role in creating an occurrence of the formal parameter of the considered program. Since each occurrence of a variable is born linear, we have to keep track of the renaming history of each variable, namely of its set of *ancestors*:

Definition 25 (Ancestors of a variable)**.** *Let* $\Pi \triangleright \Gamma \vdash \mathcal{M} : \tau$ *and* $x \in \mathrm{dom}(\Gamma)$*; the set of ancestors of* x *in* Π*, denoted by* $A(x, \Pi)$*, is defined inductively as follows:*

- *if* Π *is*

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

then $A(x, \Pi) = \{x\}$;

- *if* Π *is*
$$\frac{\Pi' \triangleright \Gamma \vdash \mathcal{M} : \tau}{\Gamma, y : \mathtt{A} \vdash \mathcal{M} : \tau}\ (w)$$
then $A(x, \Pi) = \{x\}$ *if* $y = x$, $A(x, \Pi) = A(x, \Pi')$ *otherwise;*

- *if* Π *is*
$$\frac{\Pi' \triangleright \Gamma, y : \sigma \vdash \mathcal{P} : \mathtt{A}}{\Gamma \vdash \lambda y.\mathcal{P} : \sigma \multimap \mathtt{A}}\ (\multimap I)$$
then $A(x, \Pi) = A(x, \Pi')$;

- *if* Π *is*
$$\frac{\Pi' \triangleright \Gamma' \vdash \mathcal{P} : \sigma \multimap \mathtt{A} \quad \Pi'' \triangleright \Gamma'' \vdash \mathcal{Q} : \sigma}{\Gamma', \Gamma'' \vdash \mathcal{P}\mathcal{Q} : \mathtt{A}}\ (\multimap E)$$
then $A(x, \Pi) = A(x, \Pi')$ *if* $x \in \mathrm{dom}(\Gamma')$, $A(x, \Pi) = A(x, \Pi'')$ *otherwise;*

- *if* Π *is*
$$\frac{\Pi' \triangleright \Gamma, y_1 : \sigma_1, ..., y_n : \sigma_n \vdash \mathcal{M} : \tau}{\Gamma, y : \cup_{i=1}^{n}\{\sigma_i\} \vdash \mathcal{M}[y/y_i]_{i=1}^{n} : \tau}\ (m)$$
then $A(x, \Pi) = \cup_{i=1}^{n} A(y_i, \Pi')$ *if* $x = y$, $A(x, \Pi) = A(x, \Pi')$ *otherwise;*

- *if* Π *is*
$$\frac{(\Pi_i \triangleright \Gamma_i \vdash \mathcal{M} : \sigma_i)_{1 \leq i \leq n}}{\cup_{i=1}^{n}\{\Gamma_i\} \vdash \mathcal{M} : \{\sigma_1, \dots, \sigma_n\}}\ (st)$$
then $A(x, \Pi) = \cup_{i=1}^{n} A(x, \Pi_i)$;

- *if* Π *is*
$$\frac{\Pi' \triangleright \Gamma \vdash \mathcal{M}' : \sigma'}{\Gamma \vdash \mathcal{M} : \sigma}\ (R)$$
where (R) *is a quantifier rule, then* $A(x, \Pi) = A(x, \Pi')$.

We can now prove the FPTIME soundness result for STR as well:

Theorem 18 (FPTIME soundness)**.** *Let* $\phi : \mathbb{N}^p \longrightarrow \mathbb{N}$, *with* $p \geq 1$, *and let* $\mathcal{M}$ *be a program representing* ϕ *in* STR*; then there is a polynomial* P *such that, for any binary word* $w_1, ..., w_p$, $\mathcal{M}\underline{w_1}...\underline{w_p}$ *can be evaluated to its normal form on a Turing Machine in time* $O(P(|w_1| + ... + |w_p|))$.

Proof. By Definition 24.(ii), if $\mathcal{M}$ represents ϕ then $\mathcal{M}\underline{w_1}...\underline{w_p} \xrightarrow{*} \underline{\phi(w_1, ..., w_p)}$ and $\vdash \mathcal{M} : \sigma_1 \multimap ... \multimap \sigma_p \multimap \mathbf{WI}_{h,k}$, where $\overline{\sigma}_i = [\mathbf{WI}_{h_1^i, k_1^i}, ..., \mathbf{WI}_{h_{q_i}^i, k_{q_i}^i}]$ for some h, k, h_i, k_i, q_i, h_r^i, k_r^i $(1 \leq i \leq p, 1 \leq r \leq q_i)$.
Since $\mathcal{M}$ is a program, namely a closed normal form, by Lemma 23 we can safely

assume that $\mathcal{M} = \lambda x_1 ... x_p.\mathcal{P}$ and $\mathcal{M} : \sigma_1 \multimap ... \multimap \sigma_p \multimap \mathbf{WI}_{h,k}$ ends with p applications of rule $(\multimap I)$, whose initial premise is $\Phi \triangleright x_1 : \sigma_1, ..., x_p : \sigma_p \vdash \mathcal{P} : \mathbf{WI}_{h,k}$. Then we can build the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Phi \triangleright x_1 : \sigma_1, ..., x_p : \sigma_p \vdash \mathcal{P} : \mathbf{WI}_{h,k}}{\vdash \mathcal{M} : \sigma_1 \multimap ... \multimap \sigma_p \multimap \mathbf{WI}_{h,k}}\,(\multimap I) \quad \Phi_1 \vdash \underline{w_1} : \sigma_1}{\vdots}\,(\multimap E)}{\vdash \mathcal{M}\underline{w_1}...\underline{w_{p-1}} : \sigma_p \multimap \mathbf{WI}_{h,k} \qquad \Phi_p \vdash \underline{w_p} : \sigma_p}\,(\multimap E)}{\vdash \mathcal{M}\underline{w_1}...\underline{w_p} : \mathbf{WI}_{h,k}}\,(\multimap E)$$

where each Φ_i $(1 \leq i \leq p)$ is obtained from derivations $\vdash \underline{w_i} : \mathbf{WI}_{h_t^i,k_t^i}$ $(1 \leq t \leq q_i)$, each of depth 0, by a suitable sequence of applications of rule (st).
By Lemma 25, there is a derivation $\Pi\triangleright \vdash \mathcal{P}[\underline{w_i}/x_i]_{i=1}^p : \mathbf{WI}_{h,k}$. Moreover, by observing the proof of the substitution Lemma, it is easy to see that such derivation is obtained by replacing axiom $y : \mathbf{WI}_{h_t^i,k_t^i} \vdash y : \mathbf{WI}_{h_t^i,k_t^i}$ of Φ, for each $y \in A(x_i, \Phi)$, with derivation $\vdash \underline{w_i} : \mathbf{WI}_{h_t^i,k_t^i}$ of depth 0, for $1 \leq i \leq p$ and $1 \leq t \leq q_i$; therefore $\mathtt{d}(\Pi) = \mathtt{d}(\Phi)$, so $\mathtt{d}(\Pi)$ does not depend on the size of the input.
Let $\mathcal{M}\underline{w_1}...\underline{w_p} \xrightarrow{*} \mathcal{P}[\underline{w_i}/x_i]_{i=1}^p \xrightarrow{*} \underline{\phi(w_1, ..., w_p)}$ in m β-reduction steps; by Theorem 15, $m \leq |\mathcal{M}\underline{w_1}...\underline{w_p}|^{(\mathtt{d}(\Pi)+1)}$ and each intermediate term $\mathcal{N}$ in the reduction sequence is such that $|\mathcal{N}| \leq |\mathcal{M}\underline{w_1}...\underline{w_p}|^{(\mathtt{d}(\Pi)+1)}$. Since a β-reduction step $\mathcal{N} \to \mathcal{N}'$ can be simulated in time $O(|\mathcal{N}|^2)$ on a Turing machine [Ter01], each reduction step takes a time $O(|\mathcal{M}\underline{w_1}...\underline{w_p}|^{2(\mathtt{d}(\Pi)+1)}$: then, since the size of the program is a constant with respect to the computation, the conclusion follows. □

Since STR is both sound and complete with respect to FPTIME, the desired characterization result holds:

Corollary 19 (FPTIME characterization)**.** STR *characterizes* FPTIME.

Proof. Easy by combining Lemma 32 and Theorem 18. □

Note that, since STA is also sound and complete with respect to FPTIME, the functions representable in STR are exactly the ones representable in STA. However, more programs can be typed in STR, as a consequence of the introduction of stratified (intersection) types into the equation.

3.4.3 Examples of gain in expressivity

Consider the following *successors*, concatenating a binary word with either 0 or 1:

- $\underline{\mathtt{succ}_0} = \lambda w.\lambda f_0.\lambda f_1.\lambda x.w f_0 f_1(f_0 x)$ has type $\mathbf{WI}_{m,n} \multimap \mathbf{WI}_{m+1,n}$ in STR (resp. $\mathbf{W}_{m,n} \multimap \mathbf{W}_{m+1,n}$ in STA) and corresponds to the function $f(x) = 2x$;

- $\underline{\underline{\mathtt{succ}_1}} = \lambda w.\lambda f_0.\lambda f_1.\lambda x.wf_0f_1(f_1x)$ has type $\mathbf{WI}_{m,n} \multimap \mathbf{WI}_{m,n+1}$ in $\mathtt{STR}$ (resp. $\mathbf{W}_{m,n} \multimap \mathbf{W}_{m,n+1}$ in $\mathtt{STA}$) and corresponds to the function $f(x) = 2x + 1$.

Let $\mathtt{A} = \mathtt{a} \multimap \mathtt{a}$; the derivation for $\underline{\mathtt{succ}_0}$ is the following:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Pi \triangleright w : \mathbf{WI}_{m,n}, f_0' : \{\mathtt{A}\}^m, f_1 : \{\mathtt{A}\}^n \vdash wf_0'f_1 : \mathtt{A}}{w : \mathbf{WI}_{m,n}, f_0' : \{\mathtt{A}\}^m, f_0'' : \{\mathtt{A}\}^m, f_1 : \{\mathtt{A}\}^n, x : \mathtt{a} \vdash wf_0'f_1(f_0''x) : \mathtt{a}}\,(m)}{w : \mathbf{WI}_{m,n}, f_0 : \{\mathtt{A}\}^{m+1}, f_1 : \{\mathtt{A}\}^n, x : \mathtt{a} \vdash wf_0f_1(f_0x) : \mathtt{a}}\,(m)}{w : \mathbf{WI}_{m,n} \vdash \lambda f_0.\lambda f_1.\lambda x.wf_0f_1(f_0x) : \{\mathtt{A}\}^{m+1} \multimap \{\mathtt{A}\}^n \multimap \mathtt{A}}\,(\forall I)}{\vdash \underline{\mathtt{succ}_0} : \mathbf{WI}_{m,n} \multimap \mathbf{WI}_{m+1,n}}\,(\multimap I)}$$

where Π is

$$\dfrac{\dfrac{\dfrac{\dfrac{}{w : \mathbf{WI}_{m,n} \vdash w : \mathbf{WI}_{m,n}}\,(Ax)}{w : \mathbf{WI}_{m,n} \vdash w : \{\mathtt{A}\}^m \multimap \{\mathtt{A}\}^n \multimap \mathtt{A}}\,(\forall E) \quad \dfrac{}{f_0' : \mathtt{A} \vdash f_0' : \mathtt{A}}\,(Ax)}{f_0' : \{\mathtt{A}\}^m \vdash f_0' : \{\mathtt{A}\}^m}\,(st)}{w : \mathbf{WI}_{m,n}, f_0' : \{\mathtt{A}\}^m \vdash wf_0' : \{\mathtt{A}\}^n \multimap \mathtt{A}}\,(\multimap E) \quad \dfrac{\dfrac{}{f_1 : \mathtt{A} \vdash f_1 : \mathtt{A}}\,(Ax)}{f_1 : \{\mathtt{A}\}^n \vdash f_1 : \{\mathtt{A}\}^n}\,(st)}{w : \mathbf{WI}_{m,n}, f_0' : \{\mathtt{A}\}^m, f_1 : \{\mathtt{A}\}^n \vdash wf_0'f_1 : \mathtt{A}}\,(\multimap E)$$

The typing of $\underline{\underline{\mathtt{succ}_1}}$ is similar.

As an example of the gain in expressivity of $\mathtt{STR}$ with respect to $\mathtt{STA}$, let us first consider the *iteration* of the successor function over binary words; such programming construct is not typable in $\mathtt{STA}$ with a meaningful type, because the polynomial bound is enforced by the fact that functions cannot be iterated, but only composed, in order to forbid the construction of exponential functions. Nonetheless, in $\mathtt{STR}$ it is possible to type a limited notion of iteration, as shown in the following example.

Example 13 (Iteration of the successor)**.** *In* $\mathtt{STR}$ *it is possible to iterate the successor of a binary number a constant number* k *of times.*
Consider the k*-loop term* $\underline{\mathtt{ITER}_k} = \lambda f.\lambda x.f^k x$*; in order to be applied to the program* $\underline{\mathtt{succ}_0}$*, the* $\underline{\mathtt{ITER}_k}$ *term can by typed in the following way:*

$$\dfrac{\dfrac{\dfrac{\vdots}{f_1 : \mathtt{V}_1, ..., f_k : \mathtt{V}_k, x : \mathbf{WI}_{m,n} \vdash f_k(...(f_1x)...) : \mathbf{WI}_{m+k,n}}\,(m)}{f : \{\mathtt{V}_1, \dots, \mathtt{V}_k\}, x : \mathbf{WI}_{m,n} \vdash f^k x : \mathbf{WI}_{m+k,n}}\,(m)}{\vdash \lambda f.\lambda x.f^k x : \{\mathtt{V}_1, \dots, \mathtt{V}_k\} \multimap \mathbf{WI}_{m,n} \multimap \mathbf{WI}_{m+k,n}}\,(\multimap I)$$

where $\mathtt{V}_i = \mathbf{WI}_{m+i-1,n} \multimap \mathbf{WI}_{m+i,n}$*, for* $1 \leq i \leq k$*.*
The term $\underline{\mathtt{ITER}_k}\ \underline{\mathtt{succ}_0}$ *can then be obtained by applying rule* $(\multimap E)$ *to the derivation above and to the one obtained as follows:*

$$\dfrac{(\vdash \underline{\mathtt{succ}_0} : \mathtt{V}_i)_{1 \leq i \leq k}}{\vdash \underline{\mathtt{succ}_0} : \{\mathtt{V}_1, \dots, \mathtt{V}_k\}}\,(st)$$

Similarly, in order to apply $\underline{\texttt{ITER}_k}$ *to* $\underline{\texttt{succ}_1}$*, the k-loop term can be assigned the following type:*

$$\{\mathbf{WI}_{m,n} \multimap \mathbf{WI}_{m,n+1}, ..., \mathbf{WI}_{m,n+k-1} \multimap \mathbf{WI}_{m,n+k}\} \multimap \mathbf{WI}_{m,n} \multimap \mathbf{WI}_{m,n+k}$$

Observe that this term is not typable in STA*, because it is not possible to assign the same type to every* f_i $(1 \leq i \leq k)$*.*

Now consider the term

$$\underline{\texttt{addmsb}} = \lambda w.\lambda f_0.\lambda f_1.\lambda x.(wF_0F_1(\lambda y.x))\texttt{true}$$

representing the function $\psi : \mathbb{N} \to \mathbb{N}$ such that $\psi(x) = x + 2^{\lfloor \log_2(x) \rfloor + 1}$, which consists of adding a 1-bit in the position to the immediate left of the most significative bit of the input word, where

$$\begin{aligned} F_0 &= \lambda t.\lambda b.f_0(t\,b) \\ F_1 &= \lambda t.\lambda b.(b\,f_1\,I)(f_1(t\,\texttt{false})) \end{aligned}$$

Let Π_0 be

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f_0 : \mathtt{a} \multimap \mathtt{a} \vdash f_0 : \mathtt{a} \multimap \mathtt{a}}\,(Ax) \quad \dfrac{\dfrac{}{t : \mathrm{B} \multimap \mathtt{a} \vdash t : \mathrm{B} \multimap \mathtt{a}}\,(Ax) \quad \dfrac{}{b : \mathrm{B} \vdash b : \mathrm{B}}\,(Ax)}{t : \mathrm{B} \multimap \mathtt{a}, b : \mathrm{B} \vdash t\,b : \mathtt{a}}\,(\multimap E)}{f_0 : \mathtt{a} \multimap \mathtt{a}, t : \mathrm{B} \multimap \mathtt{a}, b : \mathrm{B} \vdash f_0(t\,b) : \mathtt{a}}\,(\multimap E)}{f_0 : \mathtt{a} \multimap \mathtt{a} \vdash F_0 : (\mathrm{B} \multimap \mathtt{a}) \multimap (\mathrm{B} \multimap \mathtt{a})}\,(\multimap I)}{f_0 : \{\mathtt{a} \multimap \mathtt{a}\}^m \vdash F_0 : \{(\mathrm{B} \multimap \mathtt{a}) \multimap (\mathrm{B} \multimap \mathtt{a})\}^m}\,(st)$$

and let Π_1 be

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{b : \mathrm{B} \vdash b : \mathrm{B}}}{b : \mathrm{B} \vdash b : (\mathtt{b} \multimap \mathtt{b} \multimap \mathtt{b})[\mathtt{a} \multimap \mathtt{a}/\mathtt{b}]}\,(\forall E) \quad \dfrac{}{f_1' : \mathtt{a} \multimap \mathtt{a} \vdash f_1' : \mathtt{a} \multimap \mathtt{a}}\,(Ax)}{f_1' : \mathtt{a} \multimap \mathtt{a}, b : \mathrm{B} \vdash b\,f_1' : (\mathtt{a} \multimap \mathtt{a}) \multimap (\mathtt{a} \multimap \mathtt{a})}\,(\multimap E) \quad \vdots\ I : \mathtt{a} \multimap \mathtt{a}}{f_1' : \mathtt{a} \multimap \mathtt{a}, b : \mathrm{B} \vdash b\,f_1'\,I : \mathtt{a} \multimap \mathtt{a}}\,(\multimap E) \quad \Sigma}{f_1' : \mathtt{a} \multimap \mathtt{a}, f_1'' : \mathtt{a} \multimap \mathtt{a}, t : \mathrm{B} \multimap \mathtt{a}, b : \mathrm{B} \vdash (b\,f_1'\,I)(f_1''(t\,\texttt{false})) : \mathtt{a}}\,(\multimap E)}{f_1 : \{\mathtt{a} \multimap \mathtt{a}\}, t : \mathrm{B} \multimap \mathtt{a}, b : \mathrm{B} \vdash (b\,f_1\,I)(f_1(t\,\texttt{false})) : \mathtt{a}}\,(m)}{f_1 : \{\mathtt{a} \multimap \mathtt{a}\} \vdash F_1 : (\mathrm{B} \multimap \mathtt{a}) \multimap (\mathrm{B} \multimap \mathtt{a})}\,(\multimap I)}{f_1 : \{\mathtt{a} \multimap \mathtt{a}\}^{n+1} \vdash F_1 : \{(\mathrm{B} \multimap \mathtt{a}) \multimap (\mathrm{B} \multimap \mathtt{a})\}^n}\,(st)$$

where Σ is

$$\dfrac{\dfrac{}{f_1'' : \mathtt{a} \multimap \mathtt{a} \vdash f_1'' : \mathtt{a} \multimap \mathtt{a}}\,(Ax) \quad \dfrac{\dfrac{}{t : \mathrm{B} \multimap \mathtt{a} \vdash t : \mathrm{B} \multimap \mathtt{a}}\,(Ax) \quad \vdots\ \texttt{false} : \mathrm{B} \vdash \texttt{false} : \mathrm{B}}{t : \mathrm{B} \multimap \mathtt{a} \vdash t\ \texttt{false} : \mathtt{a}}\,(\multimap E)}{f_1'' : \mathtt{a} \multimap \mathtt{a}, t : \mathrm{B} \multimap \mathtt{a} \vdash f_1''(t\ \texttt{false}) : \mathtt{a}}\,(\multimap E)$$

Let $\mathtt{A} = \mathrm{B} \multimap \mathtt{a}$; then $\underline{\mathtt{addmsb}}$ is easily typed as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Phi \quad \vdash \mathtt{true} : \mathrm{B}}}{w : \mathbf{WI}_{m,n}, f_0 : \{\mathtt{a} \multimap \mathtt{a}\}^m, f_1 : \{\mathtt{a} \multimap \mathtt{a}\}^{n+1}, x : \mathtt{a} \vdash (wF_0F_1(\lambda y.x))\mathtt{true} : \mathtt{a}}\ (\multimap E)}{w : \mathbf{WI}_{m,n} \vdash \lambda f_0.\lambda f_1.\lambda x.(wF_0F_1(\lambda y.x))\mathtt{true} : \{\mathtt{a} \multimap \mathtt{a}\}^m \multimap \{\mathtt{a} \multimap \mathtt{a}\}^{n+1} \multimap \mathtt{a} \multimap \mathtt{a}}\ (\multimap I)}{w : \mathbf{WI}_{m,n} \vdash \lambda f_0.\lambda f_1.\lambda x.(wF_0F_1(\lambda y.x))\mathtt{true} : \mathbf{WI}_{m,n+1}}\ (\forall I)}{\vdash \underline{\mathtt{addmsb}} : \mathbf{WI}_{m,n} \multimap \mathbf{WI}_{m,n+1}}\ (\multimap I)$$

where Φ is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{w : \mathbf{WI}_{m,n} \vdash w : \mathbf{WI}_{m,n}}\ (Ax)}{w : \mathbf{WI}_{m,n} \vdash w : \{\mathtt{A} \multimap \mathtt{A}\}^m \multimap \{\mathtt{A} \multimap \mathtt{A}\}^n \multimap \mathtt{A} \multimap \mathtt{A}}\ (\forall E) \quad \Pi_0}{w : \mathbf{WI}_{m,n}, f_0 : \{\mathtt{a} \multimap \mathtt{a}\}^m \vdash wF_0 : \{\mathtt{A} \multimap \mathtt{A}\}^n \multimap \mathtt{A} \multimap \mathtt{A}}\ (\multimap E) \quad \Pi_1}{w : \mathbf{WI}_{m,n}, f_0 : \{\mathtt{a} \multimap \mathtt{a}\}^m, f_1 : \{\mathtt{a} \multimap \mathtt{a}\}^{n+1} \vdash wF_0F_1 : \mathtt{A} \multimap \mathtt{A}}\ (\multimap E) \quad \dfrac{\vdots}{x : \mathtt{a} \vdash \lambda y.x : \mathtt{A}}}{w : \mathbf{WI}_{m,n}, f_0 : \{\mathtt{a} \multimap \mathtt{a}\}^m, f_1 : \{\mathtt{a} \multimap \mathtt{a}\}^{n+1}, x : \mathtt{a} \vdash wF_0F_1(\lambda y.x) : \mathtt{A}}\ (\multimap E)$$

Note that this term is also typable in STA with type $\mathbf{W}_{m,n} \multimap \mathbf{W}_{m,n+1}$.

Example 14 (Iteration of $\underline{\mathtt{addmsb}}$)**.** *In* STR *it is possible to iterate the term* $\underline{\mathtt{addmsb}}$ *a constant number k of times, similarly to what has been shown in Example 13 for* $\underline{\mathtt{succ}_1}$ *since both terms have the same type.*

We conclude with another example of a useful term, which can be typed in STR but not in STA, representing a function through the if-then-else construct.

Example 15 (if-then-else)**.** *Consider the following derivation $\Sigma_{m,n}$:*

$$\dfrac{\dfrac{\Pi \triangleright w : \mathbf{WI}_{m,n} \vdash w(\mathtt{AND\ true}) : \{\mathtt{A}\}^n \multimap \mathtt{A} \quad \dfrac{\dfrac{\vdots}{\vdash \mathtt{AND\ false} : \mathtt{A}}}{\vdash \mathtt{AND\ false} : \{\mathtt{A}\}^n}\ (st)}{w : \mathbf{WI}_{m,n} \vdash w(\mathtt{AND\ true})(\mathtt{AND\ false}) : \mathtt{A}}\ (\multimap E) \quad \dfrac{}{\mathtt{true} : \mathrm{B} \vdash \mathtt{true} : \mathrm{B}}\ (Ax)}{w : \mathbf{WI}_{m,n} \vdash w(\mathtt{AND\ true})(\mathtt{AND\ false})\mathtt{true} : \mathrm{B}}\ (\multimap E)$$

where Π *is*

$$\dfrac{\dfrac{\dfrac{}{w : \mathbf{WI}_{m,n} \vdash w : \mathbf{WI}_{m,n}}\ (Ax)}{w : \mathbf{WI}_{m,n} \vdash w : \{\mathtt{A}\}^m \multimap \{\mathtt{A}\}^n \multimap \mathtt{A}}\ (\forall E) \quad \dfrac{\dfrac{\vdots}{\vdash \mathtt{AND\ true} : \mathtt{A}}}{\vdash \mathtt{AND\ true} : \{\mathtt{A}\}^m}\ (st)}{w : \mathbf{WI}_{m,n} \vdash w(\mathtt{AND\ true}) : \{\mathtt{A}\}^n \multimap \mathtt{A}}\ (\multimap E)$$

$\vdash \mathtt{AND} : \mathrm{B} \multimap \mathrm{B} \multimap \mathrm{B}$ *represents the usual conjunction of boolean values and* $\mathtt{A} = \mathrm{B} \multimap \mathrm{B}$*, for any* $m, n \geq 1$*.*
Let $\mathtt{ISZERO}_w = w(\mathtt{AND\ true})(\mathtt{AND\ false})\mathtt{true}$*; it is easy to check that the term* $\mathtt{ISZERO}_w[\underline{v}/w]$ *reduces to* true *if* $v = 0$ *and to* false *otherwise.*
Now let $\phi : \mathbb{N} \to \mathbb{N}$ *be defined as follows:*

$$\phi(x) = \begin{cases} x & \text{if } x = 0; \\ 2x + 1 & \text{otherwise.} \end{cases}$$

The intuitive closed normal term $\lambda w.\mathtt{ISZERO}_w\, w(\underline{\mathtt{succ}_1}\, w)$*, representing* ϕ*, can be typed in* STR *as follows:*

$$\dfrac{\dfrac{\dfrac{\Pi \quad \dfrac{\vdots}{\vdash \underline{\mathtt{succ}_1} : \mathbf{WI}_{m,n} \multimap \mathbf{WI}_{m,n+1}} \quad \dfrac{}{w_3 : \mathbf{WI}_{m,n} \vdash w_3 : \mathbf{WI}_{m,n}}\,(Ax)}{w_3 : \mathbf{WI}_{m,n} \vdash \underline{\mathtt{succ}_1}\ w_3 : \mathbf{WI}_{m,n+1}}\,(\multimap E)}{w_1 : \mathbf{WI}_{m,n}, w_2 : \mathbf{WI}_{m,n+1}, w_3 : \mathbf{WI}_{m,n} \vdash \mathtt{ISZERO}_{w_1}\ w_2\ (\underline{\mathtt{succ}_1}\ w_3) : \mathbf{WI}_{m,n+1}}\,(\multimap E)}{w : \{\mathbf{WI}_{m,n}, \mathbf{WI}_{m,n+1}\} \vdash \mathtt{ISZERO}_w\ w\ (\underline{\mathtt{succ}_1}\ w) : \mathbf{WI}_{m,n+1}}\,(m)}{\vdash \lambda w.\mathtt{ISZERO}_w\ w\ (\underline{\mathtt{succ}_1}\ w) : \{\mathbf{WI}_{m,n}, \mathbf{WI}_{m,n+1}\} \multimap \mathbf{WI}_{m,n+1}}\,(\multimap I)$$

where Π *is*

$$\dfrac{\dfrac{\Sigma_{m,n} \rhd w_1 : \mathbf{WI}_{m,n} \vdash \mathtt{ISZERO}_{w_1} : \mathrm{B}}{w_1 : \mathbf{WI}_{m,n} \vdash \mathtt{ISZERO}_{w_1} : \mathbf{WI}_{m,n+1} \multimap \mathbf{WI}_{m,n+1} \multimap \mathbf{WI}_{m,n+1}}\,(\forall E) \quad \dfrac{}{w_2 : \mathbf{WI}_{m,n+1} \vdash w_2 : \mathbf{WI}_{m,n+1}}\,(Ax)}{w_1 : \mathbf{WI}_{m,n}, w_2 : \mathbf{WI}_{m,n+1} \vdash \mathtt{ISZERO}_{w_1}\ w_2 : \mathbf{WI}_{m,n+1} \multimap \mathbf{WI}_{m,n+1}}\,(\multimap E)$$

A similar function could be built by using $\underline{\mathtt{addmsb}}$ *instead of* $\underline{\mathtt{succ}_1}$*.*

Observe that this term is not typable in STA*, because it is not possible to assign the same type to every* w_i $(1 \leq i \leq 3)$*.*

3.5 Relations between stratification and intersection

It is natural to wonder whether there is a connection between stratified and intersection types, and if so, why we should prefer one over the other in certain cases. Let $\mathcal{I}$ be the set of intersection and quantifier types, where types are considered to be strict (that is, intersection is not allowed on the right-hand side of the arrow) and intersection is a n-ary connective, for $n \geq 2$:

$$\begin{array}{lcl} \mathtt{C} & ::= & \mathtt{a} \mid \zeta \to \mathtt{C} \mid \forall \mathtt{a}.\mathtt{C} \\ \zeta & ::= & \mathtt{C} \mid \underbrace{\zeta \wedge ... \wedge \zeta}_{n}\ (n \geq 2) \end{array}$$

where $\mathtt{a}$ ranges over the set of type variables previously defined for $\mathcal{T}$.
There is quite a natural translation $(.)^*$ from $\mathcal{T}$ to $\mathcal{I}$, defined inductively as follows:

- $(\mathtt{a})^* = \mathtt{a}$;
- $(\sigma \multimap \mathtt{A})^* = (\sigma)^* \to (\mathtt{A})^*$;
- $(\{\sigma_1, ..., \sigma_n\})^* = (\sigma_1)^* \wedge ... \wedge (\sigma_n)^*$;
- $(\forall \mathtt{a}.\mathtt{A})^* = \forall \mathtt{a}.(\mathtt{A})^*$.

Such translation can be easily extended to bases, so that $(\emptyset)^* = \emptyset$, while $(\Gamma, x : \sigma)^* = (\Gamma)^*, x : (\sigma)^*$.

We can then define a type assignment system INTER, obtained from STR by applying the translation $(.)^*$ to both bases and types, such that for each rule (R) of STR:

$$\frac{(\Gamma_i \vdash \mathcal{M} : \sigma_i)_{i\in I}}{\Gamma \vdash \mathcal{M} : \sigma}\ (R)$$

where the cardinality of I depends on (R), there is a corresponding rule (R^*) in INTER:

$$\frac{((\Gamma_i)^* \vdash_{\mathtt{I}} \mathcal{M} : (\sigma_i)^*)_{i\in I}}{(\Gamma)^* \vdash \mathcal{M} : (\sigma)^*}\ (R^*)$$

Intersection is usually considered modulo idempotence ($\zeta = \zeta \wedge \zeta$), commutativity ($\zeta_1 \wedge \zeta_2 = \zeta_2 \wedge \zeta_1$) and associativity $(\zeta_1 \wedge \zeta_2) \wedge \zeta_3 = \zeta_1 \wedge (\zeta_2 \wedge \zeta_3)$. It is easy to check that the system INTER is equivalent to STR, namely that $\Gamma \vdash \mathcal{M} : \sigma$ if and only if $(\Gamma \vdash \mathcal{M} : \sigma)^*$, with the proviso that intersection is considered modulo idempotence and commutativity, *but not associativity*.

Note that the non-associative approach is not the standard use of intersection, which usually enjoys idempotence, associativity and commutativity. However, the presence of all these properties has the effect of erasing every quantitative information from the typing: this is not an issue as long as intersection types are used for proving qualitative properties of terms, as has been done for a long time, yet it proves to be troublesome when one wants to inquire about quantitative properties, such as the complexity of reduction, which will further be explored in the next chapter, or ICC related questions, as has been done in the previous sections.

Even more worthy of note is the fact that stratified types, despite being idempotent, are able to store a finer information on the usage of resources with respect to usual intersection types. Consider for example the intersection and the stratification of two copies of a type: while $\zeta \wedge \zeta = \zeta$ does not retain any memory of the intersection from which it is obtained, the stratification of σ and σ results into a type $\{\sigma\}$. Therefore, while the latter gives no information about the number of occurrences of type σ in it, it still remembers that *some* occurrences of σ have been contracted in order to obtain the current type.
For such reasons, and in order to stress that we consider intersection as set formation, we chose the notation of stratified types over that of intersection types.

Chapter 4

Bounding normalization time through intersection types

In Chapter 3 we studied a typing system using stratified types, whose behavior is essentially that of non-associative intersection types.
Actually, an earlier attempt in such direction was that of designing a type assignment system, similar to `STR`, where types did not enjoy neither idempotence nor associativity, in order to represent intersection through multisets, instead of sets.

As was already noted in the previous chapter, removing the associativity property is required in order to express a bound on the complexity of the reduction: indeed the stratification, derived from the lack of associativity, is crucial in order to obtain a decreasing measure on the derivation which, in turn, allows to give a bound on the number of nested duplications of subterms. Here we explore a further restriction of intersection types, namely the absence of both associativity and *idempotence*.

Intersection types have been introduced essentially with the aim of analyzing λ-models and the normalization properties of λ-terms; since the interpretation of types as properties of terms induces naturally the idempotence property, intersection was historically considered modulo idempotence.
Nevertheless, recently it has been observed that, when dropping idempotence, intersection types can be used for reasoning about quantitative properties, such as the complexity of β-reduction. Some results have been obtained along this line: in [Ter06] a typing system is designed, assigning non-idempotent intersection types to λ-terms, in which all and only the strongly normalizing terms are typed and the size of a derivation with subject $\mathcal{M}$ is bigger than the size of every term in the β-reduction sequence from $\mathcal{M}$ to its normal form; such property can further be exploited in order to compute a bound of every normalizing β-reduction sequence starting from $\mathcal{M}$.
A more precise result in this direction is found in [BL11], where the authors give a precise account of the number of β-reduction steps of a term to its normal form;

this result is achieved by designing a type assignment system where intersection is considered without idempotence, upon which both notions of measure of a derivation and of principal derivation for a given term are defined: finally it is shown that the measure of a principal derivation of a type for a normalizing term $\mathcal{M}$ corresponds to the maximal length of a normalizing β-reduction sequence for $\mathcal{M}$.

Other type assignment systems without idempotence have been studied in the literature for various purposes. In [KW04] non-idempotent intersection is employed in order to formalize a type inference semi-algorithm, whose complexity is further studied in [NM04]. Moreover, in [Kfo00] non idempotent intersection types are studied in relation with linear β-reduction.
Recently, non idempotent intersection types have been used in [PR10] with the aim of characterizing the solvability in the resource λ-calculus, whereas in [DHL08] the game semantics of a typed λ-calculus is described in logical form using an intersection type assignment system, where the intersection does not enjoy any of its usual properties. Some complexity results via non-idempotent intersection are given in [dCa09], through a λ-algebra induced by non idempotent intersection types, and in [BEM07], where a logical description of relational model of λ-calculus has been designed through a non-idempotent type assignment system.
A quantitative approach is also taken in [KV14], where non-idempotent intersection types are employed, in the guise of multisets of types, in order to design two type assignment systems for a calculus with explicit substitutions; by allowing the controlled use of the empty multiset, the authors give a characterization of linear-head, head and weak normalization, while the characterization of strong normalization is obtained by requiring all subterms to be typed, possibly by a witness-type.

Our aim here is to take a further step in this direction; namely, we want to use non-idempotent, non-associative intersection types in order to to express the functional dependence of the length of a normalizing β-reduction sequence for a term $\mathcal{M}$ on the size of $\mathcal{M}$ itself.
In order to obtain such result, we again take inspiration from the system `STA` [GR07] which characterizes polynomial time computation. The resulting typing system, called `STI`, allows us to give a bound on the number of steps necessary to reduce a normalizing term $\mathcal{M}$ to its normal form, of the form $|\mathcal{M}|^{d+1}$, where $|\mathcal{M}|$ is the size of the term and d is a measure of *depth* depending on the type derivation for it. Note that, since for every normalizing term there is a type derivation with minimal depth, this bound does not depend on a particular derivation.
Here a result very similar to that of Theorem 15 is proved to hold; however, such result is not very suitable for an implicit characterization of complexity classes: indeed non-idempotent intersection types are too informative, in the sense that there is not a common type which can be assigned to all usual data (such as Church

integers or binary words); as a consequence, the notion of datatype is not satisfied and so a characterization of complexity classes through STI is not achievable. As a matter of fact, in Chapter 3 we showed that taking away the associativity property is sufficient in order to obtain a polynomial characterization.

Outline of this chapter We start by presenting the type assignment system STI for pure λ-calculus and by proving its subject reduction property (Section 4.1). As usual for intersection based systems, we must consider the possible presence of many subderivations typing the same redex, whose parallel elimination is necessary in order to obtain a correct proof. In order to clarify this matter, an example of reduction is shown at the end of the section.
After introducing some necessary measures for terms and derivations, we give a weighted version of the subject reduction theorem which shows that the overall weight of a derivation strictly decreases whenever a reduction step is performed; this result then allows to bound both the number of reduction steps and the size of the normal form with respect to the size of the initial term and of the depth of the derivation (Section 4.2).
Finally, we prove that STI characterizes strongly normalizing terms (Section 4.3), by combining the result of the previous section and the adaptation to system STI of the proof given in Section 3.3.2.
The content of this chapter is introduced in [DB11] and further developed in [DR13].

4.1 Syntax and properties of system STI

We introduce STI (Soft Type assignment with Intersection), a typing system for λ-calculus assigning intersection types to λ-terms, where intersection is assumed to enjoy neither associativity nor idempotence.

4.1.1 Definition of the typing system

The definition of the language of terms and of the rewriting rule is the same already given in Section 3.2.1 for pure λ-calculus. Likewise, we say that a term $\mathcal{M}$ is an *instance* of $\mathcal{N}$ when $\mathcal{M}$ is obtained from $\mathcal{N}$ by renaming a subset of its free variables with a unique fresh name, as in Definition 20.

The types of STI are either linear or intersection types, the latter enjoying only the commutativity property:

$$\frac{}{x : \mathtt{A} \vdash x : \mathtt{A}}\ (Ax) \qquad \frac{\Gamma \vdash \mathcal{M} : \sigma \quad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x : \mathtt{A} \vdash \mathcal{M} : \sigma}\ (w)$$

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

$$\frac{\Gamma, x_1 : \sigma_1, ..., x_n : \sigma_n \vdash \mathcal{M} : \tau}{\Gamma, x : \sigma_1 \wedge ... \wedge \sigma_n \vdash \mathcal{M}[x/x_i]_{i=1}^n : \tau}\ (m) \qquad \frac{(\Gamma_i \vdash \mathcal{M} : \sigma_i)_{1 \leq i \leq n} \quad \boxminus_{i=1}^n \Gamma_i}{\wedge_{i=1}^n \Gamma_i \vdash \mathcal{M} : \sigma_1 \wedge ... \wedge \sigma_n}\ (\wedge_n)$$

Table 4.1: Derivation rules of system STI.

Definition 26 (Types)**.** *The set of* STI *types is defined by the following syntax:*

$$\mathtt{A} ::= \mathtt{a} \mid \sigma \multimap \mathtt{A} \qquad \textit{(linear types)}$$
$$\sigma ::= \mathtt{A} \mid \underbrace{\sigma \wedge ... \wedge \sigma}_{n} \qquad \textit{(intersection types)}$$

where a *ranges over a countable set of type variables and* $n \geq 2$*.*

Intersection types are considered modulo the equivalence $\sigma_1 \wedge ... \wedge \sigma_n = \sigma_{i_1} \wedge ... \wedge \sigma_{i_n}$, for every permutation $(i_1, ..., i_n)$ of $\{1, ..., n\}$.
Note that a type σ can be loosely considered as a multiset $\overline{\sigma}$, according to Definition 19, where the multiset of linear components is defined inductively as

$$\overline{\mathtt{A}} = [\mathtt{A}] \qquad \overline{\sigma_1 \wedge ... \wedge \sigma_k} = \overline{\sigma}_1 \uplus ... \uplus \overline{\sigma}_k.$$

The number of occurrences $|\sigma|$ is simply the cardinality of the multiset $\overline{\sigma}$.

As usual a *basis*, ranged over by Γ, Δ, Θ, is a partial function mapping term variables to types, whose domain is denoted by $\mathrm{dom}(\Gamma)$.
Similarly to Notation 2, we adopt the following conventions for bases:

Notation 5 (Bases)**.**

i. The condition $\Gamma_1 \# ... \# \Gamma_n$ *(or* $\#_{i=1}^n \Gamma_i$*) holds if and only if* $j \neq h$ *implies* $\mathrm{dom}(\Gamma_j) \cap \mathrm{dom}(\Gamma_h) = \emptyset$*, for* $1 \leq j, h \leq n$*.*

ii. The condition $\Gamma_1 \boxminus ... \boxminus \Gamma_n$ *(or* $\boxminus_{i=1}^n \Gamma_i$*) holds if and only if* $j \neq h$ *implies* $\mathrm{dom}(\Gamma_j) = \mathrm{dom}(\Gamma_h)$*, for* $1 \leq j, h \leq n$*.*

iii. The n-ary intersection of bases, denoted by $\Gamma_1 \wedge ... \wedge \Gamma_n$ *or* $\wedge_{i=1}^n \Gamma_i$*, is the basis such that* $(\wedge_{i=1}^n \Gamma_i)(x) = \wedge_{i=1}^n (\Gamma_i(x))$*, where* $\boxminus_{i=1}^n \Gamma_i$*.*

iv. The basis $\Gamma_1, ..., \Gamma_n$ *represents the concatenation of bases, provided that* $\#_{i=1}^{n}\Gamma_i$.

The system STI proves sequents of the shape $\Gamma \vdash \mathcal{M} : \sigma$, where Γ is a basis, $\mathcal{M}$ is a λ-term and σ is a type. The rules are given in Table 4.1.

Derivations are ranged over by Π, Σ, such that $\Pi \triangleright \Gamma \vdash \mathcal{M} : \sigma$ denotes a particular derivation Π with conclusion $\Gamma \vdash \mathcal{M} : \sigma$.

Observe that, as in system STR, terms are first built in a linear form and then variables are unified through the explicit multiplexor rule (m): this allows to control the number of contractions, which is responsible for the growth of the reduction time. The counterpart of the contraction on the right-hand side of a derivation is handled by rule $(\wedge_n)$, which is parametric in n for $n \geq 2$. In designing system STI, we again took inspiration from SLL [Laf04].
We still retain the distinction between *constructive* rules, which contribute in building the subject, and non constructive rules: the former are rules (Ax), $(\multimap I)$ and $(\multimap E)$, while the latter are rules (w), (m) and $(\wedge_n)$. As before, we denote by *renaming sequence* a (possibly empty) sequence of applications of rules (w) and (m), whose domain and range are defined as in Notation 3.

4.1.2 Properties of STI

The notion of *intersection tree* proves to be quite useful in dealing with derivations assigning intersection types to their subject:

Definition 27 (Intersection tree)**.** *The intersection tree of a derivation* Π *is a subderivation of* Π *defined inductively as follows:*

- *if* Π *is*

$$\dfrac{\Sigma \vdash \mathcal{N} : \mathtt{A}}{\Gamma \vdash \mathcal{M} : \sigma}\ \delta$$

where δ *is a renaming sequence and* Σ *is a derivation whose last application is of a constructive rule, then the intersection tree of* Π *has conclusion* $\Gamma \vdash \mathcal{M} : \sigma$ *and a single leaf* Σ*;*

- *if* Π *is*

$$\dfrac{\dfrac{(\Sigma_i \triangleright \Gamma_i \vdash \mathcal{N} : \sigma_i)_{1 \leq i \leq n}}{\wedge_{i=1}^{n}\Gamma_i \vdash \mathcal{N} : \sigma_1 \wedge ... \wedge \sigma_n}\ (\wedge_n)}{\Gamma \vdash \mathcal{M} : \sigma_1 \wedge ... \wedge \sigma_n}\ \delta$$

where δ *is a renaming sequence, then the intersection tree of* Π *has conclusion* $\Gamma \vdash \mathcal{M} : \sigma_1 \wedge ... \wedge \sigma_n$ *and its subtrees are the intersection trees of* $\Sigma_1, ..., \Sigma_n$.

Example 16. *Let* Π *be*

$$\dfrac{\Pi_{\mathtt{A}}\triangleright \vdash \mathcal{M} : \mathtt{A} \quad \Pi_{\mathtt{B}}\triangleright \vdash \mathcal{M} : \mathtt{B} \qquad \dfrac{\Pi_{\mathtt{C}}\triangleright \vdash \mathcal{M} : \mathtt{C} \quad \Pi_{\mathtt{D}}\triangleright \vdash \mathcal{M} : \mathtt{D}}{\vdash \mathcal{M} : \mathtt{C} \wedge \mathtt{D}}\ (\wedge_2)}{\vdash \mathcal{M} : \mathtt{A} \wedge \mathtt{B} \wedge (\mathtt{C} \wedge \mathtt{D})}\ (\wedge_3)$$

where $\Pi_{\mathtt{A}}, \Pi_{\mathtt{B}}, \Pi_{\mathtt{C}}, \Pi_{\mathtt{D}}$ *all end with an application of a structural rule: then the intersection tree of* Π *has conclusion* $\vdash \mathcal{M} : \mathtt{A} \wedge \mathtt{B} \wedge (\mathtt{C} \wedge \mathtt{D})$ *and leaves* $\Pi_{\mathtt{A}}, \Pi_{\mathtt{B}}, \Pi_{\mathtt{C}}, \Pi_{\mathtt{D}}$.

Since rule $(\wedge_n)$ is the only rule building an intersection type on the right of the turnstile symbol, it is possible to state the following key property of `STI`:

Property 18 (Subject with intersection type)**.** *Let* $\Pi \rhd \Gamma \vdash \mathcal{M} : \sigma_1 \wedge ... \wedge \sigma_n$ *with* $n \geq 2$*; then:*

i. Π *ends with an application of rule* $(\wedge_n)$*, possibly followed by a renaming sequence;*

ii. Π *ends with an intersection tree with at least* n *leaves.*

Proof. By induction on the shape of Π. Note that the Π must end with a non constructive rule.

i. Easy, since rule $(\wedge_n)$ is the only rule introducing the $\wedge$ connective on the right-hand side of the turnstile symbol.

ii. Let Π end with an application of rule $(\wedge_n)$: then the statement is trivially true and δ is the empty sequence.

   Otherwise, let Π end with an application of either rule (w) or (m): then the proof follows by induction.

□

By employing the previous result, we prove that the substitution property holds for terms having disjoint free variables sets:

Lemma 33 (Substitution)**.** *Let* $\Pi \rhd \Gamma, x : \sigma \vdash \mathcal{M} : \tau$ *and* $\Sigma \rhd \Delta \vdash \mathcal{N} : \sigma$*, where* $\Gamma \# \Delta$ *and* $x \notin dom(\Delta)$*; then there is* $\Phi \rhd \Gamma, \Delta \vdash \mathcal{M}[\mathcal{N}/x] : \tau$.

Proof. By induction on Π. If Π ends with an application of a constructive rule, then the proof is trivial; we thus consider the three remaining cases.

Let Π end with an application of rule (w). If the range of the application is $\{y\}$ such that $y \neq x$, then the proof follows by induction. Otherwise, let Π be

$$\frac{\Pi' \rhd \Gamma \vdash \mathcal{M} : \tau \quad x \notin \mathrm{dom}(\Gamma)}{\Pi \rhd \Gamma, x : \mathtt{A} \vdash \mathcal{M} : \tau}\ (w)$$

If Δ maps each variable of its domain to a linear type, then Φ is

$$\frac{\Pi' \rhd \Gamma \vdash \mathcal{M} : \tau}{\Gamma, \Delta \vdash \mathcal{M} : \tau}\ (w)$$

Otherwise, let us assume that $\Delta = \Delta', y : \rho$, such that $\overline{\rho} = [\mathtt{A}_1, ..., \mathtt{A}_n]$ and Δ' contains only bindings of variables to linear types: then Φ is

$$\dfrac{\dfrac{\Pi' \rhd \Gamma \vdash \mathcal{M} : \tau}{\Gamma, \Delta', y_1 : \mathtt{A}_1, ..., y_n : \mathtt{A}_n \vdash \mathcal{M} : \tau}\ (w)}{\Gamma, \Delta', y : \rho \vdash \mathcal{M} : \tau}\ (m)$$

where the sequence of applications of rule (m) builds the desired intersection type ρ. The proof can be easily extended to the general case.

Let Π be

$$\dfrac{\Pi' \rhd \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash \mathcal{M} : \tau}{\Gamma, x : \sigma_1 \wedge ... \wedge \sigma_n \vdash \mathcal{M}[x/x_i]_{i=1}^n : \tau}\ (m)$$

where $\sigma = \sigma_1 \wedge ... \wedge \sigma_n$.

By applying Property 18 to Σ, there are $\Sigma_i \rhd \Delta_i \vdash \mathcal{N}' : \sigma_i$, for $1 \leq i \leq n$, such that Σ is obtained by one application of rule $(\wedge_n)$ to $(\Sigma_i)_{i=1}^n$ and then by a renaming sequence δ.

By renaming the variables of Δ_i, we can easily obtain $\Sigma_i' \rhd \Delta_i' \vdash \mathcal{N}_i' : \sigma_i$, such that $\mathcal{N}_i'$ is an instance of $\mathcal{N}'$ and $\#_{i=1}^n \Delta_i'$. Not that this is not restrictive, since we are able to easily recover Δ' and $\mathcal{N}'$ by applying a suitable renaming sequence ρ.

By inductive hypothesis, there are

$$\begin{array}{lcl}
\Phi_1 & \rhd & \Gamma, \Delta_1', x_2 : \sigma_2, ..., x_n : \sigma_n \vdash \mathcal{M}[\mathcal{N}_1'/x_1] : \tau \\
\Phi_2 & \rhd & \Gamma, \Delta_1', \Delta_2', x_3 : \sigma_3, ..., x_n : \sigma_n \vdash \mathcal{M}[\mathcal{N}_i'/x_i]_{i=1}^2 : \tau \\
& \vdots & \\
\Phi_n & \rhd & \Gamma, \Delta_1', ..., \Delta_n' \vdash \mathcal{M}[\mathcal{N}_i'/x_i]_{i=1}^n : \tau
\end{array}$$

Then Φ is

$$\dfrac{\dfrac{\Phi_n \rhd \Gamma, \Delta_1', ..., \Delta_n' \vdash \mathcal{M}[\mathcal{N}_i'/x_i]_{i=1}^n : \tau}{\Gamma, \Delta' \vdash \mathcal{M}[\mathcal{N}'/x_i]_{i=1}^n : \tau}\ \rho}{\rhd\Gamma, \Delta \vdash \mathcal{M}[\mathcal{N}/x_i]_{i=1}^n : \tau}\ \delta$$

where ρ and δ are the previously defined renaming sequences.

Let Π be

$$\dfrac{(\Pi_i \rhd \Gamma_i, x : \sigma_i \vdash \mathcal{M} : \tau_i)_{1 \leq i \leq n}}{\Gamma, x : \sigma \vdash \mathcal{M} : \tau}\ (\wedge_n)$$

where $\Gamma = \wedge_{i=1}^n \Gamma_i$, $\sigma = \sigma_1 \wedge ... \wedge \sigma_n$ and $\tau = \tau_1 \wedge ... \wedge \tau_n$. By Property 18, Σ is

$$\dfrac{\dfrac{(\Sigma_i \rhd \Delta_i \vdash \mathcal{N}' : \sigma_i)_{1 \leq i \leq n}}{\Delta' \vdash \mathcal{N}' : \sigma_1 \wedge ... \wedge \sigma_n}\ (\wedge_n)}{\Delta \vdash \mathcal{N} : \sigma_1 \wedge ... \wedge \sigma_n}\ (\delta)$$

where δ is a renaming sequence and $\mathcal{N}$ is an instance of $\mathcal{N}'$. We can assume without loss of generality that $\Gamma_i \# \Delta_i$, for $1 \leq i \leq n$, since the renaming allows to consider

disjoint sets of variables.
By inductive hypothesis there is $\Phi_i \triangleright \Gamma_i, \Delta_i \vdash \mathcal{M}[\mathcal{N}'/x] : \tau_i$, for $1 \leq i \leq n$: then Φ is

$$\dfrac{\dfrac{(\Phi_i \triangleright \Gamma_i, \Delta_i \vdash \mathcal{M}[\mathcal{N}'/x] : \tau_i)_{1 \leq i \leq n}}{\Gamma, \Delta' \vdash \mathcal{M}[\mathcal{N}'/x] : \tau_1 \wedge ... \wedge \tau_n}\ (\wedge_n)}{\Gamma, \Delta \vdash \mathcal{M}[\mathcal{N}/x] : \tau_1 \wedge ... \wedge \tau_n}\ (\delta)$$

□

The following generation lemma relates the structure of a term with its typing:

Lemma 34 (Generation)**.** *Let* $\Pi \triangleright \Gamma \vdash \mathcal{M} : \mathtt{A}$ *and let* $\Sigma \triangleright \Delta \vdash \mathcal{N} : \mathtt{B}$ *be the smallest subderivation of* Π *such that* Σ *is obtained from* Π *by a renaming sequence:*

i. if $x \in \mathrm{FV}(\mathcal{M})$*, then* $x \in \mathrm{dom}(\Gamma)$*;*

ii. if $\mathcal{M} = x$*, then* Σ *ends with an application of rule* (Ax)*;*

iii. if $\mathcal{M} = \lambda x.\mathcal{P}$*, then* Σ *ends with an application of rule* $(\multimap I)$ *and* $\mathtt{A} = \sigma \multimap \mathtt{B}$ *for some* σ *and* $\mathtt{B}$*;*

iv. if $\mathcal{M} = \mathcal{P}\mathcal{Q}$*, then* Σ *ends with an application of rule* $(\multimap E)$*.*

Proof. Easy by considering each constructive rule. □

As in Chapter 3, in order to prove the subject reduction property, we must take into account the fact that one step of β-reduction on the subject can correspond to $n \geq 1$ parallel detour eliminations in the underlying derivation, whose behavior is that of reducing all copies of the same redex which are assigned different types:

Theorem 20 (Subject reduction)**.** $\Pi \triangleright \Gamma \vdash \mathcal{M} : \sigma$ *and* $\mathcal{M} \to \mathcal{M}'$ *implies* $\Pi' \triangleright \Gamma \vdash \mathcal{M}' : \sigma$*.*

Proof. If $\mathcal{M} \to \mathcal{M}'$, then there is a context $\mathcal{C}$ such that $\mathcal{M} = \mathcal{C}[(\lambda x.\mathcal{Q})\mathcal{N}]$ and $\mathcal{M}' = \mathcal{C}[\mathcal{Q}[\mathcal{N}/x]]$; the proof is by induction on $\mathcal{C}$.

Let $\mathcal{C} = \Box$, so $\mathcal{M} = (\lambda x.\mathcal{Q})\mathcal{N}$.
By Lemma 34, each leaf Π_i of the intersection tree of Π is

$$\dfrac{\dfrac{\dfrac{\Sigma_i' \triangleright \Gamma_i', x : \sigma_i \vdash \mathcal{Q}_i' : \mathtt{A}_i}{\Gamma_i' \vdash \lambda x.\mathcal{Q}_i' : \sigma_i \multimap \mathtt{A}_i}\ (\multimap I)}{\Gamma_i \vdash \lambda x.\mathcal{Q}_i : \sigma_i \multimap \mathtt{A}}\ (\delta_i) \qquad \Sigma_i'' \triangleright \Delta_i \vdash \mathcal{N}_i : \sigma_i \quad \Gamma_i \# \Delta_i}{\Gamma_i, \Delta_i \vdash (\lambda x.\mathcal{Q}_i)\mathcal{N}_i : \mathtt{A}_i}\ (\multimap E)$$

where $(\lambda x.\mathcal{Q})\mathcal{N}$ is an instance of $(\lambda x.\mathcal{Q}_i)\mathcal{N}_i$ and δ_i is a renaming sequence, for $1 \leq i \leq n$.

Since the domain of each application of rule (m) in δ_i deals with variables in $\mathrm{dom}(\Gamma'_i)$, sequence δ_i can be delayed in order to obtain the derivation

$$\dfrac{\dfrac{\dfrac{\Sigma'_i \triangleright \Gamma'_i, x : \sigma_i \vdash \mathcal{Q}'_i : \mathtt{A}}{\Gamma'_i \vdash \lambda x.\mathcal{Q}'_i : \sigma_i \multimap \mathtt{A}_i}\,(\multimap I) \qquad \Sigma''_i \triangleright \Delta_i \vdash \mathcal{N}_i : \sigma_i}{\Gamma'_i, \Delta_i \vdash (\lambda x.\mathcal{Q}'_i)\mathcal{N}_i : \mathtt{A}_i}\,(\multimap E)}{\Gamma_i, \Delta_i \vdash (\lambda x.\mathcal{Q}_i)\mathcal{N}_i : \mathtt{A}_i}\,(\delta_i)$$

By Lemma 33, there are $\Pi'_i \triangleright \Gamma_i, \Delta_i \vdash \mathcal{Q}_i[\mathcal{N}_i/x] : \mathtt{A}_i$: then Π' is obtained by replacing each leaf Π_i by Π'_i in the intersection tree of Π.

The cases of $\mathcal{C} \neq \Box$ follow easily by induction.

□

Non-idempotent intersection types allow to obtain a precise quantification of the occurrences of a variable in a normal term; indeed, we can think of an intersection as a multiset of types, which can be used to *count* the occurrences of free variables in a normal form:

Property 19 (Normal forms)**.** *Let $\mathcal{M}$ be a normal form; then there is a derivation $\Pi \triangleright \Gamma \vdash \mathcal{M} : \mathtt{A}$ such that $|\Gamma(x)| = n_0(x, \mathcal{M})$ for every $x \in \mathrm{FV}(\mathcal{M})$.*

Proof. By induction on the shape of $\mathcal{M}$.

Let $\mathcal{M} = x$: then the desired derivation is obtained by applying rule (Ax).

Let $\mathcal{M} = \lambda x.\mathcal{P}$, where $\mathcal{P}$ is in normal form. If $x \in \mathrm{FV}(\mathcal{P})$, then by inductive hypothesis there is a derivation $\Pi' \triangleright \Gamma, x : \sigma \vdash \mathcal{P} : \mathtt{B}$ such that $|\Gamma(y)| = n_0(y, \mathcal{P})$ for every $y \in \mathrm{FV}(\mathcal{P})$: then the desired derivation is obtained by applying rule $(\multimap I)$ to Π' to abstract over x.
If $x \notin \mathrm{FV}(\mathcal{P})$, by inductive hypothesis there is a derivation $\Pi' \triangleright \Gamma \vdash \mathcal{P} : \mathtt{B}$ such that $|\Gamma(x)| = n_0(x, \mathcal{P})$ for every $x \in \mathrm{FV}(\mathcal{P})$: then the desired derivation is obtained by applying rule (w) with range $\{x\}$ to Π', followed by the application of rule $(\multimap I)$ abstracting over x.

Let $\mathcal{M} = y\mathcal{N}_1 ... \mathcal{N}_n$, where $\mathcal{N}_1, \ldots, \mathcal{N}_n$ are normal forms. Let $\mathcal{M}$ be an instance of $\mathcal{M}' = y'\mathcal{N}'_1 ... \mathcal{N}'_n$, such that $\mathrm{FV}(y') \cap \mathrm{FV}(\mathcal{N}'_1) \cap \ldots \cap \mathrm{FV}(\mathcal{N}'_n) = \emptyset$.
By inductive hypothesis there are n derivations $\Pi_i \triangleright \Gamma'_i \vdash \mathcal{N}'_i : \mathtt{A}_i$ such that $|\Gamma'_i(x)| = n_0(x, \mathcal{N}'_i)$ for every $x \in \mathrm{FV}(\mathcal{N}'_i)$. Let $\mathtt{C} = \mathtt{A}_1 \multimap \mathtt{A}_2 \multimap \ldots \multimap \mathtt{A}_m \multimap \mathtt{B}$: then Π is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y' : \mathtt{C} \vdash y' : \mathtt{C}}\,(Ax) \qquad \Pi_1}{y' : \mathtt{C}, \Gamma'_1 \vdash y'\mathcal{N}'_1 : \mathtt{A}_2 \multimap \ldots \multimap \mathtt{A}_n \multimap \mathtt{B}}\,(\multimap E) \qquad \Pi_2}{\vdots}\,(\multimap E)}{y' : \mathtt{C}, \Gamma'_1, \ldots, \Gamma'_{n-1} \vdash y'\mathcal{N}'_1 ... \mathcal{N}'_{n-1} : \mathtt{A}_n \multimap \mathtt{B}} \qquad \Pi_n}{y' : \mathtt{C}, \Gamma'_1, \ldots, \Gamma'_n \vdash y'\mathcal{N}'_1 \ldots \mathcal{N}'_n : \mathtt{B}}\,(\multimap E)}{\Gamma \vdash y\mathcal{N}_1 ... \mathcal{N}_n : \mathtt{B}}\,(m)$$

where the renaming sequence contains $k \geq 0$ applications of rule (m), each having domain $\mathcal{Y}_j$ and range $\{z_j\}$, where $\mathcal{Y}_j \subseteq \mathrm{FV}(\mathcal{M}')$ and $z_j \in \mathrm{FV}(\mathcal{M})$, for $1 \leq j \leq k$. Let $\Gamma' = y' : \mathtt{C}, \Gamma'_1, \ldots, \Gamma'_n$: then $|\Gamma(z_j)| = \sum_{y \in \mathcal{Y}_j} |\Gamma'(y)| = \sum_{y \in \mathcal{Y}_j} n_0(y, \mathcal{M}') = n_0(z_j, \mathcal{M})$, for every $1 \leq j \leq k$, while $|\Gamma(x)| = |\Gamma'(x)| = n_0(x, \mathcal{M})$ for every $x \in \mathrm{dom}(\mathcal{M}) \cap \{z_1, \ldots, z_k\}$.

□

Example 17. *We show an example of a derivation in* STI*, with the aim of clarifying the behavior of the subject reduction in the case where the intersection tree has at least* 2 *leaves.*
Consider the following derivations:

$$\dfrac{\dfrac{\dfrac{}{y : \mathtt{A} \vdash y : \mathtt{A}}\,(Ax)}{\vdash \lambda y.y : \mathtt{A} \multimap \mathtt{A}}\,(\multimap I) \qquad \dfrac{}{z : \mathtt{A} \vdash z : \mathtt{A}}\,(Ax)}{\Sigma_1 \rhd z : \mathtt{A} \vdash (\lambda y.y)z : \mathtt{A}}\,(\multimap E)$$

and

$$\dfrac{\dfrac{\dfrac{}{y : \mathtt{a} \vdash y : \mathtt{a}}\,(Ax)}{\vdash \lambda y.y : \mathtt{a} \multimap \mathtt{a}}\,(\multimap I) \qquad \dfrac{}{z : \mathtt{a} \vdash z : \mathtt{a}}\,(Ax)}{\Sigma_2 \rhd z : \mathtt{a} \vdash (\lambda y.y)z : \mathtt{a}}\,(\multimap E)$$

where $\mathtt{A} = \mathtt{a} \multimap \mathtt{a}$.
By reducing the term $(\lambda x.xx)((\lambda y.y)z)$*, we first obtain*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x_1 : \mathtt{A} \vdash x_1 : \mathtt{A}}\,(Ax) \quad \dfrac{}{x_2 : \mathtt{a} \vdash x_2 : \mathtt{a}}\,(Ax)}{x_1 : \mathtt{A}, x_2 : \mathtt{a} \vdash x_1 x_2 : \mathtt{a}}\,(\multimap E)}{x : \mathtt{A} \wedge \mathtt{a} \vdash xx : \mathtt{a}}\,(m)}{\vdash \lambda x.xx : (\mathtt{A} \wedge \mathtt{a}) \multimap \mathtt{a}}\,(\multimap I) \qquad \dfrac{\Sigma_1 \rhd z : \mathtt{A} \vdash \boxed{(\lambda y.y)z} : \mathtt{A} \quad \Sigma_2 \rhd z : \mathtt{a} \vdash \boxed{(\lambda y.y)z} : \mathtt{a}}{\Sigma \rhd z : \mathtt{A} \wedge \mathtt{a} \vdash (\lambda y.y)z : \mathtt{A} \wedge \mathtt{a}}\,(\wedge_2)}{z : \mathtt{A} \wedge \mathtt{a} \vdash (\lambda x.xx)((\lambda y.y)z) : \mathtt{a}}\,(\multimap E)$$

Observe that there are two virtual copies of the same redex; therefore, if we reduce the redex $(\lambda y.y)z$*, we obtain*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x_1 : \mathtt{A} \vdash x_1 : \mathtt{A}}\,(Ax) \quad \dfrac{}{x_2 : \mathtt{a} \vdash x_2 : \mathtt{a}}\,(Ax)}{x_1 : \mathtt{A}, x_2 : \mathtt{a} \vdash x_1 x_2 : \mathtt{a}}\,(\multimap E)}{x : \mathtt{A} \wedge \mathtt{a} \vdash xx : \mathtt{a}}\,(m)}{\vdash \lambda x.xx : (\mathtt{A} \wedge \mathtt{a}) \multimap \mathtt{a}}\,(\multimap I) \qquad \dfrac{\dfrac{}{z : \mathtt{A} \vdash z : \mathtt{A}}\,(Ax) \quad \dfrac{}{z : \mathtt{a} \vdash z : \mathtt{a}}\,(Ax)}{z : \mathtt{A} \wedge \mathtt{a} \vdash z : \mathtt{A} \wedge \mathtt{a}}\,(\wedge_2)}{\Pi \rhd z : \mathtt{A} \wedge \mathtt{a} \vdash (\lambda x.xx)z : \mathtt{a}}\,(\multimap E)$$

where the subject of both Σ_1 *and* Σ_2 *has been reduced.*
Finally we reduce $(\lambda x.xx)z$*, so obtaining the proof*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{z_1 : \mathtt{A} \vdash z_1 : \mathtt{A}}\,(Ax) \quad \dfrac{}{z_2 : \mathtt{a} \vdash z_2 : \mathtt{a}}\,(Ax)}{z_1 : \mathtt{A}, z_2 : \mathtt{a} \vdash z_1 z_2 : \mathtt{a}}\,(\multimap E)}{z : \mathtt{A} \wedge \mathtt{a} \vdash zz : \mathtt{a}}\,(m)$$

Note that, as explained in the proof of Lemma 33, the premises of rule $(\wedge_2)$ *need to be rewritten in the substitution so that their contexts are disjoint; the original context is then recovered through a suitable renaming sequence.*

4.2 Normalization bound

In computing the normalization bound for system `STI`, we must take into account the mismatch between proof simplification and β-reduction, since the presence of intersection allows the coexistence of many subderivations typing the same subject. As a consequence, here we do not consider the derivation as reduction machine, but rather as a tool for computing the number of reduction steps.

To this aim we start by introducing a few necessary notions of measure, similar to the ones of Definitions 22 and 16:

Definition 28 (Measures)**.**

i. *The size* $|\Pi|$ *of a proof* Π *is the number of applications of rules in it.*

ii. *The size* $|\mathcal{M}|$ *of a term* $\mathcal{M}$ *is defined inductively as follows:*

$$|x| = 1; \qquad |\lambda x.\mathcal{M}| = |\mathcal{M}| + 1; \qquad |\mathcal{M}\mathcal{N}| = |\mathcal{M}| + |\mathcal{N}| + 1.$$

iii. *The rank of a multiplexor of domain* $\mathcal{X}$ *is the cardinality of the set* $\mathcal{X} \cap \mathrm{FV}(\mathcal{M})$*. If* r *is the maximum rank of an application of rule* (m) *in* Π*, then the rank* $\mathtt{rk}(\Pi)$ *of* Π *is the maximum between* 1 *and* r*.*

iv. *The degree of a derivation* Π*, denoted by* $\mathtt{d}(\Pi)$*, is the maximal nesting of applications of the* $(\wedge_n)$ *rule in* Π*, namely the maximal number of applications of rule* $(\wedge_n)$ *in a path connecting the conclusion to any axiom of* Π*.*

v. *The weight* $\mathtt{W}(\Pi, r)$ *of* Π *with respect to* r *is defined inductively as follows:*

- *if* Π *ends with an application of rule* (Ax)*, then* $\mathtt{W}(\Pi, r) = 1$*;*
- *if* Π *ends with an application of rule* $(\multimap I)$ *and* Σ *is the premise of the rule, then* $\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma, r) + 1$*;*
- *if* Π *ends with an application of rule* $(\multimap E)$ *and* Σ_1, Σ_2 *are the premises of the rule, then* $\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma_1, r) + \mathtt{W}(\Sigma_2, r) + 1$*;*
- *if* Π *ends with an application of rule* $(\wedge_n)$ *and* $\Sigma_1, ..., \Sigma_n$ *are the premises of the rule, then* $\mathtt{W}(\Pi, r) = r \cdot \max_{i=1}^{n} \mathtt{W}(\Sigma_i, r)$*;*
- *if* Π *ends with an application of either rule* (w) *or rule* (m) *and* Σ *is the premise of the rule, then* $\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma, r)$*.*

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

Lemma 35 (Relations between measures)**.** *Let* $\Pi \triangleright \Gamma \vdash \mathcal{M} : \sigma$*; then:*

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

ii. $\mathtt{W}(\Pi, r) \leq r^{\mathtt{d}(\Pi)} \cdot \mathtt{W}(\Pi, 1)$.

iii. $\mathtt{W}(\Pi, 1) = |\mathcal{M}|$.

Proof. By induction on Π. Since the proof is very similar to the one of Lemma 27, we only consider the most meaningful cases.

i. Let Π be

$$\frac{\Sigma \triangleright \Gamma, x_1 : \tau_1, ..., x_n : \tau_n \vdash \mathcal{P} : \sigma}{\Gamma, x : \tau_1 \wedge ... \wedge \tau_n \vdash \mathcal{P}[x/x_i]_{i=1}^n : \sigma}\ (m)$$

where $\mathcal{M} = \mathcal{P}[x/x_i]_{i=1}^n$.
By inductive hypothesis $\mathtt{rk}(\Sigma) \leq |\mathcal{P}| \leq |\Sigma|$. Let $k \leq n$ be the cardinality of the set $\{x_1, ..., x_n\} \cap \mathrm{FV}(\mathcal{M})$. Note that $\mathtt{rk}(\Pi) = \max\{\mathtt{rk}(\Sigma), k\}$ and $k \leq |\mathcal{M}| = |\mathcal{P}|$ by definition, so

- if $\max\{\mathtt{rk}(\Sigma), k)\} = \mathtt{rk}(\Sigma)$, then $\mathtt{rk}(\Pi) = \mathtt{rk}(\Sigma) \leq |\mathcal{M}| \leq |\Pi|$;
- if $\max\{\mathtt{rk}(\Sigma), k)\} = k$, then $\mathtt{rk}(\Pi) = k \leq |\mathcal{M}| \leq |\Pi|$.

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

All the other cases are proved easily.

ii. Let Π end with an application of rule $(\wedge_n)$ with premises $\Pi_1 ... \Pi_n$. By inductive hypothesis $\mathtt{W}(\Pi_i, r) \leq r^{\mathtt{d}(\Pi_i)} \cdot \mathtt{W}(\Pi_i, 1)$ for $1 \leq i \leq n$, so in particular $\max_{i=1}^n \mathtt{W}(\Pi_i, r) \leq r^{\max_{i=1}^n \mathtt{d}(\Pi_i)} \cdot \max_{i=1}^n \mathtt{W}(\Pi_i, 1)$: then

$$\begin{array}{rcl} \mathtt{W}(\Pi, r) & = & r \cdot \max_{i=1}^n \mathtt{W}(\Pi_i, r) \\ & \leq & r \cdot r^{\max_{i=1}^n \mathtt{d}(\Pi_i)} \cdot \max_{i=1}^n \mathtt{W}(\Pi_i, 1) \\ & = & r^{\max_{i=1}^n \mathtt{d}(\Sigma_i)+1} \cdot \max_{i=1}^n \mathtt{W}(\Pi_i, 1) \\ & \leq & r^{\mathtt{d}(\Pi)} \cdot \mathtt{W}(\Pi, 1) \end{array}$$

All the other cases are proved easily.

iii. Let Π end with an application of rule $(\wedge_n)$ with premises $\Pi_1 ... \Pi_n$. By inductive hypothesis $\mathtt{W}(\Pi_i, 1) = |\mathcal{M}|$, for $1 \leq i \leq n$: then $\mathtt{W}(\Pi, 1) = \max_{i=1}^n \mathtt{W}(\Pi_i, 1) = |\mathcal{M}|$.

All the other cases are proved easily.

□

By employing the previously introduced measures, we can prove the following weighted version of Lemma 33:

Lemma 36 (Weighted substitution)**.** *Let* $\Pi \triangleright \Gamma, x : \sigma \vdash \mathcal{M} : \tau$ *and* $\Sigma \triangleright \Delta \vdash \mathcal{N} : \sigma$*, such that* $\Gamma \# \Delta$ *and* $x \notin \mathrm{dom}(\Delta)$*; then* $\Phi \triangleright \Gamma, \Delta \vdash \mathcal{M}[\mathcal{N}/x] : \tau$ *and*

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

for every $r \geq \max\{\mathtt{rk}(\Pi), \mathtt{rk}(\Sigma)\}$.

Proof. By induction on the shape of Π. Since the first part of the statement was shown to hold in the proof of Lemma 33, here we refer to the notation used in such proof in order to show that the condition on the measure is true.

If Π ends with an application of a constructive rule, then the proof is easy; therefore we consider the cases of (w), (m) and $(\wedge_n)$.

Let Π end with an application of rule (w) with premise Π'. If the range of the application oi $\{y\}$ such that $y \neq x$, then the proof is trivial. Otherwise $\mathtt{W}(\Phi, r) = \mathtt{W}(\Phi', r) = \mathtt{W}(\Pi, r)$, since the renaming sequence does not contribute to the weight, so the inequality is satisfied[1].

Let Π end with an application of rule (m) with premise Π' and domain $\{x_1, ..., x_n\}$. For simplicity, we can safely assume that $\{x_1, \dots, x_k\} = \mathrm{FV}(\mathcal{M}) \cap \{x_1, \dots, x_n\}$; moreover $\mathtt{W}(\Sigma, r) = r \cdot \max_{i=1}^{n} \mathtt{W}(\Sigma_i, r) = r \cdot \max_{i=1}^{n} \mathtt{W}(\Sigma'_i, r)$, since the renaming of variables does not change the structure, and thus the weight, of a derivation.
By inductive hypothesis $\mathtt{W}(\Phi_1, r) \leq \mathtt{W}(\Sigma'_1, r) + \mathtt{W}(\Pi', r)$, $\mathtt{W}(\Phi_2, r) \leq \mathtt{W}(\Sigma'_2, r) + \mathtt{W}(\Phi_1, r)$, and so on; therefore, $\mathtt{W}(\Phi_k, r) \leq \mathtt{W}(\Sigma'_k, r) + \mathtt{W}(\Phi_{k-1}, r)$.
Note that both ρ and δ, being renaming sequences, do not contribute to the weight: then

$$\begin{array}{rcl} \mathtt{W}(\Phi, r) & = & \mathtt{W}(\Phi_k, r) \\ & \leq & \sum_{i=1}^{k} \mathtt{W}(\Sigma'_i, r) + \mathtt{W}(\Pi', r) \\ & \leq & k \cdot \max_{i=1}^{k} \mathtt{W}(\Sigma'_i, r) + \mathtt{W}(\Pi', r) \\ & = & \mathtt{W}(\Sigma, r) + \mathtt{W}(\Pi, r) \end{array}$$

so the inequality is satisfied.

Let Π end with an application of rule $(\wedge_n)$, with $n > 1$ and premises $\Pi_1 ... \Pi_n$. By inductive hypothesis $\mathtt{W}(\Phi_i, r) \leq \mathtt{W}(\Sigma_i, r) + \mathtt{W}(\Pi_i, r)$ for $1 \leq i \leq n$: then

$$\begin{array}{rcl} \mathtt{W}(S(\Sigma, \Pi), r) & = & r \cdot \max_{i=1}^{n} \mathtt{W}(S(\Sigma_i, \Pi_i), r) \\ & \leq & r \cdot \max_{i=1}^{n} \mathtt{W}(\Sigma_i, r) + r \cdot \max_{i=1}^{n} \mathtt{W}(\Pi_i, r) \\ & = & \mathtt{W}(\Sigma, r) + \mathtt{W}(\Pi, r) \end{array}$$

so the inequality is satisfied.

□

[1] Here the non-associativity of the intersection comes into play: indeed, if intersection were to be considered as an associative operator, then Property 18.i would not hold; therefore, we would not be able to use the inductive hypothesis in order to prove the desired inequality.

Since STI is in (mostly) natural deduction style, a *detour* can be defined as an application of rule $(\multimap I)$, possibly followed by a renaming sequence, acting as the left premise of an application of rule $(\multimap E)$: in this case, the detour can be erased by the usual procedure, simply by delaying the renaming sequence and then by performing the substitution.
The weighted substitution property states that the overall weight of a derivation decreases when a detour-elimination step is performed; since a β-reduction step corresponds to $n \geq 1$ detour-elimination steps, the following result holds:

Theorem 21 (Weighted subject reduction)**.** $\Pi \triangleright \Gamma \vdash \mathcal{M} : \sigma$ *and* $\mathcal{M} \to \mathcal{M}'$ *imply* $\Pi' \triangleright \Gamma \vdash \mathcal{M}' : \sigma$ *and* $\mathtt{W}(\Pi', r) < \mathtt{W}(\Pi, r)$*, for every* $r \geq \mathtt{rk}(\Pi)$.

Proof. Consider the context $\mathcal{C}$ such that $\mathcal{M} = \mathcal{C}[(\lambda x.\mathcal{Q})\mathcal{N}]$ and $\mathcal{M}' = \mathcal{C}[\mathcal{N}[\mathcal{Q}/x]]$; we proceed by induction on $\mathcal{C}$.
Since the first part of the statement was already shown to hold, we refer to the notation used in the proof of Theorem 20 with the aim of proving the remaining inequality.

Let $\mathcal{C} = \Box$, so $\mathcal{M} = (\lambda x.\mathcal{Q})\mathcal{N}$.
Note that, because Π' is obtained from Π by replacing every subproof Π_i with Π'_i, for $1 \leq i \leq n$, the intersection tree connecting all the subproofs is left unchanged. By Lemma 36 $\mathtt{W}(\Pi'_i, r) \leq \mathtt{W}(\Sigma''_i, r) + \mathtt{W}(\Sigma'_i, r)$, for every $r \geq \mathtt{rk}(\Pi)$: then, since $\mathtt{W}(\Pi'_i, r) < \mathtt{W}(\Sigma''_i, r) + \mathtt{W}(\Sigma'_i, r) + 2 = \mathtt{W}(\Pi'_i, r)$, the inequality holds.

Again, the cases of $\mathcal{C} \neq \Box$ follow easily by induction.

□

Now we have all the necessary ingredients for proving that both the number of normalization steps and the size of the normal form are bounded by a function of the size of the term:

Theorem 22 (Measure of reduction)**.** *Let* $\Pi \triangleright \Gamma \vdash \mathcal{M} : \sigma$ *and let* $\mathcal{M}$ *β-reduce to* $\mathcal{M}'$ *in* n *steps; then:*

i. $n < |\mathcal{M}|^{\mathtt{d}(\Pi)+1}$;

ii. $|\mathcal{M}'| < |\mathcal{M}|^{\mathtt{d}(\Pi)+1}$.

Proof.

Let $\mathcal{M} \to \mathcal{M}_1 \to \ldots \to \mathcal{M}_n = \mathcal{M}'$ and $r = \mathtt{rk}(\Pi)$.
By Lemma 35, $r \leq |\mathcal{M}|$ and $\mathtt{W}(\Pi, r) \leq r^{\mathtt{d}(\Pi)} \cdot \mathtt{W}(\Pi, 1) = r^{\mathtt{d}(\Pi)} \cdot |\mathcal{M}| \leq |\mathcal{M}|^{\mathtt{d}(\Pi)} \cdot |\mathcal{M}|$.
By Theorem 21, if Π rewrites to Π_1 in 1 step then $\mathtt{W}(\Pi_1, r) \leq \mathtt{W}(\Pi, r) - 1$; by the same Lemma, if Π_1 rewrites to Π_2 in 1 step then $\mathtt{W}(\Pi_2, r) \leq \mathtt{W}(\Pi_1, r) - 1 \leq \mathtt{W}(\Pi, r) - 2$, and so on: then $\mathtt{W}(\Pi', r) \leq \mathtt{W}(\Pi, r) - n$ and $n < \mathtt{W}(\Pi, r)$ after n $\multimap$-normalization

steps. Since the rank of a derivation never increases during reduction, by substituting in the above expression we obtain $n < \mathtt{W}(\Pi, r) \leq |\mathcal{M}|^{\mathtt{d}(\Pi)+1}$.

ii. By Lemma 35, $|\mathcal{M}_i| = \mathtt{W}(\Pi_i, 1)$ for all i. Since obviously $\mathtt{W}(\Pi', 1) \leq \mathtt{W}(\Pi', r)$ and by Lemma 35 $|\mathcal{M}'| = \mathtt{W}(\Pi', 1)$, we obtain $|\mathcal{M}'| \leq \mathtt{W}(\Pi', r) < \mathtt{W}(\Pi, r) \leq r^{\mathtt{d}(\Pi)} \cdot \mathtt{W}(\Pi, 1) = r^{\mathtt{d}(\Pi)} \cdot |\mathcal{M}| \leq |\mathcal{M}|^{\mathtt{d}(\Pi)+1}$ by applying Theorem 21.

□

Note that the exponent of such function is, in general, dependent on the term; for this reason, the bound on the normalization procedure can easily become exponential with respect to the size of the term. Nevertheless, the proof given above is independent on the reduction strategy: therefore, the result given above also shows that all terms typed in STI are strongly normalizing.

One might wonder why we choose to employ the n-ary intersection, instead of the more standard binary connective, since the use of the intersection guarantees that the typability power of the system is the same in both cases. To answer such question, let us remark that our interest lies not only in the typability, but also in using derivations for measuring the complexity of the reduction.
As an example, consider the term $\mathcal{M} = (\lambda x.\lambda y.yx...x)(II)$, where x occurs n times and I represents the identity function. By the definition of size, $|\mathcal{M}| = 2n + 6$; note that a minimal derivation typing $\mathcal{M}$ in STI has depth 1 and rank n:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{x_1 : \mathtt{a} \multimap \mathtt{a}, ..., x_n : \mathtt{a} \multimap \mathtt{a} \vdash \lambda y.yx_1...x_n : \mathtt{A}_n \multimap \mathtt{b}}\,(m)}{x : \sigma_n \vdash \lambda y.yx...x : \mathtt{A}_n \multimap \mathtt{b}}\,(\multimap I)}{\vdash \lambda x.\lambda y.yx...x : \sigma_n \multimap \mathtt{A}_n \multimap \mathtt{b}} \qquad \dfrac{\dfrac{\vdots}{(\vdash II : \mathtt{a} \multimap \mathtt{a})_{1 \leq i \leq n}}}{\vdash II : \sigma_n}\,(\wedge_n)}{\vdash (\lambda x.\lambda y.yx...x)(II) :}\,(\multimap E)$$

where $\sigma_n = (\mathtt{a} \multimap \mathtt{a}) \wedge ... \wedge (\mathtt{a} \multimap \mathtt{a})$, $\mathtt{A}_n = (\mathtt{a} \multimap \mathtt{a}) \multimap ... \multimap (\mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{b}$ and $\mathtt{a} \multimap \mathtt{a}$ occurs n times in both σ and $\mathtt{A}$. Then the resulting bound for the number of β-reduction steps is $(2n + 6)$, while the effective number of reductions is at most $2n + 1$.

Let us now consider an alternative version of the system, where the intersection is considered to be a binary connective; in this case, the multiplexor rule (m) has a fixed rank of 2 and the rule $(\wedge_n)$ becomes $(\wedge_2)$, so the depth of the minimal derivation is $n - 1$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{x_1 : \sigma_{n-1}, x_2 : \mathtt{a} \multimap \mathtt{a} \vdash \lambda y.yx_1...x_1x_2 : \mathtt{A}_n \multimap \mathtt{b}}\,(m)}{x : \sigma_n \vdash \lambda y.yx...x : \mathtt{A}_n \multimap \mathtt{b}}\,(\multimap I)}{\vdash \lambda x.\lambda y.yx...x : \sigma_n \multimap \mathtt{A}_n \multimap \mathtt{b}} \qquad \dfrac{\vdots \quad \vdots}{\vdash II : \sigma_{n-1} \quad \vdash II : \mathtt{a} \multimap \mathtt{a}}\,(\wedge_2)}{\vdash II : \sigma_n}\,(\multimap E)}{\vdash (\lambda x.\lambda y.yx...x)(II) :}$$

where $\sigma_n = (...((\mathtt{a} \multimap \mathtt{a}) \wedge (\mathtt{a} \multimap \mathtt{a})) \wedge ...) \wedge (\mathtt{a} \multimap \mathtt{a})$, $\mathtt{A}_n = (\mathtt{a} \multimap \mathtt{a}) \multimap ... \multimap (\mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{b}$ and $\mathtt{a} \multimap \mathtt{a}$ occurs n times in both σ and $\mathtt{A}$. Tehn the resulting rough bound is $(2n + 6)^n$, which is exponential in the size of the initial term.

This example shows that the choice of using the n-ary intersection allows us to obtain a more refined bound with respect to the one obtained by opting for the binary connective.

4.3 Characterization of strong normalization

In the present section we show that system $\mathtt{STI}$ characterizes strongly normalizing terms, namely that $\mathcal{M}$ is typable in $\mathtt{STI}$ if and only if $\mathcal{M}$ is strongly normalizing.

The first implication is obtained as a byproduct of the complexity bound:

Lemma 37 (Typability implies strong normalization)**.** *If $\Gamma \vdash \mathcal{M} : \sigma$, then $\mathcal{M}$ is strongly normalizing.*

Proof. By Theorem 22.i, the number of reductions steps is bounded by a finite number.

□

It is easy to check that the size of Π, defined simply as the number of applications of rules in it, decreases with each reduction step regardless of the reduction strategy we adopt; this depends essentially on the use of non-idempotent intersection types, whose main advantage is the fact that the size of a derivation reflects the size of the normal form of its subject, rather than the size of the actual term it gives type to. This ultimately means that all the copies of a subterm, which might be duplicated during some reduction step, are already laid out in the initial derivation: this property, which is inherent of non-idempotent intersection, ensures that the size of the derivation never increases.

As for the right-to-left implication, we employ the same technique of Section 3.3.2 by adapting it to system $\mathtt{STI}$.

The first requirement is that every normal form can be assigned a linear type; such statement holds by Property 19.

We then prove that the typability of a term is preserved under substitution:

Lemma 38 (Inverted substitution)**.** *Let $\Phi \triangleright \Gamma, \Delta \vdash \mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^{n} : \tau$, where $\mathcal{N}_1, ..., \mathcal{N}_n$ are typable in $\mathtt{STI}$, $\mathrm{dom}(\Gamma) = \mathrm{FV}(\mathcal{M}) \setminus \{x_1, ..., x_n\}$ and $\{x_1, ..., x_n\} \cap \mathrm{dom}(\Delta) = \emptyset$; then there are $\Delta_i \vdash \mathcal{N}_i : \sigma_i$, for $1 \leq i \leq n$, such that $\Gamma, x_1 : \sigma_1, .., x_n : \sigma_n \vdash \mathcal{M} : \tau$.*

Proof. By induction on the shape of Φ.

Let Φ be

$$\frac{}{y : \mathtt{B} \vdash y : \mathtt{B}} \ (Ax)$$

so $\mathcal{N} = y$ and $\Gamma = \emptyset$: then Π is

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

and Σ is the same derivation as Φ, where $\sigma = \tau = \mathtt{B}$ and $\mathcal{M}[\mathcal{N}/x] = x[y/x] = y$.

Let Φ end with an application of rule (w): then the proof follows easily by induction.

Let Φ be

$$\frac{\Phi' \rhd \Gamma, \Delta, y : \rho \vdash \mathcal{P} : \mathtt{A}}{\Gamma, \Delta \vdash \lambda y.\mathcal{P} : \rho \multimap \mathtt{A}} \ (\multimap I)$$

where $y \notin \mathrm{FV}(\mathcal{N})$.
Let $\mathcal{M} = \lambda y.\mathcal{Q}$, where $x \in \mathrm{FV}(\mathcal{P})$, so that $\mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n = \lambda y.\mathcal{P}$ and $\mathcal{P} = \mathcal{Q}[\mathcal{N}_i/x_i]_{i=1}^n$: by inductive hypothesis $\Phi' \rhd \Gamma, \Delta, y : \rho \vdash \mathcal{Q}[\mathcal{N}_i/x_i]_{i=1}^n : \mathtt{A}$ implies there are $\Sigma_i \rhd \Delta_i \vdash \mathcal{N}_i : \tau_i$, for $1 \leq i \leq n$, such that $\Pi' \rhd \Gamma, x_1 : \tau_1, ..., x_n : \tau_n, y : \rho \vdash \mathcal{Q} : \mathtt{A}$. Then Π is obtained by applying rule $(\multimap I)$ to Π' in order to abstract over y.
Let $\mathcal{M} = x$, so that $\mathcal{M}[\mathcal{N}/x] = x[\mathcal{N}/x] = \mathcal{N}$ and $\mathcal{N} = \lambda y.\mathcal{P}$: then Π is

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

and Σ is the same derivation as Φ.

Let Φ be

$$\frac{\Phi' \rhd \Gamma', \Delta' \vdash \mathcal{P} : \rho \multimap \mathtt{A} \quad \Phi'' \rhd \Gamma'', \Delta'' \vdash \mathcal{Q} : \rho}{\Gamma, \Delta \vdash \mathcal{P}\mathcal{Q} : \mathtt{A}} \ (\multimap E)$$

where $\Gamma = \Gamma', \Gamma''$ and $\Delta = \Delta', \Delta''$.
Let $\mathcal{M} = \mathcal{P}'\mathcal{Q}'$, so that $\mathcal{P} = \mathcal{P}'[\mathcal{N}_i/x_i]_{i=1}^n$ and $\mathcal{Q} = \mathcal{Q}'[\mathcal{N}_i/x_i]_{i=1}^n$: by inductive hypothesis $\Phi' \rhd \Gamma', \Delta' \vdash \mathcal{P}'[\mathcal{N}_i/x_i]_{i=1}^n : \rho \multimap \mathtt{A}$ and $\Phi'' \rhd \Gamma'', \Delta'' \vdash \mathcal{Q}'[\mathcal{N}_i/x_i]_{i=1}^n : \rho$ imply there are derivations $\Sigma_i' \rhd \Delta_i' \vdash \mathcal{N}_i : \tau_i'$ and $\Sigma_i'' \rhd \Delta_i'' \vdash \mathcal{N}_i : \tau_i''$, for $1 \leq i \leq n$, such that both $\Pi' \rhd \Gamma', x_1 : \tau_1', ..., x_n : \tau_n' \vdash \mathcal{P}' : \rho \multimap \mathtt{A}$ and $\Pi'' \rhd \Gamma'', x_1 : \tau_1'', ..., x_n : \tau_n'' \vdash \mathcal{Q}' : \rho$. By applying rule $(\wedge_2)$ to Σ_i' and Σ_i'', we obtain $\Sigma_i \rhd \Delta_i \vdash \mathcal{N}_i : \tau_i$, where $\Delta_i = \Delta_i' \wedge \Delta_i''$ and $\tau_i = \tau_i' \wedge \tau_i''$, for $1 \leq i \leq n$. Consider now disjoint derivations Ψ' and Ψ'', respectively copies of Π' and Π'', such that $\Psi' \rhd \Gamma', x_1' : \tau_1', ..., x_n' : \tau_n' \vdash \mathcal{P}'[x_i'/x_i]_{i=1}^n : \rho \multimap \mathtt{A}$ and $\Psi'' \rhd \Gamma'', x_1'' : \tau_1'', ..., x_n'' : \tau_n'' \vdash \mathcal{Q}'[x_i''/x_i]_{i=1}^n : \rho$: then we can build

$$\cfrac{\cfrac{\Psi' \quad \Psi''}{\Gamma', \Gamma'', x_1' : \tau_1', ..., x_n' : \tau_n', x_1'' : \tau_1'', ..., x_n'' : \tau_n'' \vdash \mathcal{P}'[x_i'/x_i]_{i=1}^n \mathcal{Q}'[x_i''/x_i]_{i=1}^n : \mathtt{A}} \ (\multimap E)}{\Gamma', \Gamma'', x_1 : \tau_1, ..., x_n : \tau_n \vdash \mathcal{P}'\mathcal{Q}' : \mathtt{A}} \ (m)$$

Let $\mathcal{M} = x$, so that $\mathcal{M}[\mathcal{N}/x] = x[\mathcal{N}/x] = \mathcal{N}$ and $\mathcal{N} = \mathcal{P}\mathcal{Q}$: then Π is

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

and Σ is the same derivation as Φ.

Let Φ be

$$\frac{\Gamma, \Delta, y_1 : \rho_1, ..., y_m : \rho_m : \tau_m \vdash \mathcal{P} : \tau}{\Gamma, \Delta, y : \rho_1 \wedge ... \wedge \rho_m \vdash \mathcal{P}[y/y_j]_{j=1}^m : \tau} \ (m)$$

where $\mathcal{P}[y/y_j]_{j=1}^m = \mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n$.
Let $y \notin \cup_{i=1}^n \mathrm{FV}(\mathcal{N}_i)$, so $\mathcal{M} = \mathcal{M}'[y/y_j]_{j=1}^m$ and $\mathcal{P} = \mathcal{M}'[\mathcal{N}_i/x_i]_{i=1}^n$. If $y \notin \mathrm{FV}(\mathcal{M})$, then $\mathcal{P} = \mathcal{P}[y/y_j]_{j=1}^m = \mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n$ and the proof comes by induction. Otherwise, $\{y_{s_1}, ..., y_{s_p}\} = \{y_1, ..., y_m\} \cap \mathrm{FV}(\mathcal{M})$ and by inductive hypothesis there are $\Sigma_i \triangleright \Delta_i \vdash \mathcal{N}_i : \sigma_i$ such that $\Pi' \triangleright \Gamma, y_{s_1} : \rho_{s_1}, ..., y_{s_p} : \rho_{s_p}, x_1 : \sigma_1, ..., x_n : \sigma_n \vdash \mathcal{M} : \tau$. Then the result is obtained from Π' by a suitable renaming sequence and by one application of rule (m) with domain $\{y_1, ..., y_n\}$ and range $\{y\}$.
Now let us consider $y \in \cup_{i=1}^n \mathrm{FV}(\mathcal{N}_i)$, so $\mathcal{M} = \mathcal{M}'[y/y_j]_{j=1}^m[x_i/x_i^1, ..., x_i/x_i^{r_i}]_{i=1}^n$ and $\mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n = (\mathcal{M}'[\mathcal{N}_i^1/x_i^1, ..., \mathcal{N}_i^{n_i}/x_i^{r_i}]_{i=1}^n)[y/y_j]_{j=1}^m$.
Let $\{y_{s_1}, ..., y_{s_p}\} = \{y_1, ..., y_m\} \cap \mathrm{FV}(\mathcal{M}')$. Since $\mathcal{P} = \mathcal{M}'[\mathcal{N}_i^1/x_i^1, ..., \mathcal{N}_i^{r_i}/x_i^{r_i}]_{i=1}^n$, by inductive hypothesis there are $\Sigma_i^h \triangleright \Delta_i^h \vdash \mathcal{N}_i^h : \sigma_i^h$, for $1 \leq i \leq n$ and $1 \leq h \leq r_i$, such that

$$\Psi \triangleright \Gamma, y_{s_1} : \rho_{s_1}, ..., y_{s_p} : \rho_{s_p}, x_1^1 : \sigma_1^1, ..., x_1^{r_1} : \sigma_1^{r_1}, ..., x_n^1 : \sigma_n^1, ..., x_n^{r_n} : \sigma_n^{r_n} \vdash \mathcal{M}' : \tau$$

Note that $\mathcal{N}_i = \mathcal{N}_i^h[y/y_m]_{j=1}^m$, so we can build $\Delta_i'^h \vdash \mathcal{N}_i^h[y/y_m]_{j=1}^m : \sigma_i^h$ from Σ_i^h by applying rule (m) with domain $\mathrm{FV}(\mathcal{N}_i^h) \cap \{y_1, ..., y_m\}$ and range $\{y\}$, for $1 \leq i \leq n$ and $1 \leq h \leq r_i$. Then we can build Σ_i as

$$\frac{(\Delta_i'^h \vdash \mathcal{N}_i^h[y/y_m]_{j=1}^m : \sigma_i^h)_{1 \leq h \leq r_i}}{\Delta_i \vdash \mathcal{N}_i : \sigma_i} \ (\wedge_{r_i})$$

where $\sigma_i = \sigma_i^1 \wedge ... \wedge \sigma_i^{r_i}$, for every $1 \leq i \leq n$.
Let δ be a renaming sequence containing n applications of rule (m), the i-th one having domain $\{x_i^1, ..., x_i^{r_i}\}$ and range $\{x_i\}$, for $1 \leq i \leq n$. If $y \in \mathrm{FV}(\mathcal{M})$, then from Ψ we derive Π proving

$$\Gamma, y : \rho_1 \wedge ... \wedge \rho_m, x_1 : \sigma_1, ..., x_n : \sigma_n \vdash \mathcal{M}'[y/y_j]_{j=1}^m[x_i/x_i^1, ..., x_i/x_i^{r_i}]_{i=1}^n : \tau$$

by a suitable renaming sequence and by one application of rule (m) with domain $\{y_1, ..., y_n\}$ and range $\{y\}$, followed by sequence δ. Otherwise, if $y \notin \mathrm{FV}(\mathcal{M})$, then $\mathcal{M}' = \mathcal{M}$ and $\{y_{s_1}, ..., y_{s_p}\} = \emptyset$, so we obtain the desired derivation Π by applying renaming sequence δ to $\Gamma, x_1^1 : \sigma_1^1, ..., x_1^{r_1} : \sigma_1^{r_1}, ..., x_n^1 : \sigma_n^1, ..., x_n^{r_n} : \sigma_n^{r_n} \vdash \mathcal{M} : \tau$.

Let Φ be

$$\frac{(\Gamma_j, \Theta_j \vdash \mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n : \tau_j)_{1 \leq j \leq m}}{\Gamma, \Delta \vdash \mathcal{M}[\mathcal{N}_i/x_i]_{i=1}^n : \tau} \ (\wedge_m)$$

where $\Gamma = \wedge_{j=1}^m \Gamma_j$ and $\tau = \tau_1 \wedge ... \wedge \tau_m$.
For $1 \leq j \leq m$, by inductive hypothesis there are $\Sigma_i^j \triangleright \Delta_i^j \vdash \mathcal{N}_i : \sigma_i^j$, for $1 \leq i \leq n$, such that $\Pi_j \triangleright \Gamma_j, x_1 : \sigma_1^j, ..., x_n : \sigma_n^j \vdash \mathcal{M} : \tau_j$. Then we can build Σ_i as

$$\frac{(\Delta_i^j \vdash \mathcal{N}_i : \sigma_i^j)_{1 \leq j \leq m}}{\Delta_i \vdash \mathcal{N}_i : \sigma_i} \ (\wedge_m)$$

where $\sigma_i = \sigma_i^1 \wedge ... \wedge \sigma_i^m$, for $1 \leq i \leq n$, such that Π is

$$\frac{(\Gamma_j, x_1 : \sigma_1^j, ..., x_n : \sigma_n^j \vdash \mathcal{M} : \tau_j)_{1 \leq j \leq m}}{\Gamma, x_1 : \sigma_1, ..., x_n : \sigma_n \vdash \mathcal{M} : \tau} \ (\wedge_m)$$

□

The simplest case of the subject expansion property follows from the previous result:

Lemma 39 (Subject expansion under substitution)**.** *$\Theta \vdash \mathcal{M}[\mathcal{N}/x] : \sigma$ and $\mathcal{N}$ typable in* `STI` *imply there exists Θ' such that $\Theta' \vdash (\lambda x.\mathcal{M})\mathcal{N} : \sigma$.*

Proof. By induction on σ.

Let $\sigma = \mathtt{A}$. Consider $\Theta = \Gamma, \Xi$ where $\mathrm{dom}(\Gamma) = \mathrm{FV}(\mathcal{M}) \setminus \{x\}$. By Lemma 38, there is $\Delta \vdash \mathcal{N} : \tau$ such that $\Gamma, x : \tau \vdash \mathcal{M} : \mathtt{A}$; let $\Delta' \vdash \mathcal{N}' : \tau$ be a copy of $\Delta \vdash \mathcal{N} : \tau$, such that $\Gamma \# \Delta'$: then we can build the following derivation

$$\frac{\dfrac{\Gamma, x : \tau \vdash \mathcal{M} : \mathtt{A}}{\Gamma \vdash \lambda x.\mathcal{M} : \tau \multimap \mathtt{A}} \ (\multimap I) \qquad \Delta' \vdash \mathcal{N}' : \tau}{\Gamma, \Delta' \vdash (\lambda x.\mathcal{M})\mathcal{N}' : \mathtt{A}} \ (\multimap E)$$

and, since $(\lambda x.\mathcal{M})\mathcal{N}$ is an instance of $(\lambda x.\mathcal{M})\mathcal{N}'$, the result follows by the application of a suitable renaming sequence.
Otherwise, let $\sigma = \sigma_1 \wedge ... \wedge \sigma_n$. By Property 18, $\Pi \triangleright \Gamma \vdash \mathcal{M}[\mathcal{N}/x] : \sigma_1 \wedge ... \wedge \sigma_n$ implies there are derivations $\Pi_i \triangleright \Gamma_i \vdash \mathcal{P} : \sigma_i$ $(1 \leq i \leq n)$ such that Π is obtained by an application of rule $(\wedge_n)$ to $(\Pi_i)_{i=1}^n$, followed by a renaming sequence δ. By applying sequence δ to Π_i we obtain $\Gamma_i'' \vdash \mathcal{M}[\mathcal{N}/x] : \sigma_i$, for $1 \leq i \leq n$.
By inductive hypothesis there are Γ_i' such that $\Gamma_i' \vdash (\lambda x.\mathcal{M})\mathcal{N} : \sigma_i$, for $1 \leq i \leq n$: then by applying rule $(\wedge_n)$ to such derivations we obtain $\wedge_{i=1}^n \Gamma_i' \vdash (\lambda x.\mathcal{M})\mathcal{N} : \sigma$.

□

Now we are able to prove that all strongly normalizing terms are typable in `STI`:

Theorem 23 (Strong normalization implies typability)**.** *If $\mathcal{M}$ is strongly normalizing, then $\mathcal{M}$ is typable in* `STI`*.*

Proof. For each of the three rules of Table 3.3 we show that, if the premises of the rule are typable in `STI`, then the conclusion is typable in `STI` as well.

The proof, analogous to the one of Theorem 23, follows from Property 19 and by Lemma 39.

□

Finally the desired result follows:

Property 20 (Strong normalization)**.** $\Pi \triangleright \Gamma \vdash \mathcal{M} : \sigma$ *if and only if* $\mathcal{M}$ *is strongly normalizing.*

Proof. Easy by combining Lemma 37 and Theorem 23.

□

Chapter 5

Conclusion and future developments

In this thesis we examined a few different, somehow orthogonal, approaches to the ICC setting. On one level, we considered both a polyvalent and a monovalent characterization of complexity classes through type assignment systems for (variants of) λ-calculus. On another level, we considered both a typing system based on light logics (ELL) and one, not having a logical counterpart but inspired by it (SLL), using a slight variation of non-associative intersection types. The latter approach is further explored by considering non-associative and non-idempotent intersection, in order to prove some quantitative properties of λ-terms.

As a future development of this thesis, it would be interesting to check whether the approach used in Chapter 3, namely the extension of a typing system based on a light logic using intersection types, could be also applied to the system of Chapter 2: would it then be possible to build a similar intersection type system, starting from ELL instead of SLL?

If we consider the system characterizing the hierarchy of k-EXP, it should be possible to design a type assignment system for $\lambda^!$-calculus by using the same stratified types of STR, namely intersection types enjoying both commutativity and idempotence, but not associativity:

$$\begin{array}{lcl} \mathtt{A} & ::= & \mathtt{a} \mid \mathtt{S} \\ \mathtt{S} & ::= & \sigma \multimap \mathtt{A} \mid \forall \mathtt{a}.\mathtt{S} \mid \mu \mathtt{a}.\mathtt{S} \\ \sigma & ::= & \mathtt{A} \mid \{\underbrace{\sigma, ..., \sigma}_{n}\} \end{array}$$

Indeed, the idempotence property seems to be essential in the ICC setting because it allows to obtain a uniform representation of data, while some notion of non-associativity is needed in order to store information about the stratification.

$$\frac{}{\Gamma, x : \mathtt{A} \mid \Delta \mid \Theta \vdash x : \mathtt{A}}\ (Ax^L) \qquad \frac{}{\Gamma \mid \Delta \mid x : \{\mathtt{A}\}, \Theta \vdash x : \mathtt{A}}\ (Ax^P)$$

$$\frac{\Gamma, x : \mathtt{A} \mid \Delta \mid \Theta \vdash \mathcal{M} : \tau}{\Gamma \mid \Delta \mid \Theta \vdash \lambda x.\mathcal{M} : \mathtt{A} \multimap \tau}\ (\multimap I^L) \qquad \frac{\Gamma \mid \Delta, x : \sigma \mid \Theta \vdash \mathcal{M} : \tau}{\Gamma \mid \Delta \mid \Theta \vdash \lambda^! x.\mathcal{M} : \sigma \multimap \tau}\ (\multimap I^I)$$

$$\frac{\Gamma_1 \mid \Delta \mid \Theta_1 \vdash \mathcal{M} : \tau \multimap \sigma \quad \Gamma_2 \mid \Delta \mid \Theta_2 \vdash \mathcal{N} : \tau \quad \Gamma_1 \# \Gamma_2}{\Gamma_1, \Gamma_2 \mid \Delta \mid \Theta_1 \cup \Theta_2 \vdash \mathcal{M}\mathcal{N} : \sigma}\ (\multimap E)$$

$$\frac{(\emptyset \mid \emptyset \mid \Theta_i \vdash \mathcal{M} : \sigma_i)_{1 \leq i \leq n}}{\Gamma \mid \cup_{i=1}^n \Theta_i, \Delta \mid \Theta \vdash !\mathcal{M} : \{\sigma_1, \dots, \sigma_n\}}\ (st)$$

Table 5.1: Derivation rules for typed $\lambda^!$-calculus with stratified types.

Both the modal and the parking context could be defined as partial functions assigning stratified types to variables, because then we would be able to assign different types to the same variable. The rules of the system (minus the quantifier and fix-point rules) would be defined as in Table 5.1.
Note that rule $(\multimap E)$ builds the parking context of its conclusion by taking the union of the parking contexts of its premises, since all variables in the parking context are assigned a stratified type.

Based on the previous definitions, booleans and Scott binary words would then have the usual datatype, while Church integers and binary words would be typable with $\mathrm{N} = \forall \mathtt{a}.\{\mathtt{a} \multimap \mathtt{a}\} \multimap \{\mathtt{a} \multimap \mathtt{a}\}$ and $\mathrm{W} = \forall \mathtt{a}.\{\mathtt{a} \multimap \mathtt{a}\} \multimap \{\mathtt{a} \multimap \mathtt{a}\} \multimap \{\mathtt{a} \multimap \mathtt{a}\}$ respectively.

Let the integer $\mathtt{d}(\sigma)$ and the set $s(\sigma)$ be defined inductively as follows:

$$\mathtt{d}(\mathtt{A}) = 0, \qquad \mathtt{d}(\{\sigma_1, \dots, \sigma_n\}) = \max_{i=1}^{n} \mathtt{d}(\sigma_i) + 1;$$

$$s(\mathtt{A}) = \{\mathtt{A}\}, \qquad s(\{\sigma_1, \dots, \sigma_n\}) = \cup_{i=1}^n s(\sigma_i).$$

For the characterization, one would then expect predicates corresponding to `k-EXP` to have the type $\{\mathrm{W}\} \multimap \sigma$, where $\mathtt{d}(\sigma) = k + 2$ and $s(\sigma) = \{\mathrm{B}\}$; similarly, functions would have the type $\{\mathrm{W}\} \multimap \tau$, where $\mathtt{d}(\tau) = k + 2$ and $s(\tau) = \{\mathrm{W}_S\}$.

The previous construction comes easily because of the restricted (!) rule we consider in Chapter 2. If we consider instead a system like the one of [CDR08], assigning `ELL` types to pure λ-calculus, then the issue of associativity for intersection becomes more tricky, because the rule of stratification does not have any restriction

on the contexts of its premises and so the use of stratified types (as defined for STR) causes subject reduction to fail.

This issue can be avoided by considering a milder notion of associativity, where the only information we care to retain is the distribution of linear types over different strata. The latter approach is being considered, in order to give a bound on the reduction of terms of call-by-value λ-calculus. Here we use stratified types in the guise of intersection types enjoying both commutativity and a limited notion of associativity; an intuitive way to explain the latter is to think of these stratified types as finite lists of (possibly empty) multisets of linear types, where the position of a multiset in the list (namely its index) represents the depth of the linear types belonging to it.

While the notion of normal form is not as interesting in this setting, we believe that the power of intersection types should be enough to capture all solvable terms, namely all terms normalizable with leftmost reduction to their head normal form.

Appendix A

Additional proofs

A.1 Depth-wise confluence

In this section we prove Property 1.i, namely that $\lambda^!$-calculus is depth-wise confluent, by following the proof given in [RP04].

First we define the determinist parallel reduction-by-level, which reduces all redexes of a term at depth i in a single step; then we introduce a non-deterministic parallel reduction-by-level, which reduces only a subset of all redexes present at depth i in a term.

Definition 29 (Reduction $\hookrightarrow_i$)**.** *The deterministic parallel reduction-by level at depth i, denoted by $\hookrightarrow_i$, is defined inductively as follows:*

- $x \hookrightarrow_i x$;
- $\mathcal{N} \hookrightarrow_i \mathcal{N}'$ *implies* $\lambda x.\mathcal{N} \hookrightarrow_i \lambda x.\mathcal{N}'$;
- $\mathcal{N} \hookrightarrow_i \mathcal{N}'$ *implies* $\lambda^! x.\mathcal{N} \hookrightarrow_i \lambda^! x.\mathcal{N}'$;
- $!\mathcal{N} \hookrightarrow_0 !\mathcal{N}$, *while* $\mathcal{N} \hookrightarrow_i \mathcal{N}'$ *implies* $!\mathcal{N} \hookrightarrow_{i+1} !\mathcal{N}'$;
- *if* $\mathcal{P}\mathcal{N}$ *is not a redex,* $\mathcal{P} \hookrightarrow_i \mathcal{P}'$ *and* $\mathcal{N} \hookrightarrow_i \mathcal{N}'$ *imply* $\mathcal{P}\mathcal{N} \hookrightarrow_i \mathcal{P}'\mathcal{N}'$;
- $\mathcal{P} \hookrightarrow_0 \mathcal{P}'$ *and* $\mathcal{N} \hookrightarrow_0 \mathcal{N}'$ *imply* $(\lambda x.\mathcal{P})\mathcal{N} \hookrightarrow_0 \mathcal{P}'[\mathcal{N}'/x]$, *while* $\mathcal{P} \hookrightarrow_i \mathcal{P}'$ *and* $\mathcal{N} \hookrightarrow_i \mathcal{N}'$ *imply* $(\lambda x.\mathcal{P})\mathcal{N} \hookrightarrow_i (\lambda x.\mathcal{P}')\mathcal{N}'$ *for* $i \geq 1$;
- $\mathcal{P} \hookrightarrow_0 \mathcal{P}'$ *implies* $(\lambda^! x.\mathcal{P})!\mathcal{N} \hookrightarrow_0 \mathcal{P}'[\mathcal{N}/x]$, *while* $\mathcal{P} \hookrightarrow_i \mathcal{P}'$ *and* $!\mathcal{N} \hookrightarrow_i !\mathcal{N}'$ *imply* $(\lambda^! x.\mathcal{P})!\mathcal{N} \hookrightarrow_i (\lambda^! x.\mathcal{P}')!\mathcal{N}'$ *for* $i \geq 1$.

We denote by $[\mathcal{M}]_i$ the term such that $\mathcal{M} \hookrightarrow_i [\mathcal{M}]_i$, also called the *complete development* of $\mathcal{M}$ at depth i. It is easy to see that for each term $\mathcal{M}$ there is only one $[\mathcal{M}]_i$:

Proposition 2 (Complete development at depth i)**.** *$\mathcal{M} \hookrightarrow_i \mathcal{P}$ and $\mathcal{M} \hookrightarrow_i \mathcal{Q}$ imply $\mathcal{P} = \mathcal{Q} = [\mathcal{M}]_i$.*

Proof. Trivial, since the reduction is deterministic. □

Definition 30 (Reduction $\twoheadrightarrow_i$)**.** *The non-deterministic parallel reduction-by-level, denoted by $\twoheadrightarrow_i$, is defined inductively as follows:*

- *$x \twoheadrightarrow_i x$;*
- *$\mathcal{N} \twoheadrightarrow_i \mathcal{N}'$ implies $\lambda x.\mathcal{N} \twoheadrightarrow_i \lambda x.\mathcal{N}'$;*
- *$\mathcal{N} \twoheadrightarrow_i \mathcal{N}'$ implies $\lambda^! x.\mathcal{N} \twoheadrightarrow_i \lambda^! x.\mathcal{N}'$;*
- *$!\mathcal{N} \twoheadrightarrow_0 !\mathcal{N}$, while $\mathcal{N} \twoheadrightarrow_{i-1} \mathcal{N}'$ implies $!\mathcal{N} \twoheadrightarrow_i !\mathcal{N}'$ for $i \geq 1$;*
- *$\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ and $\mathcal{N} \twoheadrightarrow_i \mathcal{N}'$ imply $\mathcal{P}\mathcal{N} \twoheadrightarrow_i \mathcal{P}'\mathcal{N}'$;*
- *$\mathcal{P} \twoheadrightarrow_0 \mathcal{P}'$ and $\mathcal{N} \twoheadrightarrow_0 \mathcal{N}'$ imply $(\lambda x.\mathcal{P})\mathcal{N} \twoheadrightarrow_0 \mathcal{P}'[\mathcal{N}'/x]$, while $\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ and $\mathcal{N} \twoheadrightarrow_i \mathcal{N}'$ imply $(\lambda x.\mathcal{P})\mathcal{N} \twoheadrightarrow_i (\lambda x.\mathcal{P}')\mathcal{N}'$ for $i \geq 1$;*
- *$\mathcal{P} \twoheadrightarrow_0 \mathcal{P}'$ implies $(\lambda^! x.\mathcal{P})!\mathcal{N} \twoheadrightarrow_0 \mathcal{P}'[\mathcal{N}/x]$, while $\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ and $!\mathcal{N} \twoheadrightarrow_i !\mathcal{N}'$ imply $(\lambda^! x.\mathcal{P})!\mathcal{N} \twoheadrightarrow_i (\lambda^! x.\mathcal{P}')!\mathcal{N}'$ for $i \geq 1$.*

Observe that $\mathcal{M} \to_i \mathcal{N}$ implies $\mathcal{M} \twoheadrightarrow_i \mathcal{N}$, $\mathcal{M} \twoheadrightarrow_i \mathcal{N}$ implies $\mathcal{M} \xrightarrow{*}_i \mathcal{N}$, and $\xrightarrow{*}_i$ is the transitive closure of $\twoheadrightarrow_i$. Moreover, the following substitution property holds for $\twoheadrightarrow_i$:

Proposition 3. *i. If x occurs at depth 0 in $\mathcal{M}$, then $\mathcal{M} \twoheadrightarrow_i \mathcal{M}'$ and $\mathcal{N} \twoheadrightarrow_i \mathcal{N}'$ imply $\mathcal{M}[\mathcal{N}/x] \twoheadrightarrow_i \mathcal{M}'[\mathcal{N}'/x]$;*

ii. if x occurs at depth 1 in $\mathcal{M}$, then $\mathcal{M} \twoheadrightarrow_0 \mathcal{M}'$ implies $\mathcal{M}[\mathcal{N}/x] \twoheadrightarrow_0 \mathcal{M}'[\mathcal{N}/x]$, while $\mathcal{M} \twoheadrightarrow_i \mathcal{M}'$ and $!\mathcal{N} \twoheadrightarrow_i !\mathcal{N}'$ imply $\mathcal{M}[\mathcal{N}/x] \twoheadrightarrow_i \mathcal{M}'[\mathcal{N}'/x]$ for $i \geq 1$;

Proof. By induction on $\mathcal{M}$.

i. Let $\mathcal{M} = x$; by definition $\mathcal{M}' = x$, so $\mathcal{M}[\mathcal{N}/x] = \mathcal{N} \twoheadrightarrow_i \mathcal{N}' = \mathcal{M}'[\mathcal{N}'/x]$.

   Let $\mathcal{M} = \lambda y.\mathcal{P}$; by definition $\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ implies $\mathcal{M}' = \lambda y.\mathcal{P}'$, so by inductive hypothesis $\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ and $\mathcal{N} \twoheadrightarrow_i \mathcal{N}'$ imply $\mathcal{P}[\mathcal{N}/x] \twoheadrightarrow_i \mathcal{P}'[\mathcal{N}'/x]$: then $\mathcal{M}[\mathcal{N}/x] = \lambda y.\mathcal{P}[\mathcal{N}/x] \twoheadrightarrow_i \lambda y.\mathcal{P}'[\mathcal{N}'/x] = \mathcal{M}'[\mathcal{N}'/x]$.

   The case of $\mathcal{M} = \lambda^! y.\mathcal{P}$ is similar.

   Let $\mathcal{M} = !\mathcal{P}$: note that x cannot occur at depth 0 in $\mathcal{M}$, since $|\mathcal{M}|_0 = 0$, so this case is not possible.

   Let $\mathcal{M} = \mathcal{P}\mathcal{Q}$, such that x occurs at depth 0 in both $\mathcal{P}$ and $\mathcal{Q}$; if $x \notin \mathrm{FV}(\mathcal{P})$ or $x \notin \mathrm{FV}(\mathcal{Q})$, the proof is similar. By definition $\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ and $\mathcal{Q} \twoheadrightarrow_i \mathcal{Q}'$

imply $\mathcal{M}' = \mathcal{P}'\mathcal{Q}'$, so by inductive hypothesis $\mathcal{P} \not\mapsto_i \mathcal{P}'$, $\mathcal{Q} \not\mapsto_i \mathcal{Q}'$ and $\mathcal{N} \not\mapsto_i \mathcal{N}'$ imply $\mathcal{P}[\mathcal{N}/x] \not\mapsto_i \mathcal{P}'[\mathcal{N}'/x]$ and $\mathcal{Q}[\mathcal{N}/x] \not\mapsto_i \mathcal{Q}'[\mathcal{N}'/x]$: then $\mathcal{M}[\mathcal{N}/x] = \mathcal{P}[\mathcal{N}/x]\mathcal{Q}[\mathcal{N}/x] \not\mapsto_i \mathcal{P}'[\mathcal{N}'/x]\mathcal{Q}'[\mathcal{N}'/x] = \mathcal{M}'[\mathcal{N}'/x]$.

Let $\mathcal{M} = (\lambda y.\mathcal{P})\mathcal{Q}$; by definition $\mathcal{P} \not\mapsto_i \mathcal{P}'$ and $\mathcal{Q} \not\mapsto_i \mathcal{Q}'$ imply $\mathcal{M}' = \mathcal{P}'[\mathcal{Q}'/y]$ for $i = 0$, while $\mathcal{P} \not\mapsto_i \mathcal{P}'$ and $\mathcal{Q} \not\mapsto_i \mathcal{Q}'$ imply $\mathcal{M}' = (\lambda x.\mathcal{Q}')\mathcal{N}'$ for $i \geq 1$. By inductive hypothesis $\mathcal{P}[\mathcal{N}/x] \not\mapsto_i \mathcal{P}'[\mathcal{N}'/x]$ and $\mathcal{Q}[\mathcal{N}/x] \not\mapsto_i \mathcal{Q}'[\mathcal{N}'/x]$; then $\mathcal{M}[\mathcal{N}/x] = (\lambda y.\mathcal{P}[\mathcal{N}/x])\mathcal{Q}[\mathcal{N}/x] \not\mapsto_i \mathcal{P}'[\mathcal{N}'/x][\mathcal{Q}'[\mathcal{N}'/x]/y] = \mathcal{M}'[\mathcal{N}'/x]$ for $i = 0$, while $\mathcal{M}[\mathcal{N}/x] = (\lambda y.\mathcal{P}[\mathcal{N}/x])\mathcal{Q}[\mathcal{N}/x] \not\mapsto_i (\lambda y.\mathcal{P}'[\mathcal{N}'/x])\mathcal{Q}'[\mathcal{N}'/x] = \mathcal{M}'[\mathcal{N}'/x]$ for $i \geq 1$.

Let $\mathcal{M} = (\lambda^! y.\mathcal{P})!\mathcal{Q}$; by definition $\mathcal{P} \not\mapsto_i \mathcal{P}'$ implies $\mathcal{M}' = \mathcal{P}'[\mathcal{Q}/y]$ for $i = 0$, while $\mathcal{P} \not\mapsto_i \mathcal{P}'$ and $!\mathcal{Q} \not\mapsto_i !\mathcal{Q}'$ imply $\mathcal{M}' = (\lambda^! x.\mathcal{P}')!\mathcal{Q}'$ for $i \geq 1$. Observe that x cannot occur at depth 0 in $\mathcal{Q}$, so $!\mathcal{Q}[\mathcal{N}/x] \not\mapsto_0 !\mathcal{Q}$ and $!\mathcal{Q}[\mathcal{N}/x] \not\mapsto_i !\mathcal{Q}'$ for $i \geq 1$. By inductive hypothesis $\mathcal{P}[\mathcal{N}/x] \not\mapsto_i \mathcal{P}'[\mathcal{N}'/x]$: then $\mathcal{M}[\mathcal{N}/x] = (\lambda^! y.\mathcal{P}[\mathcal{N}/x])!\mathcal{Q}[\mathcal{N}/x] \not\mapsto_i \mathcal{P}'[\mathcal{N}'/x][\mathcal{Q}/y] = \mathcal{M}'[\mathcal{N}'/x]$ for $i = 0$, while $\mathcal{M}[\mathcal{N}/x] = (\lambda^! y.\mathcal{P}[\mathcal{N}/x])!\mathcal{Q}[\mathcal{N}/x] \not\mapsto_i (\lambda y.\mathcal{P}'[\mathcal{N}'/x])!\mathcal{Q}' = \mathcal{M}'[\mathcal{N}'/x]$ for $i \geq 1$.

ii. Let $\mathcal{M} = y$: note that x cannot occur at depth 1 in $\mathcal{M}$, since $|x|_1 = 0$, so this case is not possible.

Let $\mathcal{M} = \lambda y.\mathcal{P}$; by definition $\mathcal{P} \not\mapsto_i \mathcal{P}'$ implies $\mathcal{M}' = \lambda y.\mathcal{P}'$, so by inductive hypothesis $\mathcal{P} \not\mapsto_0 \mathcal{P}'$ implies $\mathcal{P}[\mathcal{N}/x] \not\mapsto_i \mathcal{P}'[\mathcal{N}/x]$ for $i = 0$, while $\mathcal{P} \not\mapsto_i \mathcal{P}'$ and $\mathcal{N} \not\mapsto_i \mathcal{N}'$ imply $\mathcal{P}[\mathcal{N}/x] \not\mapsto_i \mathcal{P}'[\mathcal{N}'/x]$ for $i \geq 1$: then $\mathcal{M}[\mathcal{N}/x] = \lambda y.\mathcal{P}[\mathcal{N}/x] \not\mapsto_i \lambda y.\mathcal{P}'[\mathcal{N}/x] = \mathcal{M}'[\mathcal{N}/x\,]$ for $i = 0$, while $\mathcal{M}[\mathcal{N}/x] = \lambda y.\mathcal{P}[\mathcal{N}/x] \not\mapsto_i \lambda y.\mathcal{P}'[\mathcal{N}'/x] = \mathcal{M}'[\mathcal{N}'/x]$ for $i \geq 1$.

The case of $\mathcal{M} = \lambda^! y.\mathcal{P}$ is similar.

Let $\mathcal{M} = !\mathcal{P}$; if x occurs at depth 1 in $\mathcal{M}$, then x occurs at depth 0 in $\mathcal{P}$. By definition $!\mathcal{P} \not\mapsto_i !\mathcal{P}$ implies $\mathcal{M}' = !\mathcal{P}$ for $i = 0$, while $\mathcal{P} \not\mapsto_{i-1} \mathcal{P}'$ implies $\mathcal{M}' = !\mathcal{P}'$ for $i \geq 1$. If $i = 0$, then $\mathcal{M}[\mathcal{N}/x] = !(\mathcal{P}[\mathcal{N}/x]) \not\mapsto_i !(\mathcal{P}[\mathcal{N}/x]) = \mathcal{M}'[\mathcal{N}/x]$. Otherwise, by point i of the current Lemma, $\mathcal{P} \not\mapsto_{i-1} \mathcal{P}'$ and $\mathcal{N} \not\mapsto_{i-1} \mathcal{N}'$ imply $\mathcal{P}[\mathcal{N}/x] \not\mapsto_{i-1} \mathcal{P}'[\mathcal{N}'/x]$: then, since $\mathcal{P} \not\mapsto_{i-1} \mathcal{P}'$ and $\mathcal{N} \not\mapsto_{i-1} \mathcal{N}'$ imply $!\mathcal{P} \not\mapsto_i !\mathcal{P}'$ and $!\mathcal{N} \not\mapsto_i !\mathcal{N}'$, we have $\mathcal{M}[\mathcal{N}/x] = !(\mathcal{P}[\mathcal{N}/x]) \not\mapsto_i !(\mathcal{P}'[\mathcal{N}'/x]) = \mathcal{M}'[\mathcal{N}'/x]$ for $i \geq 1$.

Let $\mathcal{M} = \mathcal{P}\mathcal{Q}$, such that x occurs at depth 1 in both $\mathcal{P}$ and $\mathcal{Q}$; if $x \notin \mathrm{FV}(\mathcal{P})$ or $x \notin \mathrm{FV}(\mathcal{Q})$, the proof is similar. By definition $\mathcal{P} \not\mapsto_i \mathcal{P}'$ and $\mathcal{Q} \not\mapsto_i \mathcal{Q}'$ imply $\mathcal{M}' = \mathcal{P}'\mathcal{Q}'$, so by inductive hypothesis $\mathcal{P}[\mathcal{N}/x] \not\mapsto_i \mathcal{P}'[\mathcal{N}/x]$ and $\mathcal{Q}[\mathcal{N}/x] \not\mapsto_i \mathcal{Q}'[\mathcal{N}/x]$ for $i = 0$, while $\mathcal{P}[\mathcal{N}/x] \not\mapsto_i \mathcal{P}'[\mathcal{N}'/x]$ and $\mathcal{Q}[\mathcal{N}/x] \not\mapsto_i \mathcal{Q}'[\mathcal{N}'/x]$ for $i \geq 1$: then $\mathcal{M}[\mathcal{N}/x] = \mathcal{P}[\mathcal{N}/x]\mathcal{Q}[\mathcal{N}/x] \not\mapsto_i \mathcal{P}'[\mathcal{N}/x]\mathcal{Q}'[\mathcal{N}/x] = \mathcal{M}'[\mathcal{N}/x\,]$ for $i = 0$, while $\mathcal{M}[\mathcal{N}/x] = \mathcal{P}[\mathcal{N}/x]\mathcal{Q}[\mathcal{N}/x] \not\mapsto_i \mathcal{P}'[\mathcal{N}'/x]\mathcal{Q}'[\mathcal{N}'/x] = \mathcal{M}'[\mathcal{N}'/x]$ for $i \geq 1$.

Let $\mathcal{M} = (\lambda y.\mathcal{P})\mathcal{Q}$; by definition $\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ and $\mathcal{Q} \twoheadrightarrow_i \mathcal{Q}'$ imply $\mathcal{M}' = \mathcal{P}'[\mathcal{Q}'/y]$ for $i = 0$, while $\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ and $\mathcal{Q} \twoheadrightarrow_i \mathcal{Q}'$ imply $\mathcal{M}' = (\lambda y.\mathcal{P}')\mathcal{Q}'$ for $i \geq 1$. By inductive hypothesis $\mathcal{P}[\mathcal{N}/x] \twoheadrightarrow_i \mathcal{P}'[\mathcal{N}/x]$ and $\mathcal{Q}[\mathcal{N}/x] \twoheadrightarrow_i \mathcal{Q}'[\mathcal{N}/x]$ for $i = 0$, while $\mathcal{P}[\mathcal{N}/x] \twoheadrightarrow_i \mathcal{P}'[\mathcal{N}'/x]$ and $\mathcal{Q}[\mathcal{N}/x] \twoheadrightarrow_i \mathcal{Q}'[\mathcal{N}'/x]$ for $i \geq 1$: then $\mathcal{M}[\mathcal{N}/x] = (\lambda y.\mathcal{P}[\mathcal{N}/x])\mathcal{Q}[\mathcal{N}/x] \twoheadrightarrow_i \mathcal{P}'[\mathcal{N}'/x][\mathcal{Q}'[\mathcal{N}'/x]/y] = \mathcal{M}'[\mathcal{N}'/x]$ for $i = 0$, while $\mathcal{M}[\mathcal{N}/x] = (\lambda y.\mathcal{P}[\mathcal{N}/x])\mathcal{Q}[\mathcal{N}/x] \twoheadrightarrow_i (\lambda y.\mathcal{P}'[\mathcal{N}'/x])\mathcal{Q}'[\mathcal{N}'/x] = \mathcal{M}'[\mathcal{N}'/x]$ for $i \geq 1$.

Let $\mathcal{M} = (\lambda^! y.\mathcal{P})!\mathcal{Q}$; by definition $\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ implies $\mathcal{M}' = \mathcal{P}'[\mathcal{Q}/y]$ for $i = 0$, while $\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ and $!\mathcal{Q} \twoheadrightarrow_i !\mathcal{Q}'$ imply $\mathcal{M}' = (\lambda x.\mathcal{P}')!\mathcal{Q}'$ for $i \geq 1$. By inductive hypothesis $\mathcal{P}[\mathcal{N}/x] \twoheadrightarrow_i \mathcal{P}'[\mathcal{N}/x]$ and $!\mathcal{Q}[\mathcal{N}/x] \twoheadrightarrow_i !\mathcal{Q}[\mathcal{N}/x]$ for $i = 0$, while $\mathcal{P}[\mathcal{N}/x] \twoheadrightarrow_i \mathcal{P}'[\mathcal{N}'/x]$ and $!\mathcal{Q}[\mathcal{N}/x] \twoheadrightarrow_i !\mathcal{Q}'[\mathcal{N}'/x]$ for $i \geq 1$: then $\mathcal{M}[\mathcal{N}/x] = (\lambda^! y.\mathcal{P}[\mathcal{N}/x])!\mathcal{Q}[\mathcal{N}/x] \twoheadrightarrow_i \mathcal{P}'[\mathcal{N}/x][\mathcal{Q}[\mathcal{N}/x]/y] = \mathcal{M}'[\mathcal{N}'/x]$ for $i = 0$, while $\mathcal{M}[\mathcal{N}/x] = (\lambda^! y.\mathcal{P}[\mathcal{N}/x])!\mathcal{Q}[\mathcal{N}/x] \twoheadrightarrow_i (\lambda^! y.\mathcal{P}'[\mathcal{N}'/x])\mathcal{Q}'[\mathcal{N}'/x] = \mathcal{M}'[\mathcal{N}'/x]$ for $i \geq 1$.

□

Lemma 40. *$\mathcal{M} \twoheadrightarrow_i \mathcal{M}'$ implies $\mathcal{M}' \twoheadrightarrow_i [\mathcal{M}]_i$.*

Proof. By induction on $\mathcal{M}$.

Let $\mathcal{M} = x$; by definition $\mathcal{M}' = x$ and $[\mathcal{M}]_i = x$.

Let $\mathcal{M} = \lambda x.\mathcal{P}$; by definition $\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ implies $\mathcal{M}' = \lambda x.\mathcal{P}'$. By inductive hypothesis $\mathcal{P}' \twoheadrightarrow_i [\mathcal{P}]_i$: then $\mathcal{M}' = \lambda x.\mathcal{P}' \twoheadrightarrow_i \lambda x.[\mathcal{P}]_i = [\mathcal{M}]_i$.

The case of $\mathcal{M} = \lambda^! x.\mathcal{P}$ is similar.

Let $\mathcal{M} = !\mathcal{P}$; by definition $!\mathcal{P} \twoheadrightarrow_0 !\mathcal{P}$ implies $\mathcal{M}' = !\mathcal{P}$, while $\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ implies $\mathcal{M}' = !\mathcal{P}'$ for $i \geq 1$. If $i = 0$, then $\mathcal{M} = !\mathcal{P} \twoheadrightarrow_i !\mathcal{P} = \mathcal{M}' = [\mathcal{M}]_i$; otherwise, by inductive hypothesis $\mathcal{P}' \twoheadrightarrow_{i-1} [\mathcal{P}]_{i-1}$: then $\mathcal{M}' = !\mathcal{P}' \twoheadrightarrow_i ![\mathcal{P}]_{i-1} = [\mathcal{M}]_i$.

Let $\mathcal{M} = \mathcal{P}\mathcal{Q}$; by definition $\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ and $\mathcal{Q} \twoheadrightarrow_i \mathcal{Q}'$ imply $\mathcal{M}' = \mathcal{P}'\mathcal{Q}'$. By inductive hypothesis $\mathcal{P}' \twoheadrightarrow_i [\mathcal{P}]_i$ and $\mathcal{Q}' \twoheadrightarrow_i [\mathcal{Q}]_i$: then $\mathcal{M}' = \mathcal{P}'\mathcal{Q}' \twoheadrightarrow_i [\mathcal{P}]_i[\mathcal{Q}]_i = [\mathcal{M}]_i$.

Let $\mathcal{M} = (\lambda x.\mathcal{P})\mathcal{Q}$; if $x \notin \mathrm{FV}(\mathcal{P})$, then the proof follows by induction. Otherwise, since $\mathcal{M}$ is a w.f. term, x occurs at depth 0 in $\mathcal{P}$. By definition $\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ and $\mathcal{Q} \twoheadrightarrow_i \mathcal{Q}'$ imply $\mathcal{M}' = \mathcal{P}'[\mathcal{Q}'/x]$ for $i = 0$, while $\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ and $\mathcal{Q} \twoheadrightarrow_i \mathcal{Q}'$ imply $\mathcal{M}' = (\lambda x.\mathcal{P}')\mathcal{Q}'$ for $i \geq 1$. By inductive hypothesis $\mathcal{P}' \twoheadrightarrow_i [\mathcal{P}]_i$ and $\mathcal{Q}' \twoheadrightarrow_i [\mathcal{Q}]_i$: then by Lemma 3.i $\mathcal{M}' = \mathcal{P}'[\mathcal{Q}'/x] \twoheadrightarrow_i [\mathcal{P}]_i[[\mathcal{Q}]_i/x] = [\mathcal{M}]_i$ for $i = 0$, while $\mathcal{M}' = (\lambda x.\mathcal{P}')\mathcal{Q}' \twoheadrightarrow_i (\lambda x.[\mathcal{P}]_i)[\mathcal{Q}]_i = [\mathcal{M}]_i$.

Let $\mathcal{M} = (\lambda^! x.\mathcal{P})!\mathcal{Q}$; if $x \notin \mathrm{FV}(\mathcal{P})$, then the proof follows by induction. Otherwise, since $\mathcal{M}$ is a w.f. term, x occurs at depth 1 in $\mathcal{P}$. By definition $\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ implies $\mathcal{M}' = \mathcal{P}'[\mathcal{Q}/x] = [\mathcal{M}]_i$ for $i = 0$, while $\mathcal{P} \twoheadrightarrow_i \mathcal{P}'$ and $!\mathcal{Q} \twoheadrightarrow_i !\mathcal{Q}'$ imply $\mathcal{M}' = (\lambda^! x.\mathcal{P}')!\mathcal{Q}'$ for $i \geq 1$. By inductive hypothesis $\mathcal{P}' \twoheadrightarrow_i [\mathcal{P}]_i$ and

$!\mathcal{Q}' \not\mapsto_{i-1} [!\mathcal{Q}]_i$: then by Lemma 3.ii $\mathcal{M}' = \mathcal{P}'[\mathcal{Q}/x] \not\mapsto_i [\mathcal{P}]_i[\mathcal{Q}/x] = [\mathcal{M}]_i$ for $i = 0$, while $\mathcal{M}' = (\lambda^! x.\mathcal{P}')!\mathcal{Q}' \not\mapsto_i (\lambda x.[\mathcal{P}]_i)[!\mathcal{Q}]_i = [\mathcal{M}]_i$.

□

The proof of confluence is based on the fact that $\overset{*}{\to}_i$, being the transitive closure of $\not\mapsto_i$ for which the diamond property holds, is itself confluent:

Lemma 41 (Diamond property of $\not\mapsto_i$)**.** *If $\mathcal{M} \not\mapsto_i \mathcal{P}$ and $\mathcal{M} \not\mapsto_i \mathcal{Q}$, then there is $\mathcal{N}$ such that both $\mathcal{P} \not\mapsto_i \mathcal{N}$ and $\mathcal{Q} \not\mapsto_i \mathcal{N}$.*

Proof. By Lemma 40, $\mathcal{M} \not\mapsto_i \mathcal{P}$ implies $\mathcal{P} \not\mapsto_i [\mathcal{M}]_i$ and $\mathcal{M} \not\mapsto_i \mathcal{Q}$ implies $\mathcal{Q} \not\mapsto_i [\mathcal{M}]_i$, so $\mathcal{N} = [\mathcal{M}]_i$, as in Figure A.1.

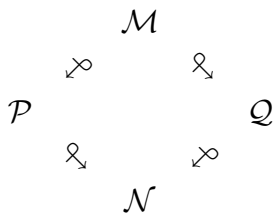

Figure A.1: The diamond property.

□

Finally the proof of the confluence at fixed depth follows:

of Prop. 1.i. Recall that $\overset{*}{\to}_i$ is the transitive closure of $\not\mapsto_i$; therefore, there are $\mathcal{P}_1, ..., \mathcal{P}_n$ and $\mathcal{Q}_1, ..., \mathcal{Q}_m$ such that $\mathcal{M} \not\mapsto_i \mathcal{P}_1 \not\mapsto_i ... \not\mapsto_i \mathcal{P}_n \not\mapsto_i \mathcal{P}$ and $\mathcal{M} \not\mapsto_i \mathcal{Q}_1 \not\mapsto_i ... \not\mapsto_i \mathcal{Q}_m \not\mapsto_i \mathcal{Q}$: then the proof follows by repeatedly applying Lemma 41, as in Figure A.2. □

A.2 Derivability of the rules for ⊗

In this section we prove that the rules of Table 2.2 are derivable by the rules of Table 2.1.

i) For rule $(\multimap I^L_\otimes)$, we have:

$$\dfrac{\dfrac{\dfrac{}{x : \mathtt{A}_1 \otimes \mathtt{A}_2 \mid \Delta \mid \Theta \vdash x : \mathtt{A}_1 \otimes \mathtt{A}_2}\,(Ax_L)}{x : \mathtt{A}_1 \otimes \mathtt{A}_2 \mid \Delta \mid \Theta \vdash x : (\mathtt{A}_1 \multimap \mathtt{A}_2 \multimap \mathtt{A}) \multimap \mathtt{A}}\,(\forall E) \quad \Sigma}{x : \mathtt{A}_1 \otimes \mathtt{A}_2 \mid \Delta \mid \Theta \vdash x\lambda y_1 y_2.\lambda z.zy_1y_2 : \mathtt{A}}\,(\multimap E) \quad \dfrac{\dfrac{\Gamma \mid \Delta, x_1 : \mathtt{A}_1, x_2 : \mathtt{A}_2 \mid \Theta \vdash \mathcal{M} : \tau}{\Gamma \mid \Delta, x_1 : \mathtt{A}_1 \mid \Theta \vdash \lambda x_2.\mathcal{M} : \mathtt{A}_2 \multimap \tau}\,(\multimap I^I)}{\Gamma \mid \Delta \mid \Theta \vdash \lambda x_1 x_2.\mathcal{M} : \mathtt{A}_1 \multimap \mathtt{A}_2 \multimap \tau}\,(\multimap I^L)$$

$$\dfrac{\Gamma, x : \mathtt{A}_1 \otimes \mathtt{A}_2 \mid \Delta \mid \Theta \vdash (x\lambda y_1 y_2.\lambda z.zy_1y_2)\lambda x_1 x_2.\mathcal{M} : \tau}{\Gamma \mid \Delta \mid \Theta \vdash \lambda x.(x\lambda y_1 y_2.\lambda z.zy_1y_2)\lambda x_1 x_2.\mathcal{M} : (\mathtt{A}_1 \otimes \mathtt{A}_2) \multimap \tau}\,(\multimap E)\,(\multimap I^L)$$

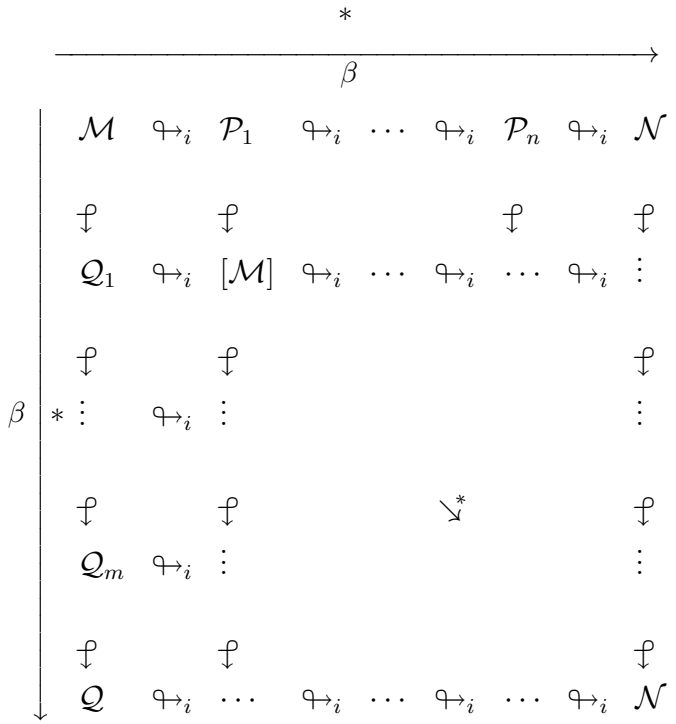

Figure A.2: The diamond closure.

where $\mathtt{B} = \mathtt{A}_1 \multimap \mathtt{A}_2 \multimap \tau$, $\mathtt{A} = \mathtt{B} \multimap \tau$ and Σ is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{z : \mathtt{B} \mid \Delta \mid \Theta \vdash z : \mathtt{B}}\,(Ax_L) \quad \dfrac{}{y_1 : \mathtt{A}_1 \mid \Delta \mid \Theta \vdash y_1 : \mathtt{A}_1}\,(Ax_L)}{y_1 : \mathtt{A}_1, z : \mathtt{B} \mid \Delta \mid \Theta \vdash z y_1 : \mathtt{A}_2 \multimap \tau}\,(\multimap E) \quad \dfrac{}{y_2 : \mathtt{A}_2 \mid \Delta \mid \Theta \vdash y_2 : \mathtt{A}_2}\,(Ax_L)}{y_1 : \mathtt{A}_1, y_2 : \mathtt{A}_2, z : \mathtt{B} \mid \Delta \mid \Theta \vdash z y_1 y_2 : \tau}\,(\multimap E)}{y_1 : \mathtt{A}_1, y_2 : \mathtt{A}_2 \mid \Delta \mid \Theta \vdash \lambda z. z y_1 y_2 : \mathtt{A}}\,(\multimap I_L)}{y_1 : \mathtt{A}_1 \mid \Delta \mid \Theta \vdash \lambda y_2 . \lambda z. z y_1 y_2 : \mathtt{A}_2 \multimap \mathtt{A}}\,(\multimap I_L)}{\emptyset \mid \Delta \mid \Theta \vdash \lambda y_1 y_2 . \lambda z. z y_1 y_2 : \mathtt{A}_1 \multimap \mathtt{A}_2 \multimap \mathtt{A}}\,(\multimap I_L)$$

ii) For rule $(\multimap I^I_{\otimes})$, we have:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x :! \sigma_1 \otimes\, ! \sigma_2 \mid \Delta \mid \Theta \vdash x :! \sigma_1 \otimes\, ! \sigma_2}\,(Ax_L)}{x :! \sigma_1 \otimes\, ! \sigma_2 \mid \Delta \mid \Theta \vdash x : (! \sigma_1 \multimap ! \sigma_2 \multimap \mathtt{A}) \multimap \mathtt{A}}\,(\forall E) \quad \Sigma}{x :! \sigma_1 \otimes\, ! \sigma_2 \mid \Delta \mid \Theta \vdash x \lambda^! y_1 y_2 . \lambda z. z ! y_1 ! y_2 : \mathtt{A}}\,(\multimap E) \quad \dfrac{\dfrac{\Gamma \mid \Delta, x_1 :! \sigma_1, x_2 :! \sigma_2 \mid \Theta \vdash \mathcal{M} : \tau}{\Gamma \mid \Delta, x_1 :! \sigma_1 \mid \Theta \vdash \lambda^! x_2 . \mathcal{M} :! \sigma_2 \multimap \tau}\,(\multimap I^I)}{\Gamma \mid \Delta \mid \Theta \vdash \lambda^! x_1 x_2 . \mathcal{M} :! \sigma_1 \multimap ! \sigma_2 \multimap \tau}\,(\multimap I^I)}{\Gamma, x :! \sigma_1 \otimes\, ! \sigma_2 \mid \Delta \mid \Theta \vdash (x \lambda^! y_1 y_2 . \lambda z. z ! y_1 ! y_2) \lambda^! x_1 x_2 . \mathcal{M} : \tau}\,(\multimap E)}{\Gamma \mid \Delta \mid \Theta \vdash \lambda x. (x \lambda^! y_1 y_2 . \lambda z. z ! y_1 ! y_2) \lambda^! x_1 x_2 . \mathcal{M} : (! \sigma_1 \otimes\, ! \sigma_2) \multimap \tau}\,(\multimap I^L)$$

where $\mathtt{B} = !\sigma_1 \multimap !\sigma_2 \multimap \tau$, $\mathtt{A} = \mathtt{B} \multimap \tau$, $\Delta' = y_1 :!\sigma_1, y_2 :!\sigma_2, \Delta$ and Σ is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{z : \mathtt{B} \mid \Delta' \mid \Theta \vdash z : \mathtt{B}}\,(Ax_L) \quad \dfrac{\dfrac{}{\emptyset \mid \emptyset \mid y_1 : \sigma_1 \vdash y_1 : \sigma_1}\,(Ax_P)}{\emptyset \mid \Delta' \mid \Theta \vdash !y_1 :!\sigma_1}\,(!)}{z : \mathtt{B} \mid \Delta' \mid \Theta \vdash z!y_1 :!\sigma_2 \multimap \tau}\,(\multimap E) \quad \dfrac{\dfrac{}{\emptyset \mid \emptyset \mid y_2 : \sigma_2 \vdash y_2 : \sigma_2}\,(Ax_P)}{\emptyset \mid \Delta' \mid \Theta \vdash !y_2 :!\sigma_2}\,(!)}{z : \mathtt{B} \mid \Delta' \mid \Theta \vdash z!y_1!y_2 : \tau}\,(\multimap E)}{\emptyset \mid \Delta' \mid \Theta \vdash \lambda z.z!y_1!y_2 : \mathtt{A}}\,(\multimap I_L)}{\emptyset \mid y_1 :!\sigma_1, \Delta \mid \Theta \vdash \lambda^! y_2.\lambda z.z!y_1!y_2 :!\sigma_2 \multimap \mathtt{A}}\,(\multimap I_I)}{\emptyset \mid \Delta \mid \Theta \vdash \lambda^! y_1 y_2.\lambda z.z!y_1!y_2 :!\sigma_1 \multimap !\sigma_2 \multimap \mathtt{A}}\,(\multimap I_I)$$

iii) For rule $(\otimes I)$, we have:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \sigma_1 \multimap \sigma_2 \multimap \mathtt{a} \mid \Delta \mid \Theta \vdash x : \sigma_1 \multimap \sigma_2 \multimap \mathtt{a}}\,(Ax^L) \quad \Gamma_1 \mid \Delta \mid \Theta \vdash \mathcal{M}_1 : \sigma_1}{x : \sigma_1 \multimap \sigma_2 \multimap \mathtt{a}, \Gamma_1 \mid \Delta \mid \Theta \vdash x\mathcal{M}_1 : \sigma_2 \multimap \mathtt{a}}\,(\multimap E) \quad \Gamma_2 \mid \Delta \mid \Theta \vdash \mathcal{M}_2 : \sigma_2}{x : \sigma_1 \multimap \sigma_2 \multimap \mathtt{a}, \Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash x\mathcal{M}_1\mathcal{M}_2 : \mathtt{a}}\,(\multimap E)}{\Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash \lambda x.x\mathcal{M}_1\mathcal{M}_2 : (\sigma_1 \multimap \sigma_2 \multimap \mathtt{a}) \multimap \mathtt{a}}\,(\multimap I^L)}{\Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash \lambda x.x\mathcal{M}_1\mathcal{M}_2 : \sigma_1 \otimes \sigma_2}\,(\forall I)$$

where $\lambda x.x\mathcal{M}_1\mathcal{M}_2 = \mathcal{M}_1 \otimes \mathcal{M}_2$ by definition.

Bibliography

[AR02] A. Asperti and L. Roversi, "Intuitionistic light affine logic", *ACM Trans. Comput. Log.*, vol. 3, no. 1, pp. 137–175, 2002. DOI: 10.1145/504077.504081. [Online]. Available: http://doi.acm.org/10.1145/504077.504081.

[Bai04] P. Baillot, "Stratified coherence spaces: a denotational semantics for light linear logic", *Theor. Comput. Sci.*, vol. 318, no. 1-2, pp. 29–55, 2004. DOI: 10.1016/j.tcs.2003.10.015. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2003.10.015.

[Bai10] P. Baillot, Ed., *Proceedings International Workshop on Developments in Implicit Computational complExity, DICE 2010, Paphos, Cyprus, 27-28th March 2010*, vol. 23, ser. EPTCS, 2010. DOI: 10.4204/EPTCS.23. [Online]. Available: http://dx.doi.org/10.4204/EPTCS.23.

[Bai11] ——, "Elementary linear logic revisited for polynomial time and an exponential time hierarchy", in *Programming Languages and Systems - 9th Asian Symposium, APLAS 2011, Kenting, Taiwan, December 5-7, 2011. Proceedings*, H. Yang, Ed., ser. Lecture Notes in Computer Science, vol. 7078, Springer, 2011, pp. 337–352, ISBN: 978-3-642-25317-1. DOI: 10.1007/978-3-642-25318-8_25. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25318-8_25.

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

[BC92] S. Bellantoni and S. A. Cook, "A new recursion-theoretic characterization of the polytime functions (extended abstract)", in *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, S. R. Kosaraju, M. Fellows, A. Wigderson, and J. A. Ellis, Eds., ACM, 1992, pp. 283–293, ISBN: 0-89791-511-9. DOI: 10.1145/129712.129740. [Online]. Available: http://doi.acm.org/10.1145/129712.129740.

[BDR14] P. Baillot, E. De Benedetti, and S. Ronchi Della Rocca, "Characterizing polynomial and exponential complexity classes in elementary lambda-calculus", in *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*, J. Diaz, I. Lanese, and D. Sangiorgi, Eds., ser. Lecture Notes in Computer Science, vol. 8705, Springer, 2014, pp. 151–163, ISBN: 978-3-662-44601-0. DOI: 10.1007/978-3-662-44602-7_13. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-44602-7_13.

[BDS13] H. P. Barendregt, W. Dekkers, and R. Statman, *Lambda Calculus with Types*, ser. Perspectives in logic. Cambridge University Press, 2013, ISBN: 978-0-521-76614-2. [Online]. Available: http://www.cambridge.org/de/academic/subjects/mathematics/logic-categories-and-sets/lambda-calculus-types.

[BEM07] A. Bucciarelli, T. Ehrhard, and G. Manzonetto, "Not enough points is enough", in *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, J. Duparc and T. A. Henzinger, Eds., ser. Lecture Notes in Computer Science, vol. 4646, Springer, 2007, pp. 298–312, ISBN: 978-3-540-74914-1. DOI: 10.1007/978-3-540-74915-8_24. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74915-8_24.

[BGM10] P. Baillot, M. Gaboardi, and V. Mogbil, "A polytime functional language from light linear logic", in *Programming Languages and Systems, 19th European Symposium on Programming, ESOP 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, A. D. Gordon, Ed., ser. Lecture Notes in Computer Science, vol. 6012, Springer, 2010, pp. 104–124, ISBN: 978-3-642-11956-9. DOI: 10.1007/978-3-642-11957-6_7. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-11957-6_7.

[BL11] A. Bernadet and S. Lengrand, "Complexity of strongly normalising λ-terms via non-idempotent intersection types", in *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, M. Hofmann, Ed., ser. Lecture Notes in Computer Science, vol. 6604, Springer, 2011, pp. 88–107, ISBN:

978-3-642-19804-5. DOI: 10.1007/978-3-642-19805-2_7. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-19805-2_7.

[BPS03] A. Bucciarelli, A. Piperno, and I. Salvo, "Intersection types and lambda-definability", *Mathematical Structures in Computer Science*, vol. 13, no. 1, pp. 15–53, 2003. DOI: 10.1017/S0960129502003833. [Online]. Available: http://dx.doi.org/10.1017/S0960129502003833.

[BT04] P. Baillot and K. Terui, "Light types for polynomial time computation in lambda-calculus", in *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, IEEE Computer Society, 2004, pp. 266–275, ISBN: 0-7695-2192-4. DOI: 10.1109/LICS.2004.1319621. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/LICS.2004.1319621.

[BT10] A. Brunel and K. Terui, "Church => scott = ptime: an application of resource sensitive realizability", in *Proceedings International Workshop on Developments in Implicit Computational complExity, DICE 2010, Paphos, Cyprus, 27-28th March 2010.*, P. Baillot, Ed., ser. EPTCS, vol. 23, 2010, pp. 31–46. DOI: 10.4204/EPTCS.23.3. [Online]. Available: http://dx.doi.org/10.4204/EPTCS.23.3.

[CD12] P. Cégielski and A. Durand, Eds., *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, vol. 16, ser. LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012, ISBN: 978-3-939897-42-2. [Online]. Available: http://drops.dagstuhl.de/opus/portals/extern/index.php?semnr=12009.

[CD80] M. Coppo and M. Dezani-Ciancaglini, "An extension of the basic functionality theory for the lambda-calculus", *Notre-Dame Journal of Formal Logic*, vol. 21, no. 4, pp. 685–693, 1980.

[CDCV80] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri, "Principal type schemes and lambda-calculus semantics", in *To H. B. Curry: essays on combinatory logic, lambda calculus and formalism*, London: Academic Press, 1980, pp. 535–560.

[CDR08] P. Coppola, U. Dal Lago, and S. Ronchi Della Rocca, "Light logics and the call-by-value lambda calculus", *Logical Methods in Computer Science*, vol. 4, no. 4, 2008. DOI: 10.2168/LMCS-4(4:5)2008. [Online]. Available: http://dx.doi.org/10.2168/LMCS-4(4:5)2008.

[Cob65] A. Cobham, "The intrinsic computational difficulty of functions", in *Logic, Methodology and Philosophy of Science, proceedings of the second International Congress, held in Jerusalem, 1964*, Y. Bar-Hillel, Ed., Amsterdam: North-Holland, 1965.

[CS12] J. Chrzaszcz and A. Schubert, "ML with PTIME complexity guarantees", in *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, P. Cégielski and A. Durand, Eds., ser. LIPIcs, vol. 16, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012, pp. 198–212, ISBN: 978-3-939897-42-2. DOI: 10.4230/LIPIcs.CSL.2012.198. [Online]. Available: http://dx.doi.org/10.4230/LIPIcs.CSL.2012.198.

[DB11] E. De Benedetti, "Polynomial lambda-calculus via intersection types", Master's thesis, Università degli Studi di Torino, 2011.

[Dbla] *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, IEEE Computer Society, 1999, ISBN: 0-7695-0158-3.

[Dblb] *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*, IEEE Computer Society, 2001, ISBN: 0-7695-1281-X.

[Dblc] *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, IEEE Computer Society, 2002, ISBN: 0-7695-1483-9.

[Dbld] *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, IEEE Computer Society, 2004, ISBN: 0-7695-2192-4.

[Dble] *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, IEEE Computer Society, 2011, ISBN: 978-0-7695-4412-0.

[dCa09] D. de Carvalho, "Execution time of lambda-terms via denotational semantics and intersection types", *MSCS*, vol. to appear, 2009. [Online]. Available: http://arxiv.org/abs/0905.4251.

[DCGL98] M. Dezani-Ciancaglini, E. Giovannetti, and U. de'Liguoro, "Intersection Types, Lambda-models and Böhm Trees", in *MSJ-Memoir Vol. 2 "Theories of Types and Proofs"*, vol. 2, Mathematical Society of Japan, 1998, pp. 45–97. [Online]. Available: http://www.di.unito.it/~dezani/papers/11.ps.

[DH07] J. Duparc and T. A. Henzinger, Eds., *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, vol. 4646, ser. Lecture Notes in Computer Science, Springer, 2007, ISBN: 978-3-540-74914-1.

[DHL08] P. Di Gianantonio, F. Honsell, and M. Lenisa, "A type assignment system for game semantics", *Theor. Comput. Sci.*, vol. 398, no. 1-3, pp. 150–169, 2008. DOI: 10.1016/j.tcs.2008.01.023. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2008.01.023.

[DJ03] V. Danos and J. Joinet, "Linear logic and elementary time", *Inf. Comput.*, vol. 183, no. 1, pp. 123–137, 2003. DOI: 10.1016/S0890-5401(03)00010-5. [Online]. Available: http://dx.doi.org/10.1016/S0890-5401(03)00010-5.

[DL09] U. Dal Lago, "Context semantics, linear logic, and computational complexity", *ACM Trans. Comput. Log.*, vol. 10, no. 4, 2009. DOI: 10.1145/1555746.1555749. [Online]. Available: http://doi.acm.org/10.1145/1555746.1555749.

[DLS14] J. Diaz, I. Lanese, and D. Sangiorgi, Eds., *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*, vol. 8705, ser. Lecture Notes in Computer Science, Springer, 2014, ISBN: 978-3-662-44601-0. DOI: 10.1007/978-3-662-44602-7. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-44602-7.

[DMZ10] U. Dal Lago, A. Masini, and M. Zorzi, "Quantum implicit computational complexity", *Theor. Comput. Sci.*, vol. 411, no. 2, pp. 377–409, 2010. DOI: 10.1016/j.tcs.2009.07.045. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2009.07.045.

[DR13] E. De Benedetti and S. Ronchi Della Rocca, "Bounding normalization time through intersection types", in *Proceedings Sixth Workshop on Intersection Types and Related Systems, ITRS 2012, Dubrovnik, Croatia, 29th June 2012.*, S. Graham-Lengrand and L. Paolini, Eds., ser. EPTCS, vol. 121, 2013, pp. 48–57. DOI: 10.4204/EPTCS.121.4. [Online]. Available: http://dx.doi.org/10.4204/EPTCS.121.4.

[DR14] ——, "A type assignment for lambda-calculus complete both for fptime and strong normalization", *Information & Computation*, vol. to appear, 2014. [Online]. Available: http://arxiv.org/abs/1410.6298.

[Gir87] J. Girard, "Linear logic", *Theor. Comput. Sci.*, vol. 50, pp. 1–102, 1987. DOI: 10.1016/0304-3975(87)90045-4. [Online]. Available: http://dx.doi.org/10.1016/0304-3975(87)90045-4.

[Gir98] J. Girard, "Light linear logic", *Inf. Comput.*, vol. 143, no. 2, pp. 175–204, 1998. DOI: 10.1006/inco.1998.2700. [Online]. Available: http://dx.doi.org/10.1006/inco.1998.2700.

[Gor10] A. D. Gordon, Ed., *Programming Languages and Systems, 19th European Symposium on Programming, ESOP 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, vol. 6012, ser. Lecture Notes in Computer Science, Springer, 2010, ISBN: 978-3-642-11956-9. DOI: 10.1007/978-3-642-11957-6. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-11957-6.

[GP13] S. Graham-Lengrand and L. Paolini, Eds., *Proceedings Sixth Workshop on Intersection Types and Related Systems, ITRS 2012, Dubrovnik, Croatia, 29th June 2012*, vol. 121, ser. EPTCS, 2013. DOI: 10.4204/EPTCS.121. [Online]. Available: http://dx.doi.org/10.4204/EPTCS.121.

[GR07] M. Gaboardi and S. Ronchi Della Rocca, "A soft type assignment system for *lambda*-calculus", in *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, J. Duparc and T. A. Henzinger, Eds., ser. Lecture Notes in Computer Science, vol. 4646, Springer, 2007, pp. 253–267, ISBN: 978-3-540-74914-1. DOI: 10.1007/978-3-540-74915-8_21. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74915-8_21.

[GSS92] J. Girard, A. Scedrov, and P. J. Scott, "Bounded linear logic: A modular approach to polynomial-time computability", *Theor. Comput. Sci.*, vol. 97, no. 1, pp. 1–66, 1992. DOI: 10.1016/0304-3975(92)90386-T. [Online]. Available: http://dx.doi.org/10.1016/0304-3975(92)90386-T.

[Hof11] M. Hofmann, Ed., *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, vol. 6604, ser. Lecture Notes in Computer Science, Springer, 2011, ISBN: 978-3-642-19804-5. DOI: 10.1007/978-3-642-19805-2.

[Online]. Available: http://dx.doi.org/10.1007/978-3-642-19805-2.

[Hof99] M. Hofmann, "Linear types and non-size-increasing polynomial time computation", in *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, IEEE Computer Society, 1999, pp. 464–473, ISBN: 0-7695-0158-3. DOI: 10.1109/LICS.1999.782641. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/LICS.1999.782641.

[Jon01] N. D. Jones, "The expressive power of higher-order types or, life without CONS", *J. Funct. Program.*, vol. 11, no. 1, pp. 5–94, 2001. [Online]. Available: http://journals.cambridge.org/action/displayAbstract?aid=68581.

[Jon97] ——, *Computability and complexity - from a programming perspective*, ser. Foundations of computing series. MIT Press, 1997, ISBN: 978-0-262-10064-9.

[Kfo00] A. J. Kfoury, "A linearization of the lambda-calculus and consequences", *J. Log. Comput.*, vol. 10, no. 3, pp. 411–436, 2000. DOI: 10.1093/logcom/10.3.411. [Online]. Available: http://dx.doi.org/10.1093/logcom/10.3.411.

[KFWE92] S. R. Kosaraju, M. Fellows, A. Wigderson, and J. A. Ellis, Eds., *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, ACM, 1992, ISBN: 0-89791-511-9.

[KV14] D. Kesner and D. Ventura, "Quantitative types for the linear substitution calculus", in *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*, J. Diaz, I. Lanese, and D. Sangiorgi, Eds., ser. Lecture Notes in Computer Science, vol. 8705, Springer, 2014, pp. 296–310, ISBN: 978-3-662-44601-0. DOI: 10.1007/978-3-662-44602-7_23. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-44602-7_23.

[KW04] A. J. Kfoury and J. B. Wells, "Principality and type inference for intersection types using expansion variables", *Theor. Comput. Sci.*, vol. 311, no. 1-3, pp. 1–70, 2004. DOI: 10.1016/j.tcs.2003.10.032. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2003.10.032.

[Laf04] Y. Lafont, "Soft linear logic and polynomial time", *Theor. Comput. Sci.*, vol. 318, no. 1-2, pp. 163–180, 2004. DOI: 10.1016/j.tcs.2003.10.018. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2003.10.018.

[LB06] U. D. Lago and P. Baillot, "On light logics, uniform encodings and polynomial time", *Mathematical Structures in Computer Science*, vol. 16, no. 4, pp. 713–733, 2006. DOI: 10.1017/S0960129506005421. [Online]. Available: http://dx.doi.org/10.1017/S0960129506005421.

[Lei02] D. Leivant, "Calibrating computational feasibility by abstraction rank", in *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, IEEE Computer Society, 2002, p. 345, ISBN: 0-7695-1483-9. DOI: 10.1109/LICS.2002.1029842. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/LICS.2002.1029842.

[Lei94] D. Leivant, "Predicative recurrence and computational complexity I: word recurrence and poly-time", in *Feasible Mathematics II*, Birkhauser, 1994, pp. 320–343.

[LM93] D. Leivant and J. Marion, "Lambda calculus characterizations of poly-time", *Fundam. Inform.*, vol. 19, no. 1/2, pp. 167–184, 1993.

[MA11] A. Madet and R. M. Amadio, "An elementary affine λ-calculus with multithreading and side effects", in *Typed Lambda Calculi and Applications - 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011. Proceedings*, C. L. Ong, Ed., ser. Lecture Notes in Computer Science, vol. 6690, Springer, 2011, pp. 138–152, ISBN: 978-3-642-21690-9. DOI: 10.1007/978-3-642-21691-6_13. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-21691-6_13.

[Mar11] J. Marion, "A type system for complexity flow analysis", in *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, IEEE Computer Society, 2011, pp. 123–132, ISBN: 978-0-7695-4412-0. DOI: 10.1109/LICS.2011.41. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/LICS.2011.41.

[Maz06] D. Mazza, "Linear logic and polynomial time", *Mathematical Structures in Computer Science*, vol. 16, no. 6, pp. 947–988, 2006. DOI: 10.1017/S0960129506005688. [Online]. Available: http://dx.doi.org/10.1017/S0960129506005688.

[Nee05] P. M. Neergaard, "Theoretical pearls: A bargain for intersection types: a simple strong normalization proof", *J. Funct. Program.*, vol. 15, no. 5, pp. 669–677, 2005. DOI: 10.1017/S0956796805005587. [Online]. Available: http://dx.doi.org/10.1017/S0956796805005587.

[NM04] P. M. Neergaard and H. G. Mairson, "Types, potency, and idempotency: why nonlinearity and amnesia make a type system work", in *Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming, ICFP 2004, Snow Bird, UT, USA, September 19-21, 2004*, C. Okasaki and K. Fisher, Eds., ACM, 2004, pp. 138–149, ISBN: 1-58113-905-5. DOI: 10.1145/1016850.1016871. [Online]. Available: http://doi.acm.org/10.1145/1016850.1016871.

[OF04] C. Okasaki and K. Fisher, Eds., *Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming, ICFP 2004, Snow Bird, UT, USA, September 19-21, 2004*, ACM, 2004, ISBN: 1-58113-905-5.

[Ong11] C. L. Ong, Ed., *Typed Lambda Calculi and Applications - 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011. Proceedings*, vol. 6690, ser. Lecture Notes in Computer Science, Springer, 2011, ISBN: 978-3-642-21690-9. DOI: 10.1007/978-3-642-21691-6. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-21691-6.

[Pap94] C. H. Papadimitriou, *Computational complexity*. Addison-Wesley, 1994, ISBN: 978-0-201-53082-7.

[PR10] M. Pagani and S. Ronchi Della Rocca, "Linearity, non-determinism and solvability", *Fundam. Inform.*, vol. 103, no. 1-4, pp. 173–202, 2010. DOI: 10.3233/FI-2010-324. [Online]. Available: http://dx.doi.org/10.3233/FI-2010-324.

[Red13] B. F. Redmond, "Stratified combinatory logic and lower complexity", in *Proceedings of DICE 2013*, 2013.

[RP04] S. Ronchi Della Rocca and L. Paolini, *The Parametric λ-Calculus: a Metamodel for Computation*, ser. Texts in Theoretical Computer Science: An EATCS Series. Berlin: Springer-Verlag, 2004, pp. 1–252, ISBN: 3-540-20032-0. [Online]. Available: http://www.springer.com/sgw/cda/frontpage/0,,5-40356-72-14202886-0,00.html.

[RR97] S. Ronchi Della Rocca and L. Roversi, "Lambda calculus and intuitionistic linear logic", *Studia Logica*, vol. 59, no. 3, pp. 417–448, 1997. DOI: 10.1023/A:1005092630115. [Online]. Available: http://dx.doi.org/10.1023/A:1005092630115.

[RSSX99] F. van Raamsdonk, P. Severi, M. H. Sørensen, and H. Xi, "Perpetual reductions in lambda-calculus", *Inf. Comput.*, vol. 149, no. 2, pp. 173–225, 1999. DOI: 10.1006/inco.1998.2750. [Online]. Available: http://dx.doi.org/10.1006/inco.1998.2750.

[RV10] L. Roversi and L. Vercelli, "Safe recursion on notation into a light logic by levels", in *Proceedings International Workshop on Developments in Implicit Computational complExity, DICE 2010, Paphos, Cyprus, 27-28th March 2010.*, P. Baillot, Ed., ser. EPTCS, vol. 23, 2010, pp. 63–77. DOI: 10.4204/EPTCS.23.5. [Online]. Available: http://dx.doi.org/10.4204/EPTCS.23.5.

[Sip97] M. Sipser, *Introduction to the theory of computation*. PWS Publishing Company, 1997, ISBN: 978-0-534-94728-6.

[Ter01] K. Terui, "Light affine calculus and polytime strong normalization", in *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*, IEEE Computer Society, 2001, pp. 209–220, ISBN: 0-7695-1281-X. DOI: 10.1109/LICS.2001.932498. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/LICS.2001.932498.

[Ter06] K. Terui, *Intersection types for computational complexity*, Slides, 2006. [Online]. Available: citeseer.ist.psu.edu/294754.html.

[Ter07] K. Terui, "Light affine lambda calculus and polynomial time strong normalization", *Arch. Math. Log.*, vol. 46, no. 3-4, pp. 253–280, 2007. DOI: 10.1007/s00153-007-0042-6. [Online]. Available: http://dx.doi.org/10.1007/s00153-007-0042-6.

[Val01] S. Valentini, "An elementary proof of strong normalization for intersection types", *Arch. Math. Log.*, vol. 40, no. 7, pp. 475–488, 2001. DOI: 10.1007/s001530000070. [Online]. Available: http://dx.doi.org/10.1007/s001530000070.

[Yan11] H. Yang, Ed., *Programming Languages and Systems - 9th Asian Symposium, APLAS 2011, Kenting, Taiwan, December 5-7, 2011. Proceedings*, vol. 7078, ser. Lecture Notes in Computer Science, Springer, 2011, ISBN: 978-3-642-25317-1. DOI: 10.1007/978-3-642-25318-8. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25318-8.