UNIVERSITÀ DEGLI STUDI DI TORINO

DIPARTIMENTO DI INFORMATICA

SCUOLA DI SCIENZE DELLA NATURA

Dottorato di ricerca in
Computer Science
XXXIV Ciclo

The computational core: reduction theory and intersection type discipline

Tesi presentata da
Riccardo Treglia

Supervisore
Prof. Ugo de'Liguoro

Coordinatrice del Dottorato: Prof.ssa Viviana Patti

Anni accademici: 2019-2021

Settore scientifico-disciplinare di afferenza: INF/01 INFORMATICA

Tibimet cuius manum in somnis tenebam.
Magistris qui me docuerunt.
Mihi qui non cerno.

Acknowledgement

The majority of what is written in this Ph.D. thesis, as well as my personal growth over the last few years, could not have been accomplished without the support of my supervisor, Ugo de'Liguoro. As a young researcher, I was fortunate to have met a professor of his stature, and I am grateful to him for spending so much time teaching me what he knows, for supporting me, and for letting me accompany him. The honor and pride I feel to have had him as my mentor is beyond words.

My sincerest appreciation goes to the Department of Computer Science at the University of Turin for allowing me to pursue my Ph.D., especially emeritus Professors Dezani and Ronchi della Rocca, whose lessons remain indelible memories of my training. I thank the whole lambda group, especially Professors Padovani and Berardi for the useful discussions.

A great deal of luck also came to me on my journey, as I met Prof. Claudia Faggian. She acted as a second supervisor for me. For patiently teaching me, I am indebted to her. Moreover, I would like to say thank you to Giulio Guerrieri, a tenacious researcher whom I admirably respect. Both, Claudia and Giulio, have been a constant source of inspiration and improvement, all the syntactic analysis that follows would never have taken place without them.

It was truly a privilege to visit Professors Hasuo and Katsumata at the NII in Tokyo. Unfortunately, due to the outcome of the Coronavirus, that experience had to end prematurely, but it gave me a deep understanding of some topics, especially thanks to Jérémy Dubut.

I would like to thank those who have been present at my walk, and I apologize if I have not been able to live up to their personalities.

Abstract

Moving from the abstract definition of monads, we introduce a version of the call-by-value computational λ-calculus based on Wadler's variant, without *let*, and with unit and bind operators. We call the calculus *computational core* and study its reduction, and prove it confluent. In particular, the reduction rules are the relation obtained by orienting the monadic laws from left to right. The rules induced by associativity and identity make the behaviour of the reduction, and the study of its operational properties, non- trivial. This happens in the setting of any monadic lambda-calculus, independently of the syntactic representation of the calculus that internalizes them. Hence, the focus of our operational analysis is on two crucial properties: returning a value and having a normal form. The cornerstone of our analysis is factorization results. Weak factorization can be achieved by considering the surface reduction relation, a contextual closure of calculi based on linear logic. We expose the operational role of the rules associated with the monadic laws of identity and associativity. We then analyze the property of having a normal form (normalization), and then a family of normalizing strategies, i.e. sub-reductions that are guaranteed to reach a normal form, if any exists. To deal with that we rely on a quantitative analysis of the number of β_c-steps.

In a second part, we study a Curry style type assignment system for the computational core. We introduce an intersection type system inspired by Barendregt, Coppo, and Dezani system for ordinary untyped λ-calculus, establishing type invariance under conversion. Finally, we introduce a notion of convergence, which is precisely related to reduction, and characterize convergent terms via their types.

In the last part, we study the semantics of an untyped lambda-calculus equipped with operators representing read and write operations from and to a global store. We adopt the monadic approach to model side-effects and treat read and write as algebraic operations over a monad. We introduce an operational semantics.

The intersection type assignment system is indeed derived by solving a suitable domain equation in the category of omega-algebraic lattices; the solution consists of a filter-model generalizing the well known construction for ordinary lambda-calculus. Then the type system is obtained out of the term interpretations into the filter-model itself. The so obtained type system satisfies the "type-semantics" property (the semantics of a term is precisely the set of types that can be assigned to it) by construction.

Finally, we prove that types are invariant under reduction and expansion of term and state configurations, and characterize convergent terms via their typings.

CONTENTS

PREFACE

An näive addressing This preface is supposed to be a preamble to the technical contents that follow, without having any ambition of formality. On the contrary, in the following pages, we essentially present the doubts, questions, and objectives that ground the thesis work. This is the result of three years of research on how to frame the investigation of the properties of *monadic* programs using intersection type theory. Soon, in the course of the speculation, it will be realised that the formalisation of an intersection type theory that could extend the results already existing in the literature has to pass through a metalanguage that is as minimal as possible but able to encapsulate the requirements for being monadic. So here it is the first, main objective: extending results that were conceived for functional programs into the word of non-functional programs, as framed via the notion of *monads*. Let us first dwell on what it means to be monadic, which will be coupled with other terms such as "with effects" or *computational*. Each of these synonyms stresses the *impure* component of the program itself, but from slightly different angles. But let us go step by step.

From functional to non-functional. Monads are pervasive structures in many fields, especially those between computer science and mathematics, and, in particular, they are a fundamental concept in extending functional programming. Functional programming is a paradigm that treats computations as the evaluation of mathematical functions, avoiding mutable data and changing-state. For each input, a certain output is intended, without caring about the world around. Functional programs will work the same way in every place and time, and will only rely on themselves.

Let us call *effects* the intercommunications with the outside world. Through effects, functional reality is augmented by means of additional structures and operations to create and manage them. Roughly speaking, the codomain of functions is embellished by means of these additional structures.

What kind of effects are we talking about? An instance is the ability to handle **exceptions**, i.e. messages that are raised if something goes wrong and allowing to decide how to proceed. Another simple and preliminary example includes writing and/or reading strings from a screen (**interactive input/output**), or, in general terms, the use of an auxiliary structure, a **store**, to save information and to extract it when necessary. Other kinds of computational effects are **nondeterminism**, that is the possibility to create more branches from a single input, or a **probabilistic choice**, that is giving a weight to each of these branches.

For this reason, functional programming is very sensitive to effects: if you add them to the theory, you soon realise that there is a need for a more general and elegant framework than a mere case-by-case description of them.

Monads are the key concept to satisfy this general drive. A monad in functional programming is seen as a type constructor, i.e. a bridge that takes programs from the functional world into the enriched, non-functional world, all of course regulated by specific monadic laws. This makes it possible to control effects uniformly without the need to write *ah hoc* code. Monads has been incorporated in functional programming languages like Haskell in order to handle non-functional aspects in a purely functional language.

Monads. The concept of a monad was first introduced by Godement in 1958 (although, more precisely, he discussed comonads in [God58]). Godement was studying sheaf theory, which is a way to capture local data about a manifold and, in doing so, obtaining global properties. His definition was also named *triple*, which is still used nowadays when it comes to the the definition of *Kleisli triple*. The concept was later renamed "monad" by Mac Lane [Mac97], because of the analogy with monoids. Monads are frequent in many branches of mathematics: from universal algebra (very interesting is the connection with Lawvere theories, see [HP07]) to mathematical analysis (since Cauchy completions are monads). Monads also gave birth to a new way to describe non-functional programming in theoretical computer science, since Moggi's work in the late 80's. In the technical report [Mog88] and in subsequent papers [Mog89, Mog91], Moggi gave a unified account of computational effects as monads.

Extending the λ-calculus. The λ-calculus is collection of formal theories whose definition is given by three constructors and a single computational rule, namely β-reduction; let us call this the "standard" or *pure* λ-calculus. The grammar of the λ-calculus allows to write pure programs only, hence the connection with functional programming, i.e. programs that can be described as mathematical functions. λ-calculus originated from certain systems of combinatory logic that were conceived around 1930 as a foundation of mathematics by Church and Curry. Nowadays, the "bible" of pure λ-calculus is certainly [Bar85]. When one moves to functions with embellished codomain, as in the monadic setting, one must also try to mirror this by enriching the calculus syntax and, consequently, by adding computational rules to manage effects: hence the name *impure*. This dichotomy is well quoted at the beginning of one of Wadler's paper [Wad95]: "shall I be pure or impure?". In fact, two approaches can actually take place: extend the pure paradigm with extra features or accept to be impure *tout court*. The latter approach will be the one adopted by us, motivated by the fact that in Kleisli categories there is no distinction between pure and impure functions, as every function has an embellished codomain. By the way, in [Wad95] the reader will find another cornerstone for the investigation in this thesis. Indeed, our calculus, that we baptize *computational core* has more affinity to Wadler's formulation of monads than to the computational λ-calculus by Moggi, although both are semantically equivalent.

Operational analysis. Reduction and operational semantics of the computational λ-calculus have been studied in the context of call-by-need calculi, e.g. in [MOTW99, AFM$^+$95], and confluence of reduction of Moggi's λ_C has been established recently in [Ham18], essentially depending on the strong normalization property of the calculus. However the calculus we consider here is untyped, and not strongly normalizing, syntactically different, simpler, and strictly speaking not a sub-calculus of any of its ancestors, as it does not have neither the *let*, nor functional application as primitive constructs. By the way, the calculus we are to propose should not be labelled as a particular, quirky calculus, at all. In fact, its theory of reduction shares essential properties with other well established calculi in the literature. Some connections with different calculi are detailed, but there would be many others with which it would be worthy to undertake an analysis and comparison, both syntactic and semantic. In fact, the study of reduction theory of the computational core is not only aimed to investigate its properties, but also to stress the similarities with other systems and shed new light on them. In addition, we stress the didactic approach to the analysis of the computational core reduction theory: the reader will notice that the study of factorization, confluence, and normalization is carried out with different techniques, from the most powerful ones such as van Oostrom's decreasing diagrams [vO94] to the more classical ones such as Takahashi's parallel reduction [Tak95]. This multiplicity of approaches represents not only a learning experience, but also a set of techniques that can be put into practice in the study of calculi comparable to the one presented here.

Intersection type theory. Intersection types were introduced in the late 70's by Dezani and Coppo [CDC80] to overcome the limitations of Curry's type discipline and enlarge the class of terms that can be typed. This is reached by means of a new type constructor, the *intersection*. A complete introduction to intersection type disciplines can be found in [BDS13]. From there, we will adopt the BCD intersection type system, hence our systems will be equipped with a subtyping pre-order with a distinguished top element. The historical fortune of this type system is certainly its expressiveness, i.e. allowing to capture various properties of λ-terms that had escaped all previous typing systems.

One of the motivations for investigating intersection types is that, when including a universal type (usually denoted by ω) that can be assigned to any term, types are invariant under term conversion, instead of just reduction. By this property, the term meaning, namely its functional behaviour, is fully characterized by the set of its types. So, establishing such a system for the computational core, opens the way to study the case of (untyped) λ-calculi with effects by well established mathematical tools.

Intersection type disciplines allows to theoretically characterize weakly normalizing terms, for instance, and also in this thesis we will reach a similar characterization. In a nutshell, intersection types are able to strictly connect the reduction behaviour of programs with their denotational semantics: we will see how through our theories of intersection types we can qualify the terms that converge to a value, and in the case of the store monad we will also be able to characterise these through a particular type.

Another interesting aspect of intersection type disciplines is their idiosyncratic semantic flavour. Intersection types have a semantics based on *duality*, which is related to (actually is a restriction of) Abramsky's domain theory in logical form [Abr91]. By this duality, type theories give rise to *filter λ-models*. Intersection type assignment systems can then be viewed as finitary descriptions of the interpretation of terms in such models, hence the meaning of a term is the set of types that can be assigned to it. The filter-model presented in [BCD83] is a new λ-model which is useful to study completeness properties of type assignment systems for terms of the pure λ-calculus.

It turns out that filter-models have the structure of an algebraic lattice with countable basis, where types are in one-to-one correspondence with the compact points of such a domain.

Indeed, each inverse limit space obtained by the classical Scott's D_∞ construction [Sco80], starting from a countably based lattice, is isomorphic to a filter-model; a comprehensive reference is certainly [AS08]. In this thesis we will deal with a filter-model that is isomorphic to a variant of the construction of D_∞, where the monad functor is involved in the domain equation.

In this thesis we intensively use such models, and in Part III we show how to derive a type assignment system for an imperative lambda calculus (that is an extension of the computational core) out of its D_∞ model.

Thesis organization. This PhD thesis consists in three main parts. Each part is provided with its own thematic introduction and is concluded by a discussion of the results presented and a reference to the existing literature. This is done to keep the parts stand-alone as much as feasible, exploiting the incremental nature of the work. In the course of the thesis, in the introduction and conclusion of each part, time and space will be spent to provide most appropriate references to the topics. Since several macro-areas are touched upon, such as monads, reduction theory, and intersection types, it is not appropriate to condense the entire state of the art into a single section, like a large and varied grab bag.

Part I deals with the reduction theory of the computational core: this part first presents preliminaries on the lambda calculus and rewriting theory, but also on monads, because framing of the calculus has a strong foundation in the intended semantics.

Part II presents an intersection type system for the computational core, leading to the result of completeness. The most important result of the second part is certainly the characterisation of the convergent terms.

This result is also mirrored in Part III, where the subject of study is an extension of the computational core, namely a monadic calculus whose monad is the store monad. This part is intended to be a case study in which the generic monad is instantiated. In this case the type system is not presented *ex abrupto* but is elegantly derived from the filter-model that is obtained by solving a suitable domain equation.

The preliminaries presented in each part, for example aspects of domain theory, type theory, and other needed knowledge, are not described in detail. In fact, this thesis has no ambition to be introductory or a good manual on basics, for which historical and better established texts are cited.

PART I

CHAPTER 1

INTRODUCTION AND PRELIMINARIES

1.1 Introduction to the Calculus

The λ-calculus has been historically conceived as an equational theory of functions, so that reduction had an ancillary role in Church's view, and it was a tool for studying the theory β, see [Bar85, Ch. 3]. The development of functional programming languages like Lisp and ML, and of proof assistants like LCF, has brought a new, different interest in the λ-calculus and its *reduction* theory.

The cornerstone of this change in perspective is Plotkin's [Plo75], where the functional parameter passing mechanism is formalized by the *call-by-value rule* β_v, allowing contraction only if the argument term is a *value*, that is a variable or an abstraction. In [Plo75] it is also introduced the notion of *weak evaluation*, namely no reduction in the body of a function (aka, an abstraction). This is now the standard evaluation implemented by functional programming languages, where *values* are the terms of interest (and the normal forms for weak evaluation in the closed case). Full β_v reduction is instead the basis of proof assistants like Coq, where *normal forms* are the result of interest. More generally, the computational perspective on λ-calculus has given a central role to reduction, whose theory provides a sound framework for reasoning about program transformations, such as compiler optimizations or parallel implementations.

The rich variety of computational effects in actual implementations of functional programming languages brings further challenges. This dramatically affects the theory of reduction of the calculi formalizing such features, whose proliferation makes it difficult to focus on suitably general issues. A major change here is the discovery by Moggi [Mog91] of a whole family of calculi that are based on a few common treats, combining call-by-value with the abstract notion of effectful computation represented by a *monad*, that has shown to be quite successful. But Moggi's *computational λ-calculus* is an equational theory in the broader sense; much less is known of the reduction theory of such calculi: this is the main purpose of the current chapter.

The computational calculus. Since the seminal work [Mog91], computational λ-calculi have been developed as foundations of programming languages, formalizing both functional and non-functional features: see e.g. [WT03, BHM02] to mention two, starting a thread in the literature that is still growing. The basic idea of computational λ-calculi is to distinguish *values* and *computations*, so that programs, represented by closed terms, are thought of as functions from values to computations. Intuitively, computations embody richer structure than values and do form a larger set in which values can be embedded. On the other hand, the essence of programming is composition; to compose functions from values to computations we need a mechanism to uniformly extend them to functions *on computations*, while preserving their original behaviour over the (image of) values.

To model these concepts, Moggi used the categorical notion of *monad*, abstractly representing the extension of the space of values to that of computations, and the associated Kleisli category, whose morphisms are functions from values to computations, which are the denotations of programs. Syntactically, following [Wad95], we can express these ideas by means of a call-by-value λ-calculus with two sorts of terms: *values*, ranged over by V, W, namely variables or abstractions, and *computations* denoted by L, M, N. Computations are formed by means of two operators: values are embedded into computations by means of the operator $[\cdot]$ written `return` in Haskell programming language, whose name refers to the unit of a monad in categorical terms; a computation $M \star (\lambda x.N)$ is formed by the binary operator $\star$, called *bind* (>>= in Haskell), representing the application to M of the extension to computations of the function $\lambda x.N$.

The monadic laws. The operational understanding of these new operators is that evaluating $M \star (\lambda x.N)$, which in Moggi's notation reads $\mathsf{let}\, x := M \,\mathsf{in}\, N$, amounts to first evaluate M until a computation of the form $[V]$ is reached, representing the trivial computation that returns the value V. Then V is passed to N by binding x to V, as expressed by the equation:

$$[V] \star \lambda x.N = N\{V/x\} \tag{1.1}$$

This is the first of the three *monadic laws* in [Wad95]. To understand the others, let us define the composition of the functions $\lambda x.M$ and $\lambda y.N$ as

$$(\lambda x.M) \bullet (\lambda y.N) := \lambda x.(M \star (\lambda y.N))$$

where we can freely assume that x is not free in N; this is named Kleisli composition in category theory. Now the remaining laws are:

$$M \star \lambda x.[x] = M \tag{1.2}$$
$$(L \star \lambda x.M) \star \lambda y.N = L \star \lambda x.(M \star \lambda y.N) \quad \text{with } x \notin \mathsf{fv}(N) \tag{1.3}$$

Equality (1.2) (*identity*) implies that $(\lambda z.M) \bullet (\lambda x.[x]) = \lambda z.M$, which paired with the instance of (1.1): $(\lambda x.[x]) \bullet (\lambda y.N) = \lambda x.N\{x/y\} =_\alpha \lambda y.N$ (where $=_\alpha$ is the usual congruence generated by the renaming of bound variables), tells that $\lambda x.[x]$ is the identity of composition $\bullet$.

Equality (1.3) (*associativity*) implies:

$$((\lambda z.L) \bullet (\lambda z.M)) \bullet (\lambda y.N) = (\lambda z.L) \bullet ((\lambda z.M) \bullet (\lambda y.N))$$

namely that composition $\bullet$ is associative.

The monadic laws correspond to the three equalities in the definition of a Kleisli triple (see [Mog91]), that is an equivalent presentation of monads [Mac97]. Indeed, the calculus in [Mog91] is the internal language of a suitable category equipped with a (strong) monad T, and with enough structure to internalize the morphisms of the respective Kleisli category. As such, it is a simply typed λ-calculus, where T is the type constructor associating to each type A the type TA of computations over A. Therefore, $[\cdot]$ and $\star$ are polymorphic operators with respective types (see [Wad92, Wad95]):

$$[\cdot] : A \to TA \qquad\qquad \star : TA \to (A \to TB) \to TB \tag{1.4}$$

The computational core. The dynamics of λ-calculi is usually defined as a reduction relation on untyped terms. Moggi's preliminary report [Mog88] specifies both an equational and, in §6, a *reduction* system even if only the former is thoroughly investigated and appears in [Mog91], while reduction is briefly treated for an untyped fragment of the calculus. However, when stepping from the typed calculus to the untyped one, we need to be careful and avoid meaningless terms to creep into the syntax, so jeopardizing the calculus theory. For example: what should be the meaning of $M \star N$ where both M and N are computations? What about $(\lambda x.N) \star V$ for any V? Shall we have functional applications of any kind?

To answer these questions, in [dT20], as well as in Chapter 2, typability is taken as syntactic counterpart of being meaningful: inspired by ideas in [Sco80], the untyped computational λ-calculus is a special case of the typed one, where there are just two types D and TD, related by the type equation $D = D \to TD$, that is Moggi's isomorphism of the call-by-value reflexive object (see [Mog88], §5). With such a proviso, we get the following syntax:

$$\begin{aligned} V, W &::= x \mid \lambda x.M && (\mathit{Val}) \\ M, N, L &::= [V] \mid M \star V && (\mathit{Com}) \end{aligned}$$

If we assume that all variables have type D, then it is easy to see that all terms in *Val* have type $D = D \to TD$, which is consistent with the substitution of variables with values in (1.1). On the other hand, considering the typing of $[\cdot]$ and $\star$ in (1.4), terms in *Com* have type TD. As we have touched above, there is some variety in notation among computational λ-calculi; we choose the above syntax because it explicitly embodies the essential constructs of a λ-calculus with monads, but for functional application, which is definable: see Chapter 2 for further explanations. We dub the calculus *computational core*, noted $\lambda_{©}$.

From equalities to reduction. Similarly to [Mog88] and [SW97], the reduction rules in the computational core $\lambda_{©}$ are the relation obtained by orienting the monadic laws from left to right. We indicate by β_c, id, and σ the rules corresponding to (1.1), (1.2) and (1.3), respectively. The compatible closure of these rules, denoted $\to_{\copyright}$, has been proved confluent in [dT20], a proof reported here in Section 4.1, which implies that equal terms have a common reduct and the uniqueness of normal forms.

In [Plo75] call-by-value reduction $\to_{\beta_v}$ is an intermediate concept between the equational theory and the evaluation relation $\underset{\mathsf{w}}{\to}_{\beta_v}$, that models an abstract machine. Evaluation consists of persistently choosing the leftmost β_v-redex that is not in the scope of an abstraction, *i.e.* evaluation is *weak*. The following crucial result bridges the reduction (hence, the foundational calculus) with the evaluation (implemented by an ideal programming language):

$$M \to^*_{\beta_v} V \text{ (for some value } V\text{) if and only if } M \underset{\mathsf{w}}{\to}^*_{\beta_v} V' \text{ (for some value } V'\text{)} \qquad (1.5)$$

Such a result (Corollary 1 in [Plo75]) comes from an analysis of the *reduction* properties of $\to_{\beta_v}$, namely standardization.

As we will see, the rules induced by associativity and identity make the behaviour of the reduction—and the study of its operational properties—*non-trivial* in the setting of any monadic λ-calculi. The issues are inherent to the rules coming from the monadic laws (1.2) and (1.3), independently of the syntactic representation of the calculus that internalizes them. The difficulty appears clearly if we want to follow a similar route to [Plo75], as we discuss next.

Reduction vs evaluation. Following [Fel88], reduction $\to_{\copyright}$ and evaluation $\underset{\mathsf{w}}{\to}_{\copyright}$ of $\lambda_{\copyright}$ can be defined as the closure of the reduction rules under arbitrary and under evaluation contexts, respectively. Consider the following grammars:

$$\begin{array}{ll} \mathsf{C} ::= \langle\,\rangle \mid [\lambda x.\mathsf{C}] \mid \mathsf{C} \star V \mid M \star (\lambda x.\mathsf{C}) & \qquad\text{(arbitrary) contexts} \\ \mathsf{E} ::= \langle\,\rangle \mid \mathsf{E} \star V & \qquad\text{evaluation contexts} \end{array}$$

where the hole $\langle\,\rangle$ can be filled by terms in *Com*, only. Observe that the closure under evaluation context E is precisely weak reduction.
Weak reduction of $\lambda_{\copyright}$, however, turns out to be non-deterministic, non-confluent, and its normal forms are *not unique*. The following is a counter-example to all such properties—see Section 4.2 for further examples.

$$\begin{tikzcd} (([z] \star z) \star \lambda x.\, M) \star \lambda y.\, [y] \arrow[r, "\mathsf{w}"'] \arrow[d, "\mathsf{w}"] & ([z] \star z) \star \lambda x.\, (M \star \lambda y.\, [y]) \\ ([z] \star z) \star \lambda x.\, M & \end{tikzcd}$$

Such an issue is not specific to the syntax of the computational core. The same phenomena show up with the *let*-notation. Evaluation contexts are now generated by

$$\mathsf{E}_{let} ::= \langle\,\rangle \mid \mathsf{let}\, x \,{:=}\, \mathsf{E}_{let}\, \mathsf{in}\, N$$

and examples similar to the one above can be reproduced. We give the details in Example 4.2.3.

Content and contributions. The focus of this part is an *operational* analysis of two crucial properties of a term M:

i M returns a *value* (*i.e.* $M \rightarrow_{\copyright}^{*} [V]$, for some V value).

ii M has a *normal form* (*i.e.* $M \rightarrow_{\copyright}^{*} N$, for some normal N).

The cornerstones of our analysis are *factorization* results (also called *semi-standardization* in the literature): any reduction sequence can be re-organized so to first performing specific steps and then everything else.

Via factorization, we establish the following key result, analogous to (1.5), relating reduction and *evaluation*:

$$M \rightarrow_{\copyright}^{*} [V] \text{ (for some value } V) \iff M \underset{\mathsf{w}}{\rightarrow}_{\beta_c}^{*} [V'] \text{ (for some value } V') \tag{1.6}$$

We then analyze the property of having a normal form (*normalization*), and define a family of *normalizing strategies*, *i.e.* sub-reductions that are guaranteed to reach a normal form, if it exists.

We already discussed *evaluation.* Let us discuss more in detail some of the other aspects.

Factorization. While $\rightarrow_{\copyright}$ cannot be factorized w.r.t. weak reduction, factorization can be achieved by considering the *surface* reduction relation, which is less constrained and better behaved than weak reduction. It disallows reduction under the $[\cdot]$ operator, only. Intuitively, weak reduction does not act in the body of a function, while surface reduction does not act in the scope of `return`. The name surface is reminiscent of a similar notion in calculi based on linear logic [Sim05, EG16], and actually we will see (Section 3.2.1) that there is a correspondence, with the $[\cdot]$ operator behaving like a box.

Identity and associativity. Our analysis exposes the operational role of the rules associated to the monadic laws of identity and associativity.

i To compute a *value*, only β_c steps are necessary.

ii To compute a *normal form*, β_c steps do not suffice: *associativity* (*i.e.* σ steps) is *necessary.*

Normalization. The study of normalization is more complex than that of evaluation, and requires some sophisticated techniques. We highlight some specific contributions.

- We define two families of normalizing strategies in $\lambda_{\copyright}$. The first one, quite constrained, relies on an *iteration of weak reduction* $\underset{\mathsf{w}}{\rightarrow}_{\lambda_{\copyright}}$. The second one, more liberal, is based on an *iteration of surface reduction* $\underset{\mathsf{s}}{\rightarrow}_{\lambda_{\copyright}}$. The definition and proof of normalization is *parametric* on both.

- The technical *difficulty* in the proofs related to normalization comes from the fact that neither weak nor surface reduction is deterministic. To deal with that we rely on a fine *quantitative analysis* of the number of β_c steps, which we carry-on when we study factorization in Section 4.3.

Some of the most challenging proofs in this part are those concerned with normalization via surface reduction. The effort is justified by the interest in having a larger and *more versatile* strategy. The definition of strategy is not an actual abstract machine but *subsumes* several ones, each following a different reduction policies. It thus facilitates reasoning about optimization techniques and parallel implementation.

A roadmap. Let us summarize the structure of this part.

Section 1.2 contains the background notions which are relevant to our concerning.

Chapter 2 gives the **formal definition of the computational core** $\lambda_{©}$, via the categorical definition of computational monad, and its reduction.

In Section 3.2 and Section 4.2, we analyze the **properties of weak and surface reduction**. We first study $\rightarrow_{\beta_c}$, and then we move to the whole $\lambda_{©}$, where associativity and identity also come to play. In Section 4.1 we deal with the proof of confluence of the calculus. In Section 4.3 we study several **factorization** results. The cornerstone of our construction is surface factorization (Theorem 4.3.1). We then further refine this result, first by postponing the id steps which are not β_c steps, and then with a weak factorization. This further phase is motivated by the properties which we analyze in Section 4.2.

In Section 4.4 we study **evaluation**, and analyze some relevant consequences of this result. We actually provide two different ways to deterministically compute a value. The first is the one given by Equation (1.6), via an *evaluation context*. The second way requires no contextual closure at all: simply applying β_c and σ *rules* will return a value, if possible.

In Chapter 5 we study **normalization and normalizing strategies**. The conclusions and related works are presented in this part last chapter, Chapter 6.

1.2 Preliminaries

1.2.1 Basics in Rewriting

In this section we recall some standard definitions and notations in rewriting (see for instance Terese [Ter03] or Baader and Nipkow [BN98]).

Rewriting System. An *abstract rewriting system (ARS)* is a pair $(A, \rightarrow)$ consisting of a set A and a binary relation $\rightarrow$ on A whose pairs are written $t \rightarrow s$ and called *steps*. A $\rightarrow$*-sequence* from t is a sequence of $\rightarrow$-steps. We denote by $\rightarrow^*$ (resp. $\rightarrow^=$; $\rightarrow^+$) the transitive-reflexive (resp. reflexive; transitive) closure of $\rightarrow$, and use $\leftarrow$ for the reverse relation of $\rightarrow$, that is, $u \leftarrow t$ if $t \rightarrow u$. We write $t \rightarrow^k s$ for a $\rightarrow$-sequence $t \rightarrow t_1 \rightarrow \ldots \rightarrow t_k = s$ of $k \in \mathbb{N}$ steps. If $\rightarrow_1, \rightarrow_2$ are binary relations on A then $\to_1 \cdot \to_2$ denotes their composition, *i.e.* $t \to_1 \cdot \to_2 s$ if there exists $u \in A$ such that $t \to_1 u \to_2 s$. We often set $\to_{12} := \to_1 \cup \to_2$.

A relation $\to$ is *deterministic* if for each $t \in A$ there is at most one $s \in A$ such that $t \to s$. It is *confluent* if $\leftarrow^* \cdot \to^* \subseteq \to^* \cdot \leftarrow^*$.

We say that $u \in A$ is $\to$*-normal* (or a $\to$-normal form) if there is no t such that $u \to t$. Confluence implies that each $t \in A$ has *unique normal form*, if any exists.

Normalization. Let $(A, \to)$ be an ARS. In general, a term may or may not reduce to a normal form. And if it does, not all reduction sequences necessarily lead to normal form. A term is *weakly* or *strongly normalizing*, depending on if it may or must reduce to normal form. If a term t is strongly normalizing, any choice of steps will eventually lead to a normal form. However, if t is weakly normalizing, how do we compute a normal form? This is the problem tackled by *normalization*: by repeatedly performing *only specific steps*, a normal form will be computed, provided that t can reduce to any. We recall three important notions of normalization:

Definition 1.2.1 (Normalizing)**.**

1. *t is* strongly $\to$-normalizing *(or **terminating**): every maximal $\to$-sequence from t ends in a normal form (*i.e.*, t has no infinite $\to$-sequence).*
2. *t is* weakly $\to$-normalizing *(or just **normalizing**): there exist a $\to$-sequence from t which ends in a normal form.*

*A reduction $\to$ is strongly (resp. weakly) normalizing if each $t \in A$ is. A reduction $\to$ is **uniformly normalizing** if its weakly normalization implies its strongly normalization.*

The restriction to a subreduction $\xrightarrow[\mathsf{e}]{} \subseteq \to$ is a way to control that in a term there are different possible choices of reduction. A *normalizing strategy* for $\to$ is a reduction strategy which, given a term t, is guaranteed to reach its $\to$-normal form, if any exists.

Definition 1.2.2 (Normalizing strategies)**.** *A reduction $\xrightarrow[\mathsf{e}]{} \subseteq \to$ is a **normalizing strategy** for $\to$ if $\xrightarrow[\mathsf{e}]{}$ has the same normal forms as $\to$ whenever t has a $\to$-normal form. As a consequence,* every *maximal $\xrightarrow[\mathsf{e}]{}$-sequence from t ends in a normal form.*

Factorization. In this part, we will extensively use factorization results.

Definition 1.2.3 (Factorization)**.** *Let $(A, \to)$ be a rewriting system with $\to = \xrightarrow[\mathsf{e}]{} \cup \xrightarrow[\mathsf{i}]{}$. The relation $\to$ satisfies* e-factorization*, written* $\mathtt{Fact}(\xrightarrow[\mathsf{e}]{}, \xrightarrow[\mathsf{i}]{})$*, if*

$$\mathtt{Fact}(\xrightarrow[\mathsf{e}]{}, \xrightarrow[\mathsf{i}]{}): \quad (\xrightarrow[\mathsf{e}]{} \cup \xrightarrow[\mathsf{i}]{})^* \subseteq \xrightarrow[\mathsf{e}]{}^* \cdot \xrightarrow[\mathsf{i}]{}^* \qquad (\textbf{Factorization})$$

*The relation $\xrightarrow[\mathsf{i}]{}$ **postpones** after $\xrightarrow[\mathsf{e}]{}$, written* $\mathtt{PP}(\xrightarrow[\mathsf{e}]{}, \xrightarrow[\mathsf{i}]{})$*, if*

$$\mathtt{PP}(\xrightarrow[\mathsf{e}]{}, \xrightarrow[\mathsf{i}]{}): \quad \xrightarrow[\mathsf{i}]{}^* \cdot \xrightarrow[\mathsf{e}]{}^* \subseteq \xrightarrow[\mathsf{e}]{}^* \cdot \xrightarrow[\mathsf{i}]{}^*. \qquad (\textbf{Postponement})$$

It is an easy result that e-factorization is equivalent to postponement, which is a more convenient way to express it.

Lemma 1.2.4. *The following are equivalent (for any two relations* $\xrightarrow[\mathsf{e}]{}, \xrightarrow[\mathsf{i}]{}$*)*

1. Postponement*:* $\mathtt{PP}(\xrightarrow[\mathsf{e}]{}, \xrightarrow[\mathsf{i}]{})$.
2. Factorization*:* $\mathtt{Fact}(\xrightarrow[\mathsf{e}]{}, \xrightarrow[\mathsf{i}]{})$.

Hindley [Hin64] first noted that a local property implies factorization. Let $\to = \xrightarrow[\mathsf{e}]{} \cup \xrightarrow[\mathsf{i}]{}$. We say that $\xrightarrow[\mathsf{i}]{}$ *strongly postpones* after $\xrightarrow[\mathsf{e}]{}$, if

$$\mathtt{SP}(\xrightarrow[\mathsf{e}]{}, \xrightarrow[\mathsf{i}]{}): \qquad \xrightarrow[\mathsf{i}]{} \cdot \xrightarrow[\mathsf{e}]{} \ \subseteq \ \xrightarrow[\mathsf{e}]{}^{*} \cdot \xrightarrow[\mathsf{i}]{}^{=} \qquad (\textbf{Strong Postponement})$$

Lemma 1.2.5 (Hindley [Hin64])**.** $\mathtt{SP}(\xrightarrow[\mathsf{e}]{}, \xrightarrow[\mathsf{i}]{})$ *implies* $\mathtt{Fact}(\xrightarrow[\mathsf{e}]{}, \xrightarrow[\mathsf{i}]{})$.

Observe that the following are special cases of Strong Postponement. The first is *linear* in $\xrightarrow[\mathsf{e}]{}$. We refer to it as *linear postponement.*

1. $\xrightarrow[\mathsf{i}]{} \cdot \xrightarrow[\mathsf{e}]{} \ \subseteq \ \xrightarrow[\mathsf{e}]{} \cdot \xrightarrow[\mathsf{i}]{}^{=}$
2. $\xrightarrow[\mathsf{i}]{} \cdot \xrightarrow[\mathsf{e}]{} \ \subseteq \ \xrightarrow[\mathsf{e}]{} \cdot \to$

Linear variants of postponement can easily be adapted to *quantitative* variants, which allow us to "count the steps", and will be useful to establish termination properties. We do this in Section 4.3.3.

Diamonds. We recall also another *quantitative* result, which we will use.

Fact 1.2.6 (Newman [New42])**.** *Given an ARS* $(A, \to)$*, (1) implies (2).*

1. ***Quasi-diamond****:* $\forall t \in A$*,* $(t_1 \leftarrow t \to t_2)$ *implies* $(t_1 = t_2$ *or exists* u *such that* $t_1 \to u \leftarrow t_2)$*.*
2. ***Random Descent****:* $\forall t \in A$*, all maximal sequences from* t *have the* same number of steps*, and all end in the same normal form, if any exists.*

If a reduction is quasi-diamond, then it has Random Descent.

Postponement, Confluence and Commutation Both postponement and confluence are commutation properties. Two relations $\triangleright$ and $\blacktriangleright$ on A *commute* if

$$\triangleleft^{*} \cdot \blacktriangleright^{*} \subseteq \blacktriangleright^{*} \cdot \triangleleft^{*} . \qquad (\textbf{Commutation})$$

A relation $\to$ on A is *confluent* if it commutes with itself. *Postponement and commutation* can also be defined in terms of each other, simply taking $\xrightarrow[\mathsf{i}]{}$ for $\triangleleft$ and $\xrightarrow[\mathsf{e}]{}$ for $\blacktriangleright$ ($\xrightarrow[\mathsf{i}]{}$ postpones after $\xrightarrow[\mathsf{e}]{}$ if and only $\xleftarrow[\mathsf{i}]{}$ commutes with $\xrightarrow[\mathsf{e}]{}$). As propounded in [vO20b], this fact allows for proving postponement by means of the decreasing diagrams technique [vO94]. This is a powerful and general tool to establish commutation properties, which reduces the problem of showing commutation to a local test; in exchange of localization, the diagrams need to be decreasing with respect to some labelling.

Definition 1.2.7 (Decreasing)**.** *Let* $\rhd := \bigcup_{k\in K} \rhd_k$ *and* $\blacktriangleright := \bigcup_{j\in J} \blacktriangleright_j$*. The pair of relations* $\rhd, \blacktriangleright$ *is decreasing if for some well-founded strict order on the set of labels* $L = K \cup J$ *the following holds, for each* $k \in K, j \in J$*:*

$$\lhd_k \cdot \blacktriangleright_j \quad \subseteq \quad (\blacktriangleright^*_{\langle k\rangle} \cdot \blacktriangleright^=_j \cdot \blacktriangleright^*_{\langle k,j\rangle}) \ \cdot \ (\lhd^*_{\langle j\rangle} \cdot \lhd^=_k \cdot \lhd^*_{\langle k,j\rangle}),$$

where $\langle L\rangle = \{i \in K \cup J \mid \exists l \in L.\ l > i\}$.

Theorem 1.2.8 (Decreasing Diagrams [vO94])**.** *If a pair of relations* $\rhd$, $\blacktriangleright$ *is decreasing, then* $\rhd$ *and* $\blacktriangleright$ *commute.*

Modularizing Confluence. A classic tool to modularize a proof of confluence is Hindley-Rosen lemma, stating that the union of confluent reductions is itself confluent if they all commute with each other.

Lemma 1.2.9 (Hindley-Rosen)**.** *Let* $\rightarrow_1$ *and* $\rightarrow_2$ *be relations on the set* A*. If* $\rightarrow_1$ *and* $\rightarrow_2$ *are confluent and commute with each other, then* $\rightarrow_1 \cup \rightarrow_2$ *is confluent.*

Like for postponement, strong commutation implies commutation.

Lemma 1.2.10 (Strong commutation [Hin64])**.** *Strong commutation* $\leftarrow_1 \cdot \rightarrow_2 \quad \subseteq \quad \rightarrow_2{}^* \cdot \leftarrow_1{}^=$ *implies commutation.*

1.2.2 Basics on λ-calculus

We recall the syntax of λ-terms, and some relevant notions about the λ-calculus, taking Plotkin's call-by-value (CbV) λ-calculus [Plo75] as a concrete example.

λ-terms and values are generated by the grammars

$$V \quad ::= \quad x \mid \lambda x.M \quad (\textit{values } Val) \qquad\qquad M, N ::= V \mid MN \quad (\textit{terms } \Lambda)$$

where x ranges over a countable set of *variables*. Terms of shape MN and $\lambda x.M$ are called *applications* and *abstractions*, respectively.

Reduction.

- **Contexts** (with one hole $\langle\ \rangle$) are generated by

  $$\mathsf{C} ::= \langle\ \rangle \mid M\mathsf{C} \mid \mathsf{C}M \mid \lambda x.\mathsf{C} \quad (\textit{Contexts})$$

  $\mathsf{C}\langle M\rangle$ stands for the term obtained from C by replacing the hole with the term M (possibly capturing free variables of M).

- A **rule** ρ is a binary relation on Λ, which we also denote $\mapsto_\rho$, writing $R \mapsto_\rho R'$. R is called a *ρ-redex.*

- A **reduction step** $\rightarrow_\rho$ is the closure under context C of ρ. Explicitly, if $T, S \in \textit{Term}$ then $T \rightarrow_\gamma S$ if $T = \mathsf{C}\langle R\rangle$ and $S = \mathsf{C}\langle R'\rangle$, for some context C and $R \mapsto_\gamma R'$.

The Call-by-Value λ-calculus. CbV λ-calculus is the rewrite system $(\Lambda, \to_{\beta_v})$, the set of λ-terms Λ, equipped with the β_v-reduction, where the reduction $\to_{\beta_v}$ is the contextual closure of the rule $\mapsto_{\beta_v}$:

$$(\lambda x.P)V \mapsto_{\beta_v} P\{V/x\} \quad (V \in \mathit{Val}).$$

Notice that here β-redexes can be fired only when the argument is a *value*.

Weak evaluation (which does not reduce in the body of a function) evaluates closed terms to values. In the literature about CbV, there are three main weak schemes: reducing from left to right, as defined by Plotkin [Plo75], from right to left [Ler90], or in an arbitrary order [LM08]. *Left* contexts L, *right* contexts R, and (arbitrary order) *weak* contexts W are, respectively, defined by

$$\mathsf{L} ::= \langle\ \rangle \mid \mathsf{L}t \mid v\mathsf{L} \qquad \mathsf{R} ::= \langle\ \rangle \mid t\mathsf{R} \mid \mathsf{R}v \qquad \mathsf{W} ::= \langle\ \rangle \mid \mathsf{W}t \mid t\mathsf{W}$$

In general, given a rule $\mapsto_\rho$ on Λ, *weak reduction* $\underset{\mathsf{w}}{\to}_\rho$ is the closure of $\mapsto_\rho$ under weak contexts W; *non-weak reduction* $\underset{\neg\mathsf{w}}{\to}_\rho$ is the closure of $\mapsto_\rho$ under contexts C that are not weak. *Left* and *non-left* reductions ($\underset{\mathsf{l}}{\to}_\rho$ and $\underset{\neg\mathsf{l}}{\to}_\rho$), *right* and *non-right* reductions are defined in a similar way.

CbV Weak Factorization. Factorization of $\to_{\beta_v}$ allows for a characterization of the terms which reduce to a value. Convergence below is a remarkable consequence of factorization.

Theorem 1.2.11 (Weak Left Factorization [Plo75])**.**

- Left Factorization of $\to_{\beta_v}$: $\quad \to_{\beta_v}^* \subseteq \underset{\mathsf{l}}{\to}_{\beta_v}{}^* \cdot \underset{\neg\mathsf{l}}{\to}_{\beta_v}{}^*$.
- Convergence: $\quad T \to_{\beta_v}^* V$ *if and only if* $T \underset{\mathsf{l}}{\to}_{\beta_v}{}^* V'$.

The same results hold for $\underset{\mathsf{w}}{\to}$ and $\underset{\mathsf{r}}{\to}_{\beta_v}$ in place of $\underset{\mathsf{l}}{\to}_{\beta_v}$.

Recalling that for closed terms the normal forms of $\underset{\mathsf{l}}{\to}$ are exactly the values, it means that given M a closed term, M has a β_v-reduction to a value, if and only if $\underset{\mathsf{l}}{\to}_{\beta_v}$-reduction from M terminates.

CHAPTER 2

THE COMPUTATIONAL CORE $\lambda_{©}$

The title of the following section might lead the reader to think that he has not yet entered the part of the text dedicated to reduction theory and, therefore, to a a purely syntactical investigation. Actually, we think that the most appropriate way to introduce the syntax of the computational core is through a description of the monads and the intended model, since these concepts will have a great influence on the design of the calculus itself. The “semantic” nature of the computational core syntax will ensure that the terms of the grammar are only those that are needed, i.e. avoiding redundancies, and reaching the requirement of *minimality* whenever possible. This is why the calculus is called *computational core*, precisely because it aims to be the heart of any calculus with effects.

2.1 Concrete Models of the Computational λ-calculus

The computational λ-calculus, denoted λ_C, has been introduced in the seminal works [Mog89, Mog91]. It is a typed calculus derived from the categorical construction of a monad (T, η, μ) (see [Mac97] chap. VI) over a cartesian category $\mathcal{C}$, equipped with some more structure to model Kleisli exponents, which represent internally the morphisms in $\mathcal{C}_T(A, B) = \mathcal{C}(A, TB)$, where $\mathcal{C}_T$ is the Kleisli category of the monad. For the precise definition see [Mog89], Defs. 3.2 and 3.5, or [Mog91] Defs. 3.2 and 3.9; see also Definition 2.1.2, Proposition 2.1.3, and Proposition 2.1.4 below.

As said before, in Moggi’s construction $\mathcal{C}$ is cartesian. When looking at Wadler’s type-theoretic definition of monads [Wad92, Wad95], that is at the basis of the implementation of monads in Haskell language, a natural interpretation of the calculus is into a cartesian closed category (ccc), such that two families of combinators, or a pair of polymorphic operators called the “unit” and the “bind”, exist satisfying the *monad laws*, namely (the syntactic counterpart of) the three equations in Definition 2.1.1 below (see also Proposition 2.3.3). This is more directly expressed by defining the interpretation of Wadler’s version of the λ_C-calculus into a (locally small) subcategory of **Set** which is a ccc: here $\mathcal{C}$ will be called a *concrete ccc.* Examples are the category **Dom** of Scott domains with continuous functions, and its subcategory **Alg** of algebraic lattices with a countable basis.

Definition 2.1.1. (Computational Monad [Wad95] §3)
Let $\mathcal{C}$ be a concrete ccc. A functional computational monad*, henceforth* functional monad *over $\mathcal{C}$ is a triple $(T, unit, \star)$ where T is a map over the objects of $\mathcal{C}$, and unit and $\star$ are families of morphisms*

$$unit_A : A \to TA \qquad\qquad \star_{A,B} : TA \times (TB)^A \to TB$$

such that, writing $\star_{A,B}$ as an infix operator and omitting subscripts:

$$\begin{array}{llcl} \textit{Left unit}: & unit\, a \star f & = & f\, a \\ \textit{Right unit}: & m \star unit & = & m \\ \textit{Assoc}: & (m \star f) \star g & = & m \star \lambda\!\!\lambda\, d.(f\, d \star g) \end{array}$$

where $\lambda\!\!\lambda$ is functional abstraction in the metalanguage: $\lambda\!\!\lambda\, d.(f\, d \star g)$ means $d \mapsto f\, d \star g$.

This definition is the semantic counterpart of the type theoretic one in [Wad95], but for the $\star$ which is curried in [Wad92] and Wadler's subsequent papers, so that it has type $TA \to (A \to TB) \to TB$; we adopt here the uncurried form to avoid the cumbersome double exponent.

Leaving aside the discussion about different ways to define a λ_C-model over a ccc in general, for which the interested reader might consult [Pow00] and the literature cited there, we limit ourself to show that a functional monad is indeed a strong monad and a λ_C-model in the sense of Moggi. Clearly Definition 2.1.1 is quite close to the notion of a *Kleisli triple* $(T, \eta, \_^\dagger)$, which is an equivalent definition of a monad.

Definition 2.1.2. (Kleisli triple [Mog91], Def. 1.2)
A Kleisli triple *over a category $\mathcal{C}$ is a structure $(T, \eta, \_^\dagger)$ where $T : Obj_{\mathcal{C}} \to Obj_{\mathcal{C}}$ is a map over the objects of $\mathcal{C}$, and there are a family of morphisms $\eta_A : A \to TA$ of $\mathcal{C}$ and a map $\_^\dagger$, we refer to as the* extension map *of T, sending any morphism $f : A \to TB$ in $\mathcal{C}$ to a morphism $f^\dagger : TA \to TB$ of the same category, such that:*

$$f^\dagger \circ \eta_A = f, \qquad \eta_A^\dagger = \mathrm{id}_{TA}, \qquad f^\dagger \circ g^\dagger = (f^\dagger \circ g)^\dagger \tag{2.1}$$

where $g : C \to TA$.

In case of concrete ccc, Definition 2.1.1 coincides with the notion of *Kleisli triple* just given. In fact, the equivalence is obtained by relating maps η and $\_^\dagger$ to *unit* and $\star$ operators, as follows:

$$unit_X = \eta_X \qquad\qquad a \star f = f^\dagger(a)$$

The first equality is just a notational variation of the coercion of values into computations. The latter, instead, establishes the connection between the $\star$ operator with the map $\_^\dagger$.

Let state and prove this equivalence formally.

Proposition 2.1.3. *A functional monad $(T, unit, \star)$ over a concrete ccc $\mathcal{C}$ induces a Kleisli triple over $\mathcal{C}$.*

Proof. Take $\eta_A = unit_A$ and $f^\dagger = \lambda\!\!\!\lambda\, x.x \star f : TA \to TB$ for $f : A \to TB$. By the equation (*Left unit*) the following diagram commutes:

$$\begin{tikzcd}
A \arrow[d, "unit_A"'] \arrow[dr, "f"] & \\
TA \arrow[r, "f^\dagger = \lambda\!\!\!\lambda\, x.\, x \star f"'] & TB
\end{tikzcd}$$

By definition we have that $\eta_A^\dagger = \lambda\!\!\!\lambda\, x.x \star unit_A$; therefore by (*Right unit*), for all $a \in TA$ we have:

$$\eta_A^\dagger\, a = (\lambda\!\!\!\lambda\, x.x \star unit_A)\, a = a \star unit_A = a$$

Finally by (*Assoc*) the following diagram commutes:

$$\begin{tikzcd}
C \arrow[d, "unit_C"'] \arrow[dr, "g"] & A \arrow[d, "unit_A"'] \arrow[dr, "f"] & \\
TC \arrow[r, "g^\dagger"'] \arrow[d, "\mathrm{id}_{TC}"'] & TA \arrow[r, "f^\dagger"'] & TB \\
TC \arrow[rr, "(f^\dagger \circ g)^\dagger = \lambda\!\!\!\lambda\, y.\, y \star (\lambda\!\!\!\lambda\, x.\, g\, x \star f)"] & & TB \arrow[u, "\mathrm{id}_{TB}"']
\end{tikzcd}$$

namely for all $c \in TC$:

$$(f^\dagger \circ g)^\dagger\, c = c \star (\lambda\!\!\!\lambda\, x.\, g\, x \star f) = (c \star g) \star f$$

□

The above proof relies on the inter-definability of extension and bind by the equation $f^\dagger a = a \star f$. On the other hand by looking closely to this equation, we see that what we have constructed is the morphism:

$$\_^\dagger = \lambda\!\!\!\lambda\, f\, x.\, x \star f : TB^A \to TB^{TA} \tag{2.2}$$

internalizing the Kleisli map. This is the essential step in the construction of what is called a $\mathcal{C}$-monad in [Pow00], §5.

Since then we have not completely exploited the fact that $\mathcal{C}$ is a ccc. As such it has all finite products, so that for any A, B the morphism

$$t_{A,B} : A \times TB \to T(A \times B)$$

is definable in terms of $\star$, pairing and projections as

$$t_{A,B} = \lambda x.\, (\pi_2\, x) \star \lambda y.\, unit\,(\pi_1\, x, y) \tag{2.3}$$

which is such that:

$$t\,(a, m) = m \star \lambda y. unit\,(a, y) \tag{2.4}$$

Then t is a *tensorial strength* in the sense of [Mog91], Def. 3.2, as stated in the following proposition.

Proposition 2.1.4. *Given a monad* $(T, unit, \star)$ *the* t *defined in Equation* (2.3) *commutes with the natural isomorphisms* $r_A : 1 \times A \to A$ *and* $\alpha_{A,B,C} : (A \times B) \times C \to A \times (B \times C)$*:*

i) $t_{1,A} \circ r_{TA} = Tr_A$

ii) $T\alpha_{A,B,C} \circ t_{A\times B,C} = t_{A,B\times C} \circ (\mathrm{id}_A \times t_{B,C}) \circ \alpha_{A,B,TC}$

Moreover:

iii) $t_{A,B} \circ (\mathrm{id}_A \times unit_B) = unit_{A\times B}$

iv) $t_{A,B} \circ (\mathrm{id}_A \times \mu_B) = \mu_{A\times B} \circ Tt_{A,B} \circ t_{A,TB}$

where $\mathrm{id}_A = \lambda x{:}A.\, x$, $\mu_A = (\mathrm{id}_{TA})^\dagger = \lambda z.\, z \star \mathrm{id}_{TB}$ *and for any* $f : A \to B$*,*

$$Tf = (unit_B \circ f)^\dagger = \lambda z.\, z \star (unit_B \circ f)$$

Therefore $(T, unit, \star, t)$ *is a* strong monad*.*

Proof. By definition unfolding and straightforward calculations. ☐

2.2 The Computational Core $\lambda_{©}$

The computational λ-calculus was introduced by Moggi in [Mog89, Mog91] as a meta-language to describe non functional effects in programming languages via an incremental approach. The basic idea is to distinguish among values of some type D and computations over such values, the latter having type TD. Semantically T is a monad, endowing D with a richer structure such that operations over computations can be seen as algebras of T.

The monadic approach is not only useful when building compilers modularly with respect to various kinds of effects [Mog91], to interpret languages with effects like control operators via a CPS translation [Fil94], or to write effectful programs in a purely functional language such as Haskell [Wad95], but also to **reason about such programs**. In this respect, typed computational lambda-calculus has been related to static program analysis and type and effect systems [WT03, BHM02], PER based relational semantics [BKHB06], and more recently co-inductive methods for reasoning about effectful programs have been investigated [DGL17].

We aim to investigate the monadic approach to effectful functional languages in the untyped case. This is motivated by the fact that similar, if not more elusive questions arise for effectful untyped languages as well as for typed ones; but also because the untyped setting is the natural one where studying program analysis via type assignment systems in Curry style, like in the case of intersection types (as we do in Part II), which we advocate. Indeed, working out the approach in the untyped case lays the foundation for doing the same also for typed languages, either by seeing intersection types as refinement types, or by looking at them as to the formulas of the endogenous logic of domain theoretic interpretations of types [Abr91]. The tools we use to establish program equivalence are the classic theory of reduction and intersection type assignment, which we exploit for defining the logical semantics of programs.

As already said in the introduction, it might appear nonsense to speak of monads w.r.t. an untyped calculus, as the monad T interprets a type constructor both in Moggi's and in Wadler's formulation of the computational λ-calculus [Mog91, Wad95]. However, much as the untyped λ-calculus can be seen as a calculus with a single type, which is interpreted by a retract of its own function space in a suitable category as formerly observed by Scott [Sco80], the untyped computational λ-calculus $\lambda_{©}$ that we are going to introduce formally, has two types: the type of *values* D and the type of *computations* TD. The type D is a retract of $D \to TD$, written $D \lhd D \to TD$, that is an appropriate space of functions from values to computations [Mog89]. Since we are interested in an extensional model, instead of retractions we consider type isomorphisms. In fact, in [Mog89], §5, the semantics of the untyped computational calculus is given by two kinds of reflexive objects in the category of λ_C-models, which are the solution of either the equation $D = TD \to TD$ in case of call-by-name, or

$$D = D \to TD \tag{2.5}$$

in case of call-by-value.

Since the distinction among values and computations is central in λ_C, which has been conceived as a generalization of Plotkin's call-by-value λ-calculus, we adopt Equation (2.5) in defining the corresponding untyped calculus, which leads to the syntax of our calculus in Definition 2.2.1.

Now we formally introduce the syntax and the reduction of the *computational core*, shortly $\lambda_{©}$, already presented in [dT20]. The notation in use for *returned values* is line with [Wad95], another notation could be !. Playing with syntactical changes would be convenient both to present the calculus in a more familiar fashion, and to establish useful connections between $\lambda_{©}$ and two well known calculi, namely Simpson's calculus [Sim05] and Plotkin's call-by-value λ-calculus [Plo75]. This topic will be detailed in Chapter 3.

Definition 2.2.1 (Terms of $\lambda_{©}$)**.** *The terms consist of two sorts of expressions:*

$$\begin{array}{rrcll} \mathit{Val}: & V, W & ::= & x \mid \lambda x.M & \textit{(values)} \\ \mathit{Com}: & M, N & ::= & [V] \mid M \star V & \textit{(computations)} \end{array}$$

where x ranges over a denumerable set Var of variables. We set Term $:= \mathit{Val} \cup \mathit{Com}$*;* $\mathsf{fv}(V)$ *and* $\mathsf{fv}(M)$ *are the sets of free variables occurring in V and M, respectively, and are defined in the obvious way. Terms are identified up to clash avoiding renaming of bound variables (α-congruence).*

Because of Equation (2.5) it is easy to see that, if we assume that all variables x have type D, then any value term V has type D and any computation term has type TD. Indeed omitting contexts, we have:

$$
\begin{array}{cc}
x : D \vdash x : D & \dfrac{x : D \vdash M : TD}{\lambda x.M : D \to TD = D} \\[2ex]
\dfrac{V : D}{[V] : TD} & \dfrac{M : TD \qquad V : D = D \to TD}{M \star V : TD}
\end{array}
\tag{2.6}
$$

Remark 2.2.2 (Application)**.** In the grammar presented in Definition 2.2.1 there is no functional application at all.

In fact, while VW might be included, yielding a term of type TD, none among MV, VM and MN have a type in the calculus; indeed these are not well formed terms according to Definition 2.2.1.

This may seem a strong limitation because we apparently cannot express iterated application, since $(VW)N$ is not well formed. Nonetheless, the application among computations is definable:

$$MN \equiv M \star (\lambda z.N \star z) \qquad \text{for} \qquad z \notin \mathsf{fv}(N) \tag{2.7}$$

Remark 2.2.3 (Let constructor)**.** With respect to Moggi's λ_C-syntax, we do not have the *let* construct, which is considered as syntactical sugar for bind and abstraction:

$$\mathsf{let}\, x := N \,\mathsf{in}\, M \;\equiv\; N \star \lambda x.M$$

Last but not least, there is a deeper reason for not having functional application as primitive, a reason connected to the actual implementation. The reason is that the bind $\star$ itself represents an effectful form of application, such that by redefining the unit and bind one obtains an actual evaluator for the desired computational effects [Wad95]. Indeed one should keep in mind that both $[\cdot]$ and $\star$ are abstractions of concrete realizations of such operators, depending on the effects, and hence on the monad we want to model. Adding application as primitive, and consequently the β_v rule, extends the calculus by a kind of a pure application that is in general different than the impure one, that is effectful and represented by the bind: this is an advantage when designing a programming language, but not with a foundational calculus.

2.3 Reflexive T-object and $\lambda_{©}$-model

We end this introduction to $\lambda_{©}$ grammar by defining the notion of $\lambda_{©}$-model in a concrete ccc, by analogy to that of environment λ-model in [Mey82].

Definition 2.3.1 (Reflexive T-object and $\lambda_{©}$-model)**.** *Let $\mathcal{C}$ be a concrete ccc, and $(T, \mathit{unit}, \star)$ a functional monad over $\mathcal{C}$. Then an object $D \in \mathit{Obj}_{\mathcal{C}}$ is* T-reflexive *if there exist the $\mathcal{C}$-morphisms $\Phi : D \to TD^D$ and $\Psi : TD^D \to D$ such that $\Phi \circ \Psi = \mathrm{id}_{TD^D}$.*

A $\lambda_{©}$-model *in $\mathcal{C}$ is a tuple $\mathcal{M}(D) = (D, T, \Phi, \Psi)$ where T is a monad, D is a reflexive T-object via Φ and Ψ. Then, setting $\mathit{Term\text{-}Env}_D = \mathit{Var} \to D$ as the set of* variable environments *ranged over by ρ, we define a pair of maps $[\![\cdot]\!]^D : \mathit{Val} \times \mathit{Term\text{-}Env}_D \to D$ and $[\![\cdot]\!]^{TD} : \mathit{Com} \times \mathit{Term\text{-}Env}_D \to D$, such that:*

i) $[\![x]\!]^D_\rho = \rho(x)$

ii) $[\![\lambda x.M]\!]^D_\rho = \Psi(\lambda\!\!\!\lambda\, d \in D.[\![M]\!]^{TD}_{\rho[x\mapsto d]})$

iii) $[\![[V]]\!]^{TD}_\rho = \mathit{unit}\,[\![V]\!]^D_\rho$

iv) $[\![M \star V]\!]^{TD}_\rho = [\![M]\!]^{TD}_\rho \star \Phi([\![V]\!]^D_\rho)$

where $\rho[x \mapsto d](x) = d$ and $\rho[x \mapsto d](y) = \rho(y)$ if $y \not\equiv x$. Finally we say that $\mathcal{M}$ is extensional *if also $\Psi \circ \Phi = \mathrm{id}_D$.*

As in case of environment λ-models, these interpretations $[\![\cdot]\!]^D$ and $[\![\cdot]\!]^{TD}$ are well defined depends on the fact that $\lambda\!\!\!\lambda\, d \in D.[\![M]\!]^{TD}_{\rho[x\mapsto d]} \in TD^D$, which is easily established by induction over M. In the following we shall write D for a $\lambda_{©}$-model $\mathcal{M}(D)$ whenever the context is unambiguous, and call it just a model.

By $M\{V/x\}$ and $W\{V/x\}$ we denote the capture avoiding substitution of x by V in M and W, respectively. This means that x is not bound in M, W and that V is free for x in M, namely $\mathsf{fv}(V) \cap BV(M) = \emptyset$, and similarly for W. Since we have a denumerable set of variables and identify α-congruent terms, such conditions can always be satisfied.

Lemma 2.3.2. *Let D be a model. Then for all $V, W \in \mathit{Val}$ and $M \in \mathit{Com}$, and for all $\rho \in \mathit{Term\text{-}Env}_D$:*

$$[\![W\{V/x\}]\!]^D_\rho = [\![W]\!]^D_{\rho[x\mapsto[\![V]\!]^D_\rho]} \quad \textit{and} \quad [\![M\{V/x\}]\!]^{TD}_\rho = [\![M]\!]^{TD}_{\rho[x\mapsto[\![V]\!]^D_\rho]}$$

Proof. By an easy induction over W and M. □

For any model D and $M, N \in \mathit{Com}$, we write $D \models M = N$ if for all $\rho \in \mathit{Term\text{-}Env}_D$ it holds $[\![M]\!]^{TD}_\rho = [\![N]\!]^{TD}_\rho$. Then we write $\models M = N$ if $D \models M = N$ for any model D.

Proposition 2.3.3 (Monad laws)**.** *For all $V \in \mathit{Val}$ and $M, N, L \in \mathit{Com}$ it holds that:*

i) $\models [V] \star (\lambda x.M) = M\{V/x\}$

ii) $\models M \star \lambda x.[x] = M$

iii) $\models (L \star \lambda x.M) \star \lambda y.N = L \star \lambda x.(M \star \lambda y.N)$ *where $x \notin FV(N)$.*

Proof. By definition unfolding and easy calculations. E.g. for arbitrary D and $\rho \in \mathit{Term\text{-}Env}_D$:

$$\begin{array}{rcl}
[\![[V] \star (\lambda x.M)]\!]^{TD}_\rho & = & \mathit{unit}\,[\![V]\!]^D_\rho \star \Phi(\Psi(\lambda\!\!\!\lambda\, d \in D.[\![M]\!]^{TD}_{\rho[x\mapsto d]})) \\
& = & \mathit{unit}\,[\![V]\!]^D_\rho \star \lambda\!\!\!\lambda\, d \in D.[\![M]\!]^{TD}_{\rho[x\mapsto d]} \\
& = & (\lambda\!\!\!\lambda\, d \in D.[\![M]\!]^{TD}_{\rho[x\mapsto d]})[\![V]\!]^D_\rho \\
& = & [\![M]\!]^{TD}_{\rho[x\mapsto[\![V]\!]^D_\rho]} = [\![M\{V/x\}]\!]^{TD}_\rho
\end{array}$$

using Definition 2.3.1, equations $\Phi \circ \Psi = \mathrm{id}_{TD^D}$, *Left unit* and Lemma 2.3.2. □

2.4 The Reduction Relation

Henceforth, but just in this part, we will use a slightly different notational variant of the computational core, representing the bind constructor just as an application:

$$VM := M \star V$$

Note that, recalling Remark 2.2.2, this notational variation could be misleading, since it seems that a functional application is taking place: this is not the case as the bind operator is not just a functional application. This choice is to make the rewriting treatment more readable and connection with other calculi more visible.

In conclusion, this allows to present the calculus in a more familiar fashion and it also enlightens some issues left aside in the literature. But in [dT20], as well as in Part II and Part III, the bind operator is explicitly presented, as in the referred [Wad95]. In the authors' opinion, the bind notation is convenient when it comes to a semantical analysis, thanks to the its connection with monad laws. In addition it useful to stress that the bind operator has a operational flavour, too: it is a chronological point of view about the evaluation, as clearly understandable in the connection with let constructor in Remark 2.2.3.

Definition 2.4.1 (Reduction)**.** *The relation* $\lambda_{©} = \beta_c \cup \mathsf{id} \cup \sigma$ *is the union of the following binary relations over Com:*

$$\begin{array}{llll}
\beta_c) & (\lambda x.M)([V]) & \mapsto_{\beta_c} & M\{V/x\} \\
\mathsf{id}\,) & (\lambda x.[x])M & \mapsto_{\mathsf{id}} & M \\
\sigma\,) & (\lambda y.N)((\lambda x.M)L) & \mapsto_{\sigma} & (\lambda x.(\lambda y.N)M)L \quad \textit{for } x \notin \mathsf{fv}(N)
\end{array}$$

The substitution is defined as aspected. The reduction $\to_{©}$ *is the contextual closure of* $\lambda_{©}$*, where* contexts *are defined as follows:*

$$\mathsf{C} ::= \langle\,\rangle \mid [\lambda x.\mathsf{C}] \mid V\mathsf{C} \mid (\lambda x.\mathsf{C})M \qquad\qquad \textit{Contexts}$$

Remark 2.4.2. Let us continue with the Remark 2.2.2 and with the fact that the application VW can be introduced in our calculus. In fact, as remarked in [dT20], the only form of application which is admissible is VW, that is a computation with type TD. By adding these new terms to *Com*, we have to consider the axiom of β_v-conversion, namely $(\lambda x.M)W = M\{W/x\}$. However, this introduces redundancies in the calculus; let VW be defined as abbreviation of $V[W]$, then by Equation (1.1) and taking $V = \lambda x.M$:

$$(\lambda x.M)W \equiv (\lambda x.M)[W] \mapsto_{\beta_c} M\{W/x\} \tag{2.8}$$

Remark 2.4.3 (β_c and β_v)**.** We are now in place to show how a β_v-reduction is simulated by a β_c-reduction, simulated possibly in more steps. Observe that the reduction relation $\to_{©}$ is only on computations, therefore no computation N will ever reduce to some value V; however, this is represented by a reduction

$N \to_{©}^* [V]$, where $[V]$ is the coercion of the value V into a computation. Moreover, let us assume that $M \to_{©}^* [\lambda x.M']$. We have:

$$\begin{array}{rcl} MN & \equiv & (\lambda z.zN)M \\ & \to_{©}^* & (\lambda z.z[V])[\lambda x.M'] \\ & \to_{\beta_c} & (\lambda x.M')[V] \\ & \to_{\beta_c} & M'\{V/x\} \end{array}$$

where, if $z \notin \mathsf{fv}(N)$ then $z \notin \mathsf{fv}(V)$.

Surface and weak reduction. As we shall see in the next sections, there are two natural restriction of $\to_{©}$: *weak reduction* $\underset{\mathsf{w}}{\to}_{©}$ which does not fire in the scope of λ, and *surface reduction* $\underset{\mathsf{s}}{\to}_{©}$, which does not fire in the scope of $[\cdot]$. The former is the evaluation usually studied in CbV λ-calculus (Theorem 1.2.11). The latter is the natural evaluation in linear logic, and in Simpson's calculus, whose relation with $\lambda_{©}$ we discuss in Section 3.2.1.

Surface and *weak contexts* are, respectively, defined by the grammars

$$\begin{array}{lr} \mathsf{S} ::= \langle\,\rangle \mid V\mathsf{S} \mid (\lambda x.\mathsf{S})M & \text{Surface Contexts} \\ \mathsf{W} ::= \langle\,\rangle \mid V\mathsf{W} & \text{Weak Contexts} \end{array}$$

For $\rho \in \{\beta_c, \mathsf{id}, \sigma, \lambda_{©}\}$, *weak* reduction $\underset{\mathsf{w}}{\to}_\rho$ is the closure of ρ under weak contexts W, *surface* reduction $\underset{\mathsf{s}}{\to}_\rho$ is its closure under surface contexts S. *Non-surface* reduction $\underset{\neg\mathsf{s}}{\to}_\rho$ is the closure of ρ under contexts C that are not surface. Similarly, *non-weak* reduction $\underset{\neg\mathsf{w}}{\to}_\rho$ is the closure of ρ under contexts C that are not weak.

Clearly, $\underset{\mathsf{w}}{\to}_\rho \subseteq \underset{\mathsf{s}}{\to}_\rho$. Note that $\underset{\mathsf{w}}{\to}_{\beta_c}$ is a *deterministic* relation, while $\underset{\mathsf{s}}{\to}_{\beta_c}$ is not.

Example 2.4.4. To clarify the difference between surface and weak, let us consider the term $(\lambda x.\underline{\boldsymbol{I}[x]})[\lambda y.\boldsymbol{I}[y]]$, where $\boldsymbol{I} = \lambda z.[z]$. We underline the fired redex.

- $(\lambda x.\underline{\boldsymbol{I}[x]})[\lambda y.\boldsymbol{I}[y]] \underset{\mathsf{s}}{\to}_{©} (\lambda x.[x])[\lambda y.\boldsymbol{I}[y]]$;
- $(\lambda x.\underline{\boldsymbol{I}[x]})[\lambda y.\boldsymbol{I}[y]] \underset{\mathsf{w}}{\not\to}_{©} (\lambda x.[x])[\lambda y.\boldsymbol{I}[y]]$;
- $(\lambda x.\boldsymbol{I}[x])[\lambda y.\underline{\boldsymbol{I}[y]}] \underset{\mathsf{s}}{\not\to}_{©} (\lambda x.\boldsymbol{I}[x])[\lambda y.[y]]$.

Remark 2.4.5 (Uniqueness of weak contexts)**.** As we recalled in Section 1.2.2, in CbV weak contexts can be given in three forms, according to the order in which redexes which are not in the scope of abstractions are fired: $\mathsf{L}, \mathsf{R}, \mathsf{W}$. When the grammar of terms is restricted to computations, the three coincide. So in the computational core there is *only one* definition of weak context.

In Section 3.2 and Section 4.2, we analyze the *properties* of weak and surface reduction. We first study $\to_{\beta_c}$, and then we move to the whole $\lambda_{©}$, where σ and id also come to play.

Computations are closed under reduction. The set of computations *Com* is closed under substitutions and reduction:

Proposition 2.4.6.

1. *If $M \in Com$ and $V \in Val$ then $M\{V/x\} \in Com$.*

2. *If $M \in Com$ and $M \to_{\rho} M'$ then $M' \in Com$, for $\rho \in \{\beta_c, \sigma, \mathsf{id}\}$. The same property holds for* surface *reduction $\underset{\mathsf{s}}{\to}_{\rho}$ and* weak *reduction $\underset{\mathsf{w}}{\to}_{\rho}$.*

Proof. Item 1 follows by observing that in $M\{V/x\}$ we just replace the value x by another value V. Item 2 follows by item 1 and by induction over contexts. □

CHAPTER 3

RELATING $\lambda_{©}$ WITH OTHER CALCULI

We relate the computational core to other well-known calculi: Sabry and Wadler's refined computational calculus, Simpson's linear calculus, Plotkin's CbV, Moggi's original computational calculus. These connections are done in order to place the computational core in the existing literature, and transferring already known properties from these calculi to the computational core .

3.1 The Computational Core and let-notation

It is natural to compare the core calculus $\lambda_{©}$ with other untyped computational calculi with the *let*-constructor, as already mentioned in Remark 2.2.3. A useful reference is [SW97], where Moggi's untyped calculus λ_C [Mog88] is refined into a variant, called λ_{ml^*}, that we display in Figure 3.1.

$$\begin{array}{rrcl} \textit{Values:} & V, W & ::= & x \mid \lambda x.M \\ \textit{Computations:} & M, N & ::= & [V] \mid \mathsf{let}\, x := M \,\mathsf{in}\, N \mid VW \end{array}$$

Reduction $\to_{ml^*}$ is the contextual closure of the following rules.

$$\begin{array}{rrcll} (c.\beta) & (\lambda x.M)V & \to & M\{V/x\} & \\ (c.\eta) & \lambda x.Vx & \to & V & x \notin \mathsf{fv}(V) \\ (c.\mathsf{let}.\beta) & \mathsf{let}\, x := [V] \,\mathsf{in}\, N & \to & N\{V/x\} & \\ (c.\mathsf{let}.\eta) & \mathsf{let}\, x := M \,\mathsf{in}\, [x] & \to & M & x \notin \mathsf{fv}(M) \\ (c.\mathsf{let}.ass) & \mathsf{let}\, y := (\mathsf{let}\, x := L \,\mathsf{in}\, M) \,\mathsf{in}\, N & \to & \mathsf{let}\, x := L \,\mathsf{in}\, (\mathsf{let}\, y := M \,\mathsf{in}\, N) & \end{array}$$

Figure 3.1: λ_{ml^*}: Syntax and Reduction

To state a correspondence between $\lambda_{©}$ and λ_{ml^*}, consider the translations in Figure 3.2. The translations induce an equational correspondence (in the sense of [SF93]) by adding η-equality to $\lambda_{©}$.

$$
\begin{array}{rcl}
\multicolumn{3}{c}{(\cdot)^{\bullet} : \lambda_{ml^*} \to \lambda_{\copyright}} \\[1em]
(x)^{\bullet} & := & x \\
(\lambda x.M)^{\bullet} & := & \lambda x.(M)^{\bullet} \\
(VW)^{\bullet} & := & V^{\bullet}\,[W^{\bullet}] \\
([V])^{\bullet} & := & [V^{\bullet}] \\
(\mathsf{let}\, x := M \,\mathsf{in}\, N)^{\bullet} & := & (\lambda x.N^{\bullet})M^{\bullet}
\end{array}
\qquad
\begin{array}{rcl}
\multicolumn{3}{c}{(\cdot)^{\circ} : \lambda_{\copyright} \to \lambda_{ml^*}} \\[1em]
(x)^{\circ} & := & x \\
(\lambda x.M)^{\circ} & := & \lambda x.(M)^{\circ} \\
([V])^{\circ} & := & [V^{\circ}] \\
(V\,[W])^{\circ} & := & V^{\circ}\,W^{\circ} \\
(xM)^{\circ} & := & \mathsf{let}\, y := M^{\circ} \,\mathsf{in}\, xy \\
((\lambda x.N)M)^{\circ} & := & \mathsf{let}\, x := M^{\circ} \,\mathsf{in}\, N^{\circ}
\end{array}
$$

where in the last two clauses for $(\cdot)^{\circ}$, $y \notin \mathsf{fv}(M)$ and $M \neq [W]$ for any value W.

Figure 3.2: Translations

More precisely, let $\to_{\eta}$ be the contextual closure of the rule

$$\lambda x.(V[x]) \mapsto_{\eta} V$$

Let $=_{\copyright\eta}$ be the reflexive-transitive and symmetric closure of the reduction $\to_{\copyright\eta} = \to_{\copyright} \cup \to_{\eta}$, and similarly for $=_{ml^*}$ with respect to $\to_{ml^*}$.

Proposition 3.1.1. *The following hold:*

1. $M =_{\copyright\eta} (M^{\circ})^{\bullet}$ *for every term* M *in* $\lambda_{\copyright}$;
2. $(P^{\bullet})^{\circ} =_{ml^*} P$ *for every term* P *in* λ_{ml^*};
3. $M =_{\copyright\eta} N$ *implies* $M^{\circ} =_{ml^*} N^{\circ}$;
4. $P =_{ml^*} Q$ *implies* $P^{\bullet} =_{\copyright\eta} Q^{\bullet}$.

Proof. By induction over the definition of the translations, see Appendix C.1 for a detailed proof, and Proposition C.1.1 for another, more informative one. □

In Proposition 3.1.1 we consider $=_{\copyright\eta}$, which includes η-conversion, since

$$xM \neq_{\copyright} ((xM)^{\circ})^{\bullet} = (\lambda y.x[y])(M^{\circ})^{\bullet}$$

(where $=_{\copyright}$ is the reflexive-transitive and symmetric closure of $\to_{\copyright}$) and so condition (1) in Proposition 3.1.1 would not hold if we replace $=_{\copyright\eta}$ with $=_{\copyright}$.

It is natural to wonder if there is a Galois connection as defined in [SW97]. This amounts to replace $=_{\copyright\eta}$ with $\to^{*}_{\copyright\eta}$ and $=_{ml^*}$ with $\to^{*}_{ml^*}$ in Proposition 3.1.1. The answer is negative because the condition corresponding to (1), namely $M \to^{*}_{\copyright\eta} (M^{\circ})^{\bullet}$, fails since $xM \not\to^{*}_{\copyright\eta} ((xM)^{\circ})^{\bullet}$. The lack of a Galois connection means that the two calculi do not have the same reduction theory, even if they share the same convertibility theory, and have the same denotational semantics, by Proposition 3.1.1

3.2 The Operational Properties of β_c

Since β-reduction is the "engine" of any λ-calculus, we continue such comparisons by focussing on the properties of $\to_{\beta_c}$ and its surface and weak restriction. We do so by relating it first with the reduction $\to_{\beta_!}$ of Simpson's linear λ-calculus [Sim05], and then with $\to_{\beta_v}$ of Plotkin's CbV calculus.

3.2.1 Computational versus Bang Calculus

We call *bang calculus* the fragment of Simpson's linear λ-calculus [Sim05] without linear abstraction. It has also been studied in [EG16, GM19] (with the name bang calculus, which we adopt), and it is closely related to Levy's Call-by-Push-Value [Lev99].

When considering only β_c reduction, it is easy to see that $(\mathit{Com}, \rightarrow_{\beta_c})$ is the restriction of the bang calculus to computations. Therefore, β_c has the same syntactical properties, which we review below.

The bang calculus. We briefly recall the bang calculus $(\Lambda^!, \rightarrow_{\beta_!})$.

Terms $\Lambda^!$ are defined by

$$T, S, Q, P ::= x \mid TS \mid \lambda x.T \mid !T \qquad\qquad (\textbf{Terms } \Lambda^!)$$

Contexts (C) and *surface contexts* (S) are generated by the grammars:

$$\begin{aligned} \mathsf{C} &::= \langle\rangle \mid T\mathsf{C} \mid \mathsf{C}T \mid \lambda x.\mathsf{C} \mid !\mathsf{C} && (\textbf{Contexts}) \\ \mathsf{S} &::= \langle\rangle \mid T\mathsf{S} \mid \mathsf{S}T \mid \lambda x.\mathsf{S} && (\textbf{Surface Contexts}) \end{aligned}$$

The reduction $\rightarrow_{\beta_!}$ is the closure under context C of the rule

$$(\lambda x.P)!Q \mapsto_{\beta_!} P\{Q/x\}$$

Surface reduction $\underset{\mathsf{s}}{\rightarrow}_{\beta_!}$ is the closure of the rule $\mapsto_{\beta_!}$ under surface contexts S. *Non-surface* reduction $\underset{\mathsf{s}}{\rightarrow}_{\beta_!}$ is the closure of the rule $\mapsto_{\beta_!}$ under contexts C that are not surface. Surface reduction *factorizes* $\rightarrow_{\beta_!}$.

Theorem 3.2.1 (Surface Factorization [Sim05], Prop. 5.2)**.** *In* $(\Lambda^!, \rightarrow_{\beta_!})$*:*

1. Surface Factorization of $\rightarrow_{\beta_!}$: $\quad \rightarrow_{\beta_!}^* \subseteq \underset{\mathsf{s}}{\rightarrow}_{\beta_!}^* \cdot \underset{\neg\mathsf{s}}{\rightarrow}_{\beta_!}^*$.
2. Boxes: $\quad T \rightarrow_{\beta_!}^* !Q$ *if and only if* $T \underset{\mathsf{s}}{\rightarrow}_{\beta_!}^* !P$

Surface reduction is non-deterministic, but satisfies the diamond property of Fact 1.2.6.

Theorem 3.2.2 (Confluence and diamonds [Sim05])**.** *In* $\Lambda^!$*:*

- *the relation* $\rightarrow_{\beta_!}$ *is confluent;*
- *the relation* $\underset{\mathsf{s}}{\rightarrow}_{\beta_!}$ *has the quasi-diamond property.*

Restriction to Computations. The restriction of the calculus above to computations, *i.e.* $(\mathit{Com}^!, \rightarrow_{\beta_!})$ is *exactly the same* as the fragment of $\lambda_©$ with β_c-rule as unique reduction rule, *i.e.* $(\mathit{Com}, \rightarrow_{\beta_c})$.

First, observe that the set of computations Com defined in Chapter 2 is a subset of the terms $\Lambda^!$, and moreover it is closed under $\rightarrow_{\beta_c}$ reduction (exactly as in Proposition 2.4.6). Second, observe that the restriction of context and surface contexts to computations, give exactly the grammar defined in Chapter 2. Then $(\mathit{Com}^!, \rightarrow_{\beta_!})$ and $(\mathit{Com}, \rightarrow_{\beta_c})$ are in fact *the same* (modulo replacing the ! operator with $[\cdot]$), and:

- $M \to_{\beta_!} N$ if and only if $M \to_{\beta_c} N$.
- $M \xrightarrow[\mathsf{s}]{}_{\beta_!} N$ if and only if $M \xrightarrow[\mathsf{s}]{}_{\beta_c} N$.

Hence, $\to_{\beta_c}$ in the computational core *inherits all the syntactical properties of the reduction* $\to_{\beta_!}$, and in particular, *surface factorization* and the *quasi-diamond property* of $\xrightarrow[\mathsf{s}]{}_{\beta_!}$. We will use both extensively.

Fact 3.2.3 (Properties of β_c and its *surface* restriction). *In* $\lambda_{©}$*:*
$\to_{\beta_c}$ *is non deterministic and confluent,*
$\xrightarrow[\mathsf{s}]{}_{\beta_c}$ *is quasi-diamond, and so is confluent,*
$\to_{\beta_c}$ *satisfies surface factorization.*

3.2.2 Computational versus Call-by-Value

$(Com, \to_{\beta_c})$ is also isomorphic to the *kernel* of Plotkin's CbV λ-calculus, that is the restriction of $\to_{\beta_v}$ to the the terms $Com^v \subseteq \Lambda$ which are defined as follows.

$$\begin{array}{rrcl} Val^v: & V, W & ::= & x \mid \lambda x.M \\ Com^v: & M, N, L & ::= & V \mid VM \end{array}$$

The isomorphism between $(Com, \to_{\beta_c})$ and $(Com^v, \to_{\beta_v})$ is given in Appendix C.2. Observe also that the restriction of *weak context* to Com^v gives exactly the grammar defined in Chapter 2, and this for all three weak contexts $(\mathsf{L}, \mathsf{R}, \mathsf{W})$, which all collapse to the same shape. Summing up:

Fact 3.2.4 (Properties of β_c and its *weak* restriction). *In* $\lambda_{©}$*:*
$\xrightarrow[\mathsf{w}]{}_{\beta_c}$ *is deterministic,*
$\to_{\beta_c}$ *satisfies weak factorization:* $\quad \to_{\beta_c}^* \subseteq \xrightarrow[\mathsf{w}]{}_{\beta_c}^* \cdot \xrightarrow[\neg\mathsf{w}]{}_{\beta_c}^*$

Call-by-value versus its kernel. Interestingly, the *kernel* of the CbV (and hence $\to_{\beta_c}$) is as expressive as CbV λ-calculus, as we discuss below. This result was already shown by Accattoli [Acc12].[1]

With respect to its kernel, Plotkin's CbV λ-calculus is more liberal in that application is unrestricted (left-hand side need not be a value). The kernel has the same expressive power as CbV calculus, because the full syntax can be encoded into that of the restricted one, and because the kernel can simulate every reduction sequence of the full calculus.

Formally, consider the translation $(\cdot)^\bullet$ from the CbV λ-calculus to the kernel.

$$(x)^\bullet = x \qquad (\lambda x.P)^\bullet = \lambda x.P^\bullet \qquad (PQ)^\bullet = \begin{cases} P^\bullet Q^\bullet & \text{if } P \text{ is a value;} \\ (\lambda x.xQ^\bullet)P^\bullet & \text{otherwise.} \end{cases}$$

Proposition 3.2.5 (Simulation of the CbVλ-calculus into its kernel). *For every term P in the CbV λ-calculus, if $P \to_{\beta_v} Q$ then $P^\bullet \to_{\beta_v}^+ Q^\bullet$.*

[1]Precisely, Accattoli studies the relation between the kernel calculus λ_{vker} and the *value substitution calculus* λ_{vsub}, *i.e.* CbV and the kernel extended with explicit substitutions. The syntax is slightly different, but not in an essential way.

3.3 The Computational Core and the Original Moggi's Calculus

Moggi's type free calculus in [Mog89] §6, called λ_C (a name we have been using here for Moggi's typed calculus) is clearly the ancestor of $\lambda_{©}$, but the usage of unit and bind in place of the *let*-constructor is not a little change.

On the one hand we have defined reduction by orienting the three monad laws: this is simpler than having six rules plus η as in Moggi's case. By the way, relating the two calculi is not straightforward. Indeed the untyped λ_C includes application of arbitrary terms and admits terms like $\mathsf{let}\, x := M \,\mathsf{in}\, N$ for non values N; also sorts are not preserved by reduction, and a non-value may reduce to a value, as in CbV λ-calculus: however, if this is not disturbing in case of the latter, it produces a semantic mismatch when monads are involved, since D and TD are different domains, in general.

In [dT19], and here in Appendix C.3, we report the grammar and reduction theory of λ_C and formally define a translation $(\cdot)^\circ : \lambda_{©} \to \lambda_C$, where $(VM)^\circ$ is equal to $(\mathsf{let}\, x := (M)^\circ \,\mathsf{in}\, (V)^\circ x)$. Actually, the translation will be formalized between Moggi's calculus and the notational variant of the computational core where one has $M \star V$ that will be in use in Parts II and III. By the way, this translation preserves conversion, but not reduction, for similar arguments that we have encountered when dealing with λ_{ml^*}.

In the opposite direction, we have a translation $(\cdot)^\bullet : \lambda_C \to \lambda_{©}$ that is as $(n)^\bullet = (n)^{\bullet Com}$ and $(v)^\bullet = [(v)^{\bullet Val}]$ where in Moggi's terms n is a non value, v is a value, and the translations $(n)^{\bullet Com}$ and $(v)^{\bullet Val}$ into *Com* and *Val*, respectively, are mutually defined. E.g. in case of the *let*-expressions we have four clauses:

$$\begin{array}{rcl}
(\mathsf{let}\, x := n \,\mathsf{in}\, n')^{\bullet Com} & = & \lambda x.(n')^{\bullet Com}(n)^{\bullet Com} \\
(\mathsf{let}\, x := v \,\mathsf{in}\, n')^{\bullet Com} & = & \lambda x.(n')^{\bullet Com}[(v)^{\bullet Val}] \\
(\mathsf{let}\, x := n \,\mathsf{in}\, v)^{\bullet Com} & = & \lambda x.[(v)^{\bullet Val}](n)^{\bullet Com} \\
(\mathsf{let}\, x := v \,\mathsf{in}\, v')^{\bullet Com} & = & \lambda x.[(v')^{\bullet Val}][(v)^{\bullet Val}]
\end{array}$$

This translation preserves reduction when η is dropped from reduction in the untyped λ_C; otherwise, we must add η_c rule to the reduction relation, hence losing confluence (see Example 4.1.4).

By this, both the confluence proof of λ_C in [MOTW99] §8.3, where it is called COMP, and the proof by checking critical pair using the tool PolySOL in [Ham18], cannot be used in our case, and we have preferred to prove confluence of $\lambda_{©}$ from scratch, although following a standard pattern: see e.g. [AFM$^+$95]. Confluence of COMP is established in [MOTW99] via a translation from a call-by-need linear λ-calculus, but without η, facing a similar difficulty as we mention at the end of Section 4.1, where we treat a way to prove confluence of $\lambda_{©}$.

CHAPTER 4

OPERATIONAL PROPERTIES: CONFLUENCE, FACTORIZATIONS, AND RETURNING VALUES

4.1 Route to Confluence of $\lambda_{\copyright}$

A fundamental property of reduction in ordinary λ-calculus is confluence, established in the Church-Rosser theorem.

Although the most essential notions of rewriting properties have already been presented in Section 1.2.1, in the following, before the most crucial results, the definitions of the properties we are going to prove will be covered again.

In this section we focus on proving confluence of $\to$ for the $\lambda_{\copyright}$-calculus, i.e. without any differentiation on in which kind of context the reduction is taking place. This is a harder task since reduction in $\lambda_{\copyright}$ has three axioms instead of just the β rule of the λ-calculus, whose left-hand sides generate a number of critical pairs. Before embarking into the proof, let us see a few examples, some of them could be a rephrase of already presented examples for different purposes.

Example 4.1.1. In this example we see how outer reduction by σ may overlap with an inner reduction by β_c. Representing the given reductions by solid arrows, we see how to recover confluence by a reduction and a relation represented by a dashed arrow and a dashed line, respectively:

$$\begin{tikzcd}
(\lambda y.\, N)((\lambda x.\, M)[V]) \arrow[r, "\sigma"] \arrow[d, "\beta_c"'] & (\lambda x.\, (\lambda y.\, N)M)[V] \arrow[d, dashed, "\beta_c"] \\
(\lambda y.\, N)(M\{V/x\}) \arrow[r, dash, dashed, "\equiv"'] & ((\lambda y.\, N)M)\{V/x\}
\end{tikzcd}$$

where $x \notin \mathsf{fv}(N)$, which is the side condition to rule σ; therefore the two terms in the lower line of the diagram are syntactically identical.

Example 4.1.2. In this example we see how outer reduction by σ, overlapping with outer id, can be recovered by an inner reduction by id:

$$\begin{tikzcd}
(\lambda x.\,[x])((\lambda y.\,N)M) \arrow[rr, "\sigma"] \arrow[dr, "\mathsf{id}"'] & & (\lambda y.(\lambda x.\,[x])\,N)M \arrow[dl, dashed, "\mathsf{id}"] \\
& (\lambda y.\,N)M &
\end{tikzcd}$$

Example 4.1.3. Here the outer reduction by σ overlaps with an inner reduction by id. This is recovered by means of an inner reduction by β_c:

$$\begin{tikzcd}
(\lambda y.\,N)((\lambda x.\,[x])M) \arrow[r, "\sigma"] \arrow[d, "\mathsf{id}"'] & (\lambda x.(\lambda y.\,N)\,[x])M \arrow[d, dashed, "\beta_c"] \\
(\lambda y.\,N)M \arrow[r, dashed, no head, "\alpha"'] & (\lambda x.\,N\{x/y\})M
\end{tikzcd}$$

where $x \notin \mathsf{fv}(N)$ as observed in Example 4.1.1, and therefore $\lambda x.\,N\{x/y\}$ is the renaming by x of the bound variable y in $\lambda y.\,N$: then the dashed line represents α-congruence.

Example 4.1.4. We end by considering the issue of weak and full extensionality, that have not been treated in Section 2.4. Weak extensionality, also called ξ-rule of the ordinary λ-calculus, is reduction under abstraction. This is guaranteed by the contextual closure, but only in the context of computation terms.

Concerning extensionality an analogous of η-rule is:

$$\eta_c) \quad \lambda x.\,(V\,[x]) \to_{\eta_c} V, \qquad x \notin FV(V) \tag{4.1}$$

This involves extending reduction from *Com* to the whole *Term*. However the reduction obtained by adding η_c to $\to_{©}$ is not confluent:

$$\begin{tikzcd}
(\lambda z.N)((\lambda x.\,(y[x]))M) \arrow[r, "\sigma"] \arrow[d, "\eta_c"'] & (\lambda x.(\lambda z.N)\,(y[x]))M \arrow[d, dashed] \\
(\lambda z.N)(yM) \arrow[r, dashed] & ?
\end{tikzcd}$$

After having inspected the above examples, one might by tempted to conclude that the reduction in Definition 2.4.1 enjoys the diamond property, namely it is confluent within (at most) two single steps, one per side (for the diamond property see Equation (4.3) below: we say here ‘at most’ because $\to$ is not reflexive). Unfortunately, this is not the case because of rule β_c, that can multiplicate redexes in the reduced term exactly as the β-rule in ordinary λ-calculus. Even worse, rule

σ generates critical pairs with all other rules and with itself, preventing the simple extension of confluence proofs for β-reduction to succeed.

Following a strategy used in case of call-by-need calculi with the let-construct (see e.g. [AFM+95, MOTW99]), we split the proof in three steps, proving confluence of $\beta_c \cup \mathsf{id}$ and σ, separately, and then combining these results by means of the commutativity of these relations.

Confluence of $\beta_c \cup \mathsf{id}$. In this first step we adapt the parallel reduction method, originally due to Tait and Martin Löf [Bar85], and further developed by Takahashi [Tak95]. See e.g. the book [Ter03] ch. 10. Let's define the following relation $\mathrel{-\!\!\ominus\!\!\rightarrow}$:

Definition 4.1.5. *The relation $\mathrel{-\!\!\ominus\!\!\rightarrow} \subseteq \mathit{Term} \times \mathit{Term}$ is inductively defined by:*

1. $x \mathrel{-\!\!\ominus\!\!\rightarrow} x$
2. $M \mathrel{-\!\!\ominus\!\!\rightarrow} N \Rightarrow \lambda x.M \mathrel{-\!\!\ominus\!\!\rightarrow} \lambda x.N$
3. $V \mathrel{-\!\!\ominus\!\!\rightarrow} V' \Rightarrow [V] \mathrel{-\!\!\ominus\!\!\rightarrow} [V]'$
4. $M \mathrel{-\!\!\ominus\!\!\rightarrow} M'$ *and* $V \mathrel{-\!\!\ominus\!\!\rightarrow} V' \Rightarrow VM \mathrel{-\!\!\ominus\!\!\rightarrow} V'M'$
5. $M \mathrel{-\!\!\ominus\!\!\rightarrow} M'$ *and* $V \mathrel{-\!\!\ominus\!\!\rightarrow} V' \Rightarrow (\lambda x.M)[V] \mathrel{-\!\!\ominus\!\!\rightarrow} M'\{V'/x\}$
6. $M \mathrel{-\!\!\ominus\!\!\rightarrow} M' \Rightarrow (\lambda x.[x])M \mathrel{-\!\!\ominus\!\!\rightarrow} M'$

By Item 1 - Item 4 above, relation $\mathrel{-\!\!\ominus\!\!\rightarrow}$ is reflexive and coincides with its compatible closure. Also $\rightarrow_{\beta_c\mathsf{id}} \subseteq \mathrel{-\!\!\ominus\!\!\rightarrow}$; intentionally, this is not the case w.r.t. the whole $\rightarrow$.

Lemma 4.1.6. *For $M, M' \in \mathit{Com}$ and $V, V' \in \mathit{Val}$ and every variable x, if $M \mathrel{-\!\!\ominus\!\!\rightarrow} M'$ and $V \mathrel{-\!\!\ominus\!\!\rightarrow} V'$, then $M\{V/x\} \mathrel{-\!\!\ominus\!\!\rightarrow} M'\{V'/x\}$.*

Proof. By an easy induction on the definition of $M \mathrel{-\!\!\ominus\!\!\rightarrow} M'$ and $V \mathrel{-\!\!\ominus\!\!\rightarrow} V'$. □

Now, by means of Lemma 4.1.6 one easily proves that $\mathrel{-\!\!\ominus\!\!\rightarrow} \subseteq \rightarrow^*_{\beta_c\mathsf{id}}$.

The next step in the proof is to show that the relation $\mathrel{-\!\!\ominus\!\!\rightarrow}$ satisfies the *triangle property TP* defined in Section 1.2.1 :

$$\forall P\, \exists P^*\, \forall Q.\ P \mathrel{-\!\!\ominus\!\!\rightarrow} Q \ \Rightarrow\ Q \mathrel{-\!\!\ominus\!\!\rightarrow} P^* \tag{4.2}$$

where $P, P^*, Q \in \mathit{Term}$. *TP* implies the *diamond property DP*, which for $\mathrel{-\!\!\ominus\!\!\rightarrow}$ is:

$$\forall P, Q, R.\ P \mathrel{-\!\!\ominus\!\!\rightarrow} Q \ \&\ P \mathrel{-\!\!\ominus\!\!\rightarrow} R \ \Rightarrow\ \exists P'.\ Q \mathrel{-\!\!\ominus\!\!\rightarrow} P' \ \&\ R \mathrel{-\!\!\ominus\!\!\rightarrow} P' \tag{4.3}$$

In fact, if *TP* holds then we can take $P' \equiv P^*$ in *DP*, since the latter only depends on P. We then define P^* in terms of P as follows:

1. $x^* \equiv x$
2. $(\lambda x.M)^* \equiv \lambda x.M^*$
3. $[V]^* \equiv [V^*]$

4. $((\lambda x.M)[V])^* \equiv M^*\{V^*/x\}$
5. $((\lambda x.[x])M)^* \equiv M^*$, if $M \not\equiv [V]$ for $V \in \mathit{Val}$
6. $(VM)^* \equiv V^*M^*$, $M \not\equiv [W]$ for $W \in \mathit{Val}$ and $V \not\equiv \lambda x.[x]$

Lemma 4.1.7. *For all $P, Q \in \mathit{Term}$, if $P \mathrel{\ominus\!\!\!\rightarrow} Q$ then $Q \mathrel{\ominus\!\!\!\rightarrow} P^*$, namely $\mathrel{\ominus\!\!\!\rightarrow}$ satisfies TP.*

Proof. By induction on $P \mathrel{\ominus\!\!\!\rightarrow} Q$. The base case $x \mathrel{\ominus\!\!\!\rightarrow} x$ follows by $x^* \equiv x$. All remaining cases follow by the induction hypotheses; in particular, if $P \equiv (\lambda x.M)[V] \mathrel{\ominus\!\!\!\rightarrow} M'\{V'/x\} \equiv Q$ because $M \mathrel{\ominus\!\!\!\rightarrow} M'$ and $V \mathrel{\ominus\!\!\!\rightarrow} V'$, then by induction $M' \mathrel{\ominus\!\!\!\rightarrow} M^*$ and $V' \mathrel{\ominus\!\!\!\rightarrow} V^*$, so that $M'\{V'/x\} \mathrel{\ominus\!\!\!\rightarrow} M^*\{V^*/x\} \equiv P^*$ by Lemma 4.1.6. □

According to [Bar85], Def. 3.1.11, a notion of reduction R is said to be *confluent* or *Church-Rosser*, shortly *CR*, if $\to^*_R$ satisfies *DP*; more explicitly in our setting for all $M, N, L \in \mathit{Com}$:

$$M \to^*_{\copyright} N \;\;\&\;\; M \to^*_{\copyright} L \Rightarrow \exists M' \in \mathit{Com}.\, N \to^*_{\copyright} M' \;\;\&\;\; L \to^*_{\copyright} M'$$

Corollary 4.1.8. *The notion of reduction $\beta_c \cup \mathsf{id}$ is CR.*

Proof. As observed above $\to_{\beta_c\mathsf{id}} \subseteq \mathrel{\ominus\!\!\!\rightarrow}$, hence $M \to^*_{\beta_c\mathsf{id}} N$ implies $M \mathrel{\ominus\!\!\!\rightarrow}^+ N$, where $\mathrel{\ominus\!\!\!\rightarrow}^+$ is the transitive closure of $\mathrel{\ominus\!\!\!\rightarrow}$, and similarly $M \mathrel{\ominus\!\!\!\rightarrow}^+ L$. By Lemma 4.1.7 $\mathrel{\ominus\!\!\!\rightarrow}$ satisfies *TP*, hence it satisfies *DP*. By an easy argument (see e.g. [Bar85] Lemma 3.2.2) we conclude that $N \mathrel{\ominus\!\!\!\rightarrow}^+ M'$ and $L \mathrel{\ominus\!\!\!\rightarrow}^+ M'$ for some M', from which the thesis follows by the fact that $\mathrel{\ominus\!\!\!\rightarrow} \subseteq \to^*_{\beta_c\mathsf{id}}$. □

Confluence of σ. To prove confluence of σ (more properly of its contextual closure) we use Newman Lemma (see [Bar85], Prop. 3.1.24). A notion of reduction R is *weakly Church-Rosser*, shortly *WCR*, if for all $M, N, L \in \mathit{Com}$:

$$M \to_R N \;\;\&\;\; M \to_R L \Rightarrow \exists M' \in \mathit{Com}.\, N \to^*_R M' \;\;\&\;\; L \to^*_R M'$$

Lemma 4.1.9. *The notion of reduction σ is WCR.*

Proof. It suffices to show the thesis for the critical pair $M_1 \to_\sigma M_2$ and $M_1 \to_\sigma M_3$ where:

$$\begin{array}{lcl} M_1 & \equiv & (\lambda z.P)((\lambda y.N)((\lambda x.M)L)) \\ M_2 & \equiv & (\lambda y.((\lambda z.P)N))((\lambda x.M)L) \\ M_3 & \equiv & (\lambda z.P)((\lambda x.((\lambda y.N)M))L) \end{array}$$

Then in one step we have:

$$M_2 \to_\sigma ((\lambda x.(\lambda y.((\lambda z.P)N))M)L) \equiv M_4$$

but

$$M_3 \to_\sigma ((\lambda x.(\lambda z.P)((\lambda y.N)M)))L \to_\sigma M_4$$

where the two reduction steps are necessary. □

Recall that a notion of reduction R is *strongly normalizing*, shortly *SN*, if there exists no infinite reduction $M \to_R M_1 \to_R M_2 \to_R \cdots$ out of any $M \in \mathit{Com}$.

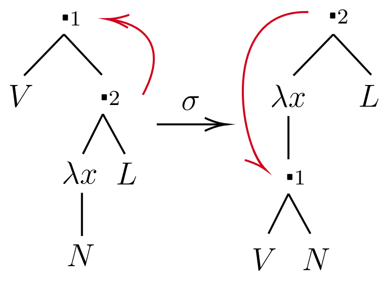
Lemma 4.1.10. *The notion of reduction σ is SN.*

Proof. Given $M \in \mathit{Com}$, let's denote by the same M the expression obtained by marking differently all applications in M, say $\bullet_1, \ldots, \bullet_n$ (every term is finite, so it is the number of application constructors). For example $M \equiv (\lambda x.[x])((\lambda y.[y])[z])$ becomes with this slight change of notation $(\lambda x.[x]) \bullet_1 ((\lambda y.[y]) \bullet_2 [z])$.

We say that $\bullet_j$ is to the right to $\bullet_i$ in M if there exists a subterm $V \bullet_i L$ of M such that $\bullet_j$ occurs in L. Finally, let's denote by $\sharp M$ the number of pairs $(\bullet_i, \bullet_j)$ such that $\bullet_j$ is to the right of $\bullet_i$ in M.

If a term includes an σ-redex $(\lambda y.P) \bullet_j ((\lambda x.N) \bullet_i L)$, which is contracted to $(\lambda x.(\lambda y.P) \bullet_j N) \bullet_i L$, then $\bullet_j$ is to the right of $\bullet_i$ in the redex, but not in the contractum. Also it is easily seen by induction on terms that, if $\bullet_j$ is not to the right of $\bullet_i$ in M and $M \to_\sigma N$, the same holds in N.

It follows that, if $M \to_\sigma N$ then $\sharp M > \sharp N$, hence σ is *SN*. The described counter-clockwise movement is depicted in the image below:



□

Corollary 4.1.11. *The notion of reduction σ is CR.*

Proof. By Lemma 4.1.9, Lemma 4.1.10 and by Newman Lemma, stating that a notion of reduction which is *WCR* and *SN* is *CR*. □

The last step to confluence Finally we show that $\to_{\beta_c\mathsf{id}}$ and $\to_\sigma$ commute.

Lemma 4.1.12. *Reductions $\to_{\beta_c\mathsf{id}}$ and $\to_\sigma$ commute.*

Proof. By Lemma 1.2.10, two strongly commuting relations commute, and commutativity is clearly symmetric; hence it suffices to show that

$$N \,{}_{\beta_c\mathsf{id}}\!\longleftarrow M \to_\sigma L \Rightarrow \exists P \in \mathit{Com}.\ N \to_\sigma^= P \,{}_{\beta_c\mathsf{id}}\!\overset{*}{\longleftarrow} L$$

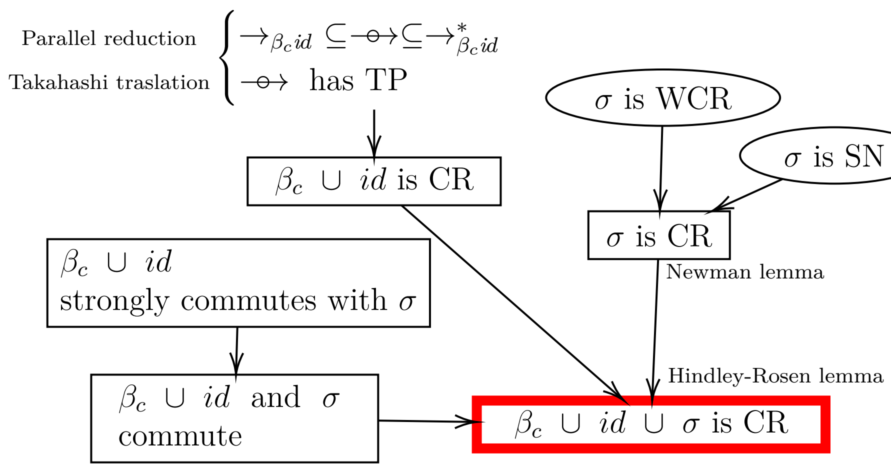
We can limit the cases to the critical pairs, that are exactly those in Examples 4.1.1 to 4.1.3, which commute. □

Theorem 4.1.13 (Confluence). *The notion of reduction $\lambda_{©} = \beta_c \cup \mathsf{id} \cup \sigma$ is CR.*

Proof. By the commutative union lemma (see [BN98], Lem. 2.7.10 and [Bar85], Prop. 3.3.5, where it is called Hindley-Rosen Lemma), if $\to_{\beta_c\mathsf{id}}$ and $\to_\sigma$ and are both *CR* (Corollary 4.1.8 and Corollary 4.1.11), and commute (Lemma 4.1.12), then $\to_{©} = \to_{\beta_c\mathsf{id}} \cup \to_\sigma$ is *CR*. □

Consequence of Theorem 4.1.13 is the unicity of the normal form of a term, if any. Also, by construction and Proposition 2.3.3, convertible terms, related by the reflexive, symmetric and transitive closure $=_{©}$ of $\to_{©}$, namely its *convertibility relation*, are equated in any model of $\lambda_{©}$.

The image below aims to recap the process in proving the confluence of $\lambda_{©}$:



Parallel reduction
Takahashi traslation
$\to_{\beta_c id} \subseteq \multimap \subseteq \to^*_{\beta_c id}$
$\multimap$ has TP
σ is WCR
σ is SN
$\beta_c \cup id$ is CR
σ is CR
Newman lemma
$\beta_c \cup id$ strongly commutes with σ
Hindley-Rosen lemma
$\beta_c \cup id$ and σ commute
$\beta_c \cup id \cup \sigma$ is CR

4.1.1 Recap: Confluence Properties of β_c, σ, and id

Finally, we briefly revisit the confluence of $\lambda_{©}$, in order to analyze the confluence properties of the different subsystems. In fact, in Chapter 5 we will use confluence of $\to_{\sigma\beta_c}$ (Theorem 5.3.4). Sketching the following proof we highlight other ways by which the confluence could have been proved.

Proposition 4.1.14 (Confluence of β_c, σ and id).

1. *Each of the relations* $\to_{\beta_c}$, $\to_{\mathsf{id}}$, $\to_\sigma$ *is confluent.*
2. $(\to_{\beta_c} \cup \to_{\mathsf{id}})$ *and* $(\to_{\beta_c} \cup \to_\sigma)$ *are confluent.*
3. $(\to_{\beta_c} \cup \to_\sigma \cup \to_{\mathsf{id}})$ *is confluent.*
4. $(\to_\sigma \cup \to_{\mathsf{id}})$ *is* not *confluent.*

Proof.

1. The confluence of $\to_{\beta_c}$ is stated in Theorem 3.2.2; $\to_\sigma$ is locally confluent and terminating, and so confluent (Corollary 4.1.11); $\to_{\mathsf{id}}$ satisfies the diamond property of Fact 1.2.6, and so is confluent.
2. It is easily verified that $\to_{\beta_c}$ and $\to_{\mathsf{id}}$ strongly commute or as in Corollary 4.1.8, and so also $\to_{\beta_c}$ and $\to_\sigma$. The claim then follows by Hindley-Rosen Lemma.
3. $\to_{\beta_c} \cup \to_{\mathsf{id}}$ strongly commutes with $\to_\sigma$ (Lemma 4.1.12). This point is delicate because to close a diagram of the shape $\leftarrow_\sigma \cdot \to_{\mathsf{id}}$ may require a $\to_{\beta_c}$ step, (see Example 4.2.7). The claim then follows by Hindley-Rosen Lemma.
4. A counterexample is provided by the same diagram which is mentioned in the previous point Example 4.2.7, requiring a $\to_{\beta_c}$-step to close. □

4.2 Weak and Surface Reduction of $\lambda_{\copyright}$

The aim of studying *factorization* is to investigate evaluation and normalization. The factorization theorems that will be presented in Section 4.3 are based on both weak and surface reductions. The construction which we develop in the next sections demands more work than one may expect. This is due to the fact that the rules induced by the monadic laws of associativity and identity make the analysis of the reduction properties non trivial, as seen for confluence. In particular—as anticipated in the introduction— *weak reduction* does not factorize $\lambda_{\copyright}$, and has severe drawbacks, which we explain next. *Surface reduction* behaves better, but also has some issues. For this reason, we will use a combination of both, surface and weak reductions. In the rest of this chapter, we examine their respective properties.

4.2.1 Weak Reduction: the impact of Associativity and Identity

Weak (left) reduction (Section 1.2.2) is one of the most common and studied way to implement evaluation in CbV, and more generally in calculi with effects.

The closure $\underset{\mathsf{w}}{\rightarrow}_{\beta_c}$ of β_c under weak context is a *deterministic* relation, as expected. However, when including the rules induced by the monadic equation of associativity and identity, the reduction is *non-deterministic*, *non-confluent*, and normal forms are *not unique*.

This is somehow surprising, given the prominent role of such a reduction in the literature of calculi with effects. Notice that the issues only come from σ and id, not from β_c. Summing up:

1. $\underset{\mathsf{w}}{\rightarrow}_{\mathsf{id}}$ and $\underset{\mathsf{w}}{\rightarrow}_{\beta_c\mathsf{id}}$ are non-deterministic, but are both confluent.

2. $\underset{\mathsf{w}}{\rightarrow}_{\sigma}$, $\underset{\mathsf{w}}{\rightarrow}_{\sigma\beta_c}$, $\underset{\mathsf{w}}{\rightarrow}_{\sigma\mathsf{id}}$ and $\underset{\mathsf{w}}{\rightarrow}_{\copyright}$ are non-deterministic, non-confluent and normal forms are not unique. That is, adding σ, weak reduction loses confluence and uniqueness of the normal form.

Example 4.2.1 (Non-confluence)**.** An example of the non-determinism of $\underset{\mathsf{w}}{\rightarrow}_{\mathsf{id}}$ is the following:

$$V((\lambda y.!y)N) \underset{\mathsf{w}}{\leftarrow}_{\mathsf{id}} (\lambda x.!x)(V((\lambda y.!y)N)) \underset{\mathsf{w}}{\rightarrow}_{\mathsf{id}} (\lambda x.!x)(VN)$$

For an example of non-confluence of weak reduction (Point 2 above), consider $T = V((\lambda x.P)((\lambda y.Q)L))$ where $V = \lambda z.z[z]$ and $P = Q = L = z[z]$. Then,

$$M_1 = (\lambda x.VP)((\lambda y.Q)L) \underset{\mathsf{w}}{\leftarrow}_{\sigma} \ T \underset{\mathsf{w}}{\rightarrow}_{\sigma} V((\lambda y.(\lambda x.P)Q)L) = N_1$$

$$M_1 \underset{\mathsf{w}}{\rightarrow}_{\sigma} M_2 \ = \ (\lambda y.(\lambda x.VP)Q)L \neq (\lambda y.V((\lambda x.P)Q))L \ = \ N_2 \underset{\mathsf{w}}{\leftarrow}_{\sigma} N_1$$

where $N_1 \underset{\mathsf{w}}{\not\rightarrow}_{\copyright} M_2$, and $M_1 \underset{\mathsf{w}}{\not\rightarrow}_{\copyright} N_2$, and both M_2 and N_2 are $\underset{\mathsf{w}}{\rightarrow}_{\copyright}$-normal.

Reduction $\rightarrow_{\beta_c}$ (like $\rightarrow_{\beta_v}$) admits weak factorization

$$\rightarrow^*_{\beta_c} \subseteq \underset{\mathsf{w}}{\rightarrow}^*_{\beta_c} \cdot \underset{\neg\mathsf{w}}{\rightarrow}^*_{\beta_c}$$

This is not the case for $\rightarrow_{\copyright}$. The following counter-example is due to van Oostrom [vO20a] .

Example 4.2.2 (Non-factorization [vO20a])**.** $\rightarrow_{\copyright}$ does not admit weak factorization. Consider the reduction sequence

$$M := (\lambda y.\boldsymbol{I}[y])(z[z]) \underset{\neg\mathsf{w}}{\rightarrow}_{\beta_c} (\lambda y.[y])(z[z]) \underset{\mathsf{w}}{\rightarrow}_{\mathsf{id}} z[z]$$

No weak step is possible from M.

Let: Different Notation, Same issues. We stress that the issues are inherent to the associativity and identity rules, not to the specific syntax of $\lambda_{\copyright}$. Exactly the same issues appear in Sabry-Wadler's λ_{ml^*} (Section 3.1).

Example 4.2.3 (Evaluation context in let-notation)**.** In let-notation, weak reduction corresponds to *sequencing.* The evaluation context is

$$\mathsf{E}_{\mathsf{let}} = \langle\rangle \mid \texttt{let}\ x = \mathsf{E}_{\mathsf{let}}\ \texttt{in}\ M$$

We write $\underset{\mathsf{e}}{\rightarrow}_{ml^*}$ for the closure of the λ_{ml^*} rules (in Section 3.1) under $\mathsf{E}_{\mathsf{let}}$. We observe two problems, the first one due to the rule $c.\mathsf{let}.ass$, the second one to the rule $c.\mathsf{let}.\eta$.

1. Non-confluence. Because of the associative rule ($c.\mathsf{let}.ass$), the reduction $\underset{\mathsf{e}}{\rightarrow}_{ml^*}$ is *non-deterministic, non-confluent, and normal forms are not unique.* Consider the following term, where $R = P = Q = L = zz$

$$T := \overline{\texttt{let}\ z = \underline{(\texttt{let}\ x = (\texttt{let}\ y = L\ \texttt{in}\ Q)\ \texttt{in}\ P)}\ \texttt{in}\ R}$$

There are two weak redexes, the overlined and the underlined one. So

$$\begin{aligned}
T\ &\underset{\mathsf{e}}{\rightarrow}_{ml^*}\ \texttt{let}\ x = (\texttt{let}\ y = L\ \texttt{in}\ Q)\ \texttt{in}\ (\texttt{let}\ z = P\ \texttt{in}\ R)\\
&\underset{\mathsf{e}}{\rightarrow}_{ml^*}\ \texttt{let}\ y = L\ \texttt{in}\ (\texttt{let}\ x = Q\ \texttt{in}\ (\texttt{let}\ z = P\ \texttt{in}\ R)) := T'\\
T\ &\underset{\mathsf{e}}{\rightarrow}_{ml^*}\ \texttt{let}\ z = (\texttt{let}\ y = L\ \texttt{in}\ (\texttt{let}\ x = Q\ \texttt{in}\ P))\ \texttt{in}\ R\\
&\underset{\mathsf{e}}{\rightarrow}_{ml^*}\ \texttt{let}\ y = L\ \texttt{in}\ (\texttt{let}\ z = (\texttt{let}\ x = Q\ \texttt{in}\ P)\ \texttt{in}\ R) := T''
\end{aligned}$$

Neither T' nor T'' can be further reduced. In fact, they are two different normal forms with respect to the reduction $\underset{\mathsf{e}}{\rightarrow}_{ml^*}$.

2. Non-factorization. Because of the $c.\mathsf{let}.\eta$ rule, *factorization w.r.t. sequencing* does not hold. That is, a reduction sequence $M \rightarrow^*_{ml^*} N$ cannot be reorganized as weak steps followed by non-weak steps. Consider the following variation on van Oostrom's Example 4.2.2:

$$\begin{aligned}
M\ &:=\ \texttt{let}\ y = (zz)\ \texttt{in}\ (\texttt{let}\ x = [y]\ \texttt{in}\ [x])\\
&\underset{\neg\mathsf{w}}{\rightarrow}_{c.\mathsf{let}.\eta}\ \texttt{let}\ y = (zz)\ \texttt{in}\ [y]\ \underset{\mathsf{w}}{\rightarrow}_{c.\mathsf{let}.\eta}\ (zz) =: N
\end{aligned}$$

No sequencing step is possible from M, so it is *impossible* to factorize the reduction form M to N as $M \underset{\mathsf{w}}{\rightarrow}^*_{ml^*} \cdot \underset{\neg\mathsf{w}}{\rightarrow}^*_{ml^*} N$

4.2.2 Surface Reduction

In $\lambda_{\copyright}$, surface reduction is non-deterministic, but confluent, and well-behaving.

Fact 4.2.4 (Non-determinism)**.** *For $\rho \in \{\lambda_{\copyright}, \beta_c, \sigma, \mathsf{id}, \sigma\beta_c, \beta\mathsf{id}, \sigma\mathsf{id}\}$, $\underset{s}{\rightarrow}_{\rho}$ is non-deterministic (because in general more than one surface redex can be fired).*

We now study confluence of surface reduction. We will use confluence of $\underset{s}{\rightarrow}_{\sigma\beta_c}$ (point 2. below) in Chapter 5 (Theorem 5.3.3).

Proposition 4.2.5 (Confluence of surface reductions.)**.**

1. *Each of the relations $\underset{s}{\rightarrow}_{\beta_c}$, $\underset{s}{\rightarrow}_{\mathsf{id}}$, $\underset{s}{\rightarrow}_{\sigma}$ is confluent.*
2. *$(\underset{s}{\rightarrow}_{\beta_c} \cup \underset{s}{\rightarrow}_{\mathsf{id}})$ and $(\underset{s}{\rightarrow}_{\beta_c} \cup \underset{s}{\rightarrow}_{\sigma})$ are confluent.*
3. *$(\underset{s}{\rightarrow}_{\beta_c} \cup \underset{s}{\rightarrow}_{\sigma} \cup \underset{s}{\rightarrow}_{\mathsf{id}})$ is confluent.*
4. *$(\underset{s}{\rightarrow}_{\sigma} \cup \underset{s}{\rightarrow}_{\mathsf{id}})$ is* not *confluent.*

Proof. We rely on confluence of $\underset{s}{\rightarrow}_{\beta_c}$ (by Theorem 3.2.2), and on Hindley-Rosen Lemma (Lemma 1.2.9).

We prove commutation via strong commutation (Lemma 1.2.10). The only delicate point is the commutation of $\underset{s}{\rightarrow}_{\sigma}$ with $\underset{s}{\rightarrow}_{\mathsf{id}}$ (points 3. and 4.).

1. $\underset{s}{\rightarrow}_{\sigma}$ is locally confluent and terminating, and so confluent. $\underset{s}{\rightarrow}_{\mathsf{id}}$ satisfies the diamond property of Fact 1.2.6, and so also confluence.
2. It is easily verified that $\underset{s}{\rightarrow}_{\beta_c}$ and $\underset{s}{\rightarrow}_{\mathsf{id}}$ strongly commute, and similarly for $\underset{s}{\rightarrow}_{\beta_c}$ and $\underset{s}{\rightarrow}_{\sigma}$. The claim then follows by Hindley-Rosen Lemma.
3. $\underset{s}{\rightarrow}_{\beta_c} \cup \underset{s}{\rightarrow}_{\mathsf{id}}$ strongly commutes with $\underset{s}{\rightarrow}_{\sigma}$. This point is delicate because to close a diagram of the shape $\underset{s}{\leftarrow}_{\sigma} \cdot \underset{s}{\rightarrow}_{\mathsf{id}}$ may require a $\underset{s}{\rightarrow}_{\beta_c}$ step, see Example 4.2.7 . The claim then follows by Hindley-Rosen Lemma.
4. A counterexample is provided by the same diagram which is mentioned in the previous point Example 4.2.7, requiring a $\underset{s}{\rightarrow}_{\beta_c}$-step to close. □

Example 4.2.6 (Surface reduction)**.** Let us give an example for non-determinism and for confluence.

1. Consider the term $(\lambda x.R)((\lambda y.R')N)$ where R and R' are any two redexes.
2. Consider the same term as in Example 4.2.1: $T = V((\lambda x.P)((\lambda y.Q)L))$. Then,
$$M_2 \underset{s}{\leftarrow}_{\sigma} M_1 \underset{s}{\leftarrow}_{\sigma} T \underset{s}{\rightarrow}_{\sigma} N_1 \underset{s}{\rightarrow}_{\sigma} N_2$$
Now we can close the diagram:
$$M_2 = (\lambda y.(\lambda x.VP)Q)L \underset{s}{\leftarrow}_{\sigma} (\lambda y.V((\lambda x.P)Q))L = N_2$$

Example 4.2.7. In the following example, where $M = N = z!z$, the σ-redex overlaps with the id-redex. The corresponding steps are surface and the only way to close the diagram is by means of a β_c step, which is also surface.

$$\begin{tikzcd}
(\lambda y.\, N)((\lambda x.\, [x])M) \arrow[r, "\sigma"] \arrow[d, "\mathsf{id}"'] & (\lambda x.(\lambda y.\, N)\, [x])M \arrow[d, dashed, "\beta_c"] \\
(\lambda y.\, N)M \arrow[r, dashed, dash, "="] & (\lambda x.\, N)M\{x/y\}
\end{tikzcd}$$

Note that $x \notin \mathsf{fv}(N)$, and therefore $\lambda x.\, N\{x/y\}$ is the renaming by x of the bound variable y in $\lambda y.\, N$.

This is also a counterexample to confluence of $(\to_\sigma \cup \to_{\mathsf{id}})$ and of $(\underset{\mathsf{s}}{\to}_\sigma \cup \underset{\mathsf{s}}{\to}_{\mathsf{id}})$.

In Section 4.3, we prove that surface reduction *does factorize* $\to_{\copyright}$, similarly to what happens for Simpson's calculus (see Theorem 3.2.1). Surface reduction also has a drawback: it does not allows us to separate $\to_{\beta_c}$ and $\to_\sigma$ steps. This fact makes it difficult to reason about returning a value, treated in Section 4.4.

Example 4.2.8 (An issue with surface reduction)**.** Define $\Delta = \lambda z.z[z]$. Consider the term

$$\Delta((\lambda x.[\Delta])(xx))$$

which is normal for $\to_{\beta_c}$ and in particular for $\underset{\mathsf{s}}{\to}_{\beta_c}$, but

$$\Delta((\lambda x.[\Delta])(x[x]))\underset{\mathsf{s}}{\to}_\sigma(\lambda x.\Delta[\Delta])(x[x])\underset{\mathsf{s}}{\to}_{\beta_c}(\lambda x.\Delta[\Delta])(x[x])$$

Here it is not be possible to postpone a step $\underset{\mathsf{s}}{\to}_\sigma$ after a step $\underset{\mathsf{s}}{\to}_{\beta_c}$.

4.3 Surface and Weak Factorization

In this section, we prove several factorization results for $\lambda_{\copyright}$. Surface factorization is the cornerstone of the following developments.

Theorem 4.3.1 (Surface Factorization of $\lambda_{\copyright}$)**.** $\lambda_{\copyright}$ *admits surface factorization:*

$$M \to_{\copyright}^* N \textit{ implies } M \underset{\mathsf{s}}{\to}_{\copyright}^* \cdot \underset{\neg\mathsf{s}}{\to}_{\copyright}^* N$$

We then refine this result first by *postponing* id steps which are not also β_c steps, and then by means of *weak factorization* (on the surface steps). This further phase serves two purposes: **(1.)** to separate $\to_{\beta_c}$ and $\to_\sigma$ steps, by postponing σ steps after the $\underset{\mathsf{w}}{\to}_{\beta_c}$ steps and **(2.)** to perform a fine analysis of quantitative properties, namely the number of β_c steps. We will need (1.) to define the *evaluation* relation, and (2.) to define the *normalizing strategies.*

Technical lemmas. In this rewriting investigation, we often exploit some basic properties of the contextual closure, that we collect here. If a step $T \to_\rho T'$ is obtained by closure under *non-empty context* of a rule $\mapsto_\rho$, then T and T' have *the same shape*, that is, both terms are an application (resp. an abstraction, a variable, or a returned value, i.e. a computation of shape $[V]$ for some value V).

Fact 4.3.2 (Shape preservation)**.** *Let $\mapsto_\rho$ be a rule and $\to_\rho$ be its contextual closure. Assume $T = \mathsf{C}\langle R\rangle \to_\rho \mathsf{C}\langle R'\rangle = T'$ and that the context C is* non-empty*. Then T and T' have the same shape.*

An easy-to-verify consequence is the following.

Lemma 4.3.3 (Redexes preservation)**.** *Assume $T \underset{\neg\mathsf{s}}{\to}_{\copyright} S$ and $\gamma \in \{\beta_c, \sigma, \mathsf{id}\}$. T is a γ-redex if and only if S is a γ-redex.*

Proof. For $\gamma \in \{\sigma, \beta_c\}$, see Corollary A.1.2. For $\gamma = \mathsf{id}$, see Lemma B.1.1. □

Notice the following inclusions, which we will use freely.

Fact 4.3.4. $\underset{\mathsf{w}}{\to}_{\copyright} \subset \underset{\mathsf{s}}{\to}_{\copyright}$ *and* $\underset{\neg\mathsf{s}}{\to}_{\copyright} \subset \underset{\neg\mathsf{w}}{\to}_{\copyright}$*, because a weak context is necessarily a surface context (but a surface context need not be a weak context).*

4.3.1 Surface Factorization of $\lambda_{\copyright}$ (modularly)

We prove surface factorization of $\lambda_{\copyright}$. We already know that surface factorization holds for $\to_{\beta_c}$ (Theorem 3.2.1), so we can rely on it, and work modularly, following the approach proposed in [AFG21]. The tests for call-by-name head factorization and call-by-value weak factorization in [AFG21] easily adapt, yielding the following convenient test. It modularly establishes surface factorization of a reduction $\to_{\beta_c} \cup \to_\gamma$, where $\to_\gamma$ is a new reduction added to $\to_{\beta_c}$. Details are in the Appendix.

Proposition 4.3.5 (A modular test for surface factorization)**.** *Let $\to_{\beta_c}$ be β_c-reduction and $\to_\gamma$ be the contextual closure of a rule $\mapsto_\gamma$. The reduction $\to_{\beta_c} \cup \to_\gamma$ satisfies surface factorization if:*

1. Surface factorization of $\to_\gamma$*:* $\quad \to_\gamma^* \subseteq \underset{\mathsf{s}}{\to}_\gamma^* \cdot \underset{\neg\mathsf{s}}{\to}_\gamma^*$
2. $\mapsto_\gamma$ *is* substitutive*:* $\quad R \mapsto_\gamma R'$ *implies* $R\{Q/x\} \mapsto_\gamma R'\{Q/x\}$.
3. Root linear swap*:* $\quad \underset{\neg\mathsf{s}}{\to}_{\beta_c} \cdot \mapsto_\gamma \subseteq \mapsto_\gamma \cdot \to_{\beta_c}^*$.

We will use the following easy property (which is an instance of Lemma A.1.4 in the Appendix).

Lemma 4.3.6. *Let $\to_\xi, \to_\gamma$ be the contextual closure of rules $\mapsto_\xi, \mapsto_\gamma$.*
$\underset{\neg\mathsf{s}}{\to}_\xi \cdot \mapsto_\gamma \subseteq \underset{\mathsf{s}}{\to}_\gamma \cdot \to_\xi^=$ *implies* $\underset{\neg\mathsf{s}}{\to}_\xi \cdot \underset{\mathsf{s}}{\to}_\gamma \subseteq \underset{\mathsf{s}}{\to}_\gamma \cdot \to_\xi^=$.

Root Lemmas. Lemmas 4.3.7 and 4.3.8 below provide *everything we need to verify the conditions of the test*, and so establish surface factorization of $\lambda_{©}$.

Lemma 4.3.7 (σ-Roots)**.** *Let $\gamma \in \{\sigma, \mathsf{id}, \beta_c\}$. The following holds:*

$$M \underset{\neg\mathsf{s}}{\rightarrow}_{\gamma} L \mapsto_{\sigma} N \textit{ implies } M \mapsto_{\sigma} \cdot \rightarrow_{\gamma} N.$$

Proof. We have $L = (\lambda x.L_1)((\lambda y.L_2)L_3) \mapsto_{\sigma} (\lambda y.(\lambda x.L_1)L_2)L_3$. Since L is a σ-redex, M is also a σ-redex (Lemma 4.3.3). So $M = (\lambda x.M_1)((\lambda y.M_2)M_3) \underset{\neg\mathsf{s}}{\rightarrow}_{\gamma}$ $(\lambda x.L_1)((\lambda y.L_2)L_3) = L$, where for only one $i \in 1, 2, 3$ $M_i \underset{\neg\mathsf{s}}{\rightarrow}_{\gamma} L_i$ and otherwise $M_j = L_j$, for $i \neq j$ (by Fact A.1.1). Therefore $M = (\lambda x.M_1)((\lambda y.M_2)M_3) \mapsto_{\sigma}$ $(\lambda y.(\lambda x.M_1)M_2)M_3 \underset{\neg\mathsf{s}}{\rightarrow}_{\gamma} (\lambda y.(\lambda x.L_1)L_2)L_3$. □

Lemma 4.3.8 (id-Roots)**.** *Let $\gamma \in \{\sigma, \mathsf{id}, \beta_c\}$. The following holds:*

$$M \underset{\neg\mathsf{s}}{\rightarrow}_{\gamma} U \mapsto_{\mathsf{id}} N \textit{ implies } M \mapsto_{\mathsf{id}} \cdot \rightarrow_{\gamma} N.$$

Proof. We have $U = \boldsymbol{I}N \mapsto_{\mathsf{id}} N$. Since U is an id-redex, M also is (Lemma 4.3.3). So $M = \boldsymbol{I}P \underset{\neg\mathsf{s}}{\rightarrow}_{\gamma} \boldsymbol{I}N$ and $P \underset{\neg\mathsf{s}}{\rightarrow}_{\gamma} N$. Therefore $M = \boldsymbol{I}P \mapsto_{\mathsf{id}} P \underset{\neg\mathsf{s}}{\rightarrow}_{\gamma} N$. □

Let us make explicit the content of these two lemmas. By simply instantiating γ Lemma 4.3.7 and Lemma 4.3.8 we know the following:

Fact 4.3.9. *1. $t \underset{\neg\mathsf{s}}{\rightarrow}_{\sigma} p \mapsto_{\sigma} q$ implies $t \mapsto_{\sigma} \cdot \rightarrow_{\sigma}^{=} q$ and so (Lemma 4.3.6) $t \underset{\neg\mathsf{s}}{\rightarrow}_{\sigma} p \underset{\mathsf{s}}{\rightarrow}_{\sigma} q$ implies $t \underset{\mathsf{s}}{\rightarrow}_{\sigma} \cdot \rightarrow_{\sigma}^{=} q$, (*i.e. *Strong Postponement holds).*

2. $t \underset{\neg\mathsf{s}}{\rightarrow}_{\mathsf{id}} p \mapsto_{\sigma} q$ implies $t \mapsto_{\sigma} \cdot \rightarrow_{\mathsf{id}}^{=} q$ and so (Lemma 4.3.6) $t \underset{\neg\mathsf{s}}{\rightarrow}_{\mathsf{id}} p \underset{\mathsf{s}}{\rightarrow}_{\sigma} q$ implies $t \underset{\mathsf{s}}{\rightarrow}_{\sigma} \cdot \rightarrow_{\mathsf{id}}^{=} q$.

3. $t \underset{\neg\mathsf{s}}{\rightarrow}_{\beta_c} p \mapsto_{\sigma} q$ implies $t \mapsto_{\sigma} \cdot \rightarrow_{\beta_c}^{=} q$.

Therefore, by instantiating γ, we have:

Fact 4.3.10. *1. $t \underset{\neg\mathsf{s}}{\rightarrow}_{\mathsf{id}} p \mapsto_{\mathsf{id}} q$ implies $t \mapsto_{\mathsf{id}} \cdot \rightarrow_{\mathsf{id}}^{=} q$, and so (Lemma 4.3.6) $t \underset{\neg\mathsf{s}}{\rightarrow}_{\mathsf{id}} p \underset{\mathsf{s}}{\rightarrow}_{\mathsf{id}} q$ implies $t \underset{\mathsf{s}}{\rightarrow}_{\mathsf{id}} \cdot \rightarrow_{\mathsf{id}}^{=} q$ (*i.e. *Strong Postponement holds).*

2. $t \underset{\neg\mathsf{s}}{\rightarrow}_{\sigma} p \mapsto_{\mathsf{id}} q$ implies $t \mapsto_{\mathsf{id}} \cdot \rightarrow_{\sigma}^{=} q$, and so (Lemma 4.3.6) $t \underset{\neg\mathsf{s}}{\rightarrow}_{\sigma} p \underset{\mathsf{s}}{\rightarrow}_{\mathsf{id}} q$ implies $t \underset{\mathsf{s}}{\rightarrow}_{\mathsf{id}} \cdot \rightarrow_{\sigma}^{=} q$

3. $t \underset{\neg\mathsf{s}}{\rightarrow}_{\beta_c} p \mapsto_{\mathsf{id}} q$ implies $t \mapsto_{\mathsf{id}} \cdot \rightarrow_{\beta_c}^{=} q$.

Now we are ready to combine everything, by using the modular test.

Surface factorization of $\to_{\mathsf{id}} \cup \to_{\sigma}$. Surface factorization of $\to_{\mathsf{id}\sigma}$ follows immediately from the root lemmas.

Lemma 4.3.11 (Surface factorization of $\mathsf{id}\sigma$)**.** *Surface factorization of $\to_{\mathsf{id}} \cup \to_{\sigma}$ holds, because:*

1. *Surface factorization of $\to_{\sigma}$ holds.*

2. *Surface factorization of $\to_{\mathsf{id}}$ holds.*

3. *Linear swap: $\underset{\neg\mathsf{s}}{\to}_{\mathsf{id}} \cdot \underset{\mathsf{s}}{\to}_{\sigma} \subseteq \underset{\mathsf{s}}{\to}_{\sigma} \cdot \to_{\mathsf{id}}^{*}$*

4. *Linear swap: $\underset{\neg\mathsf{s}}{\to}_{\sigma} \cdot \underset{\mathsf{s}}{\to}_{\mathsf{id}} \subseteq \underset{\mathsf{s}}{\to}_{\mathsf{id}} \cdot \to_{\sigma}^{*}$*

Proof. (1) follows from Fact 4.3.9, point 1, by (linear) Strong Postponement. (2) follows from Fact 4.3.10, point 1, by (linear) Strong Postponement. (3) is Fact 4.3.9, point 2. (4) is Fact 4.3.10, point 2. □

Surface factorization of $\lambda_{\copyright}$, modularly. We now use the test for modular factorization (Proposition 4.3.5)

Theorem 4.3.1 (Surface Factorization of $\lambda_{\copyright}$)**.** *$\lambda_{\copyright}$ admits surface factorization:*

$$M \to_{\copyright}^{*} N \text{ implies } M \underset{\mathsf{s}}{\to}_{\copyright}^{*} \cdot \underset{\neg\mathsf{s}}{\to}_{\copyright}^{*} N$$

Proof. All conditions in Proposition 4.3.5 hold, namely

1. *Surface factorization* of $\to_{\mathsf{id}} \cup \to_{\sigma}$ holds.

2. *Substitutivity*: $\mapsto_{\mathsf{id}}$ and $\mapsto_{\sigma}$ are substitutive.

3. *Root linear swap.* For $\xi \in \{\mathsf{id}, \sigma\}$: $\underset{\neg\mathsf{s}}{\to}_{\beta_c} \cdot \mapsto_{\xi} \subseteq \mapsto_{\xi} \cdot \to_{\beta_c}^{=}$.

Indeed: (1) is Lemma 4.3.11, (2) is immediate, (3) is point 3. in Fact 4.3.9 and Fact 4.3.10. □

4.3.2 A Closer Look to id-steps, via Postponement

We establish a postponement result on which evaluation and normalization rely.

Observe that the rule $\to_{\mathsf{id}}$ overlaps with $\to_{\beta_c}$. We define as $\to_{\iota}$ a $\to_{\mathsf{id}}$ step that is not a $\to_{\beta_c}$ step.

$$\mapsto_{\iota} := \mapsto_{\mathsf{id}} \setminus \mapsto_{\beta_c} \qquad (\iota \text{ rule})$$

Clearly, $\to_{\copyright} = \to_{\beta_c} \cup \to_{\sigma} \cup \to_{\iota}$.

In the proofs, it is convenient to split $\to_{\beta_c}$ in steps which are also $\to_{\mathsf{id}}$ steps, and those which are not.

$$\mapsto_{\beta 1} := \mapsto_{\mathsf{id}} \cap \mapsto_{\beta_c} \qquad (\beta 1 \text{ rule})$$

$$\mapsto_{\beta 2} := \mapsto_{\beta_c} \setminus \mapsto_{\mathsf{id}} \qquad (\beta 2 \text{ rule})$$

Notice that $\to_{\beta_c} = \to_{\beta 1} \cup \to_{\beta 2}$.

We prove that $\to_{\iota}$ steps can be postponed after both $\to_{\beta_c}$ and $\to_{\sigma}$ steps, by using the Decreasing Diagrams technique (Theorem 1.2.8). The proof closely follows van Oostrom's proof of the postponement of η after β (in [vO20b])

Theorem 4.3.12 (Postponement of ι). *If $M \to_{\copyright}{}^* N$ then $M \to_{\sigma\beta_c}{}^* \cdot \to_{\iota}{}^* N$.*

Proof. Let $\lhd\ = \to_\iota$ and let $\blacktriangleright = \to_{\beta 1} \cup \to_{\beta 2} \cup \to_\sigma$.
We equip the labels $\{\beta 1, \beta 2, \sigma, \iota\}$ with the following order

$$\beta 2 < \iota \qquad \iota < \beta 1 \qquad \iota < \sigma$$

We prove that the pair of relations $\rhd, \blacktriangleright$ is decreasing by checking the following three local commutations hold (the technical details are in Appendix):

1. $\to_\iota \cdot \to_{\beta 2}\ \subseteq\ \to_{\beta 1}{}^* \cdot \to_{\beta 2}{}^= \cdot \to_{\beta 1}{}^* \cdot \to_\iota^*$ (see Lemma Lemma B.1.4)
2. $\to_\iota \cdot \to_{\beta 1}\ \subseteq\ \to_{\beta 1}{}^* \cdot \to_\iota^=$ (see Lemma Lemma B.1.3)
3. $\to_\iota \cdot \to_\sigma\ \subseteq (\to_\sigma \cup \to_{\beta 1})^* \cdot \to_\iota^=$ (see Lemma Lemma B.1.5)

Hence, by Theorem 1.2.8, the relations $\rhd$ and $\blacktriangleright$ commute. Otherwise stated, $\to_\iota$ postpones after $\to_{\beta_c} \cup \to_\sigma$:

$$\to_\iota^* \cdot \to_{\sigma\beta_c}{}^*\ \subseteq\ \to_{\sigma\beta_c}{}^* \cdot \to_\iota^*\ .$$

Or, equivalently (Lemma 1.2.4)

$$\to_{\copyright}\ \subseteq\ \to_{\sigma\beta_c}{}^* \cdot \to_\iota^*\ .$$

□

Corollary 4.3.13 (Surface Factorization, revisited).
If $M \to_{\copyright}^* N$ then $M \underset{\mathsf{s}}{\to}{}^*_{\sigma\beta_c} \cdot \underset{\neg\mathsf{s}}{\to}{}^*_{\sigma\beta_c} \cdot \to_\iota{}^* N$.

Proof. Immediate consequence of Theorem 4.3.12 and of surface factorization of $\beta_c\sigma$ reduction. □

4.3.3 Weak Factorization

Surface factorization (Theorem 4.3.1) and ι-postponement (Theorem 4.3.12) imply a result of weak factorization, from which we then obtain evaluation via weak β_c steps (Theorem 4.4.5).

Remarkably, weak factorization of a $\underset{\mathsf{s}}{\to}_{\sigma\beta_c}$-sequence has the property that the number of β_c steps is preserved. This property has no role with respect to evaluation, but it will be crucial when we investigate normalizing strategies in Chapter 5. For this reason we include it in the statement of Theorem 4.3.15.

Quantitative Linear Postponement. The condition in Lemma 1.2.5 — Hindley's strong postponement — can be refined into quantitative variants, which allow us to "count the steps" and are useful to establish termination properties.

Lemma 4.3.14 (Linear Postponement). *Given an ARS $(A, \to)$, assume $\to = \underset{\mathsf{e}}{\to} \cup \underset{\mathsf{i}}{\to}$.*

- *If* $\underset{i}{\rightarrow} \cdot \underset{e}{\rightarrow} \subseteq \underset{e}{\rightarrow} \cdot \underset{i}{\rightarrow}^{=}$*, then* $M \rightarrow^* N$ *implies* $M \underset{e}{\rightarrow}^* \cdot \underset{i}{\rightarrow}^* N$ *and the two sequences have the same number of* $\underset{e}{\rightarrow}$ *steps.*

- *Assume (for* $l \in L$ *an index set)* $\rightarrow_l = \underset{e}{\rightarrow}_l \cup \underset{i}{\rightarrow}_l$, $\underset{e}{\rightarrow} = \bigcup_l \underset{e}{\rightarrow}_l$, $\underset{i}{\rightarrow} = \bigcup_l \underset{i}{\rightarrow}_l$*. If*

  $$(\#) \qquad \underset{i}{\rightarrow}_j \cdot \underset{e}{\rightarrow}_k \subseteq \underset{e}{\rightarrow}_k \cdot \rightarrow_j \quad (j, k \in L)$$

  Then $M \rightarrow^* N$ *implies* $M \underset{e}{\rightarrow}^* \cdot \underset{i}{\rightarrow}^* N$ *and the two sequences have the same number of* $\rightarrow_l$ *steps for each* $l \in L$*.*

  Observe that in (#), the last step is $\rightarrow_j$*, not necessarily* $\underset{i}{\rightarrow}_j$*.*

Weak Factorization.

Theorem 4.3.15 (Weak factorization)**.**

1. $M \underset{s}{\rightarrow}^*_{\sigma\beta_c} N$ *implies* $M \underset{w}{\rightarrow}^*_{\sigma\beta_c} \cdot \underset{\neg w}{\rightarrow}^*_{\sigma\beta_c} N$*, where all steps are surface. Moreover, the two sequences have the same number of* β_c *steps.*

2. $M \underset{w}{\rightarrow}^*_{\sigma\beta_c} N$ *implies* $M \underset{w}{\rightarrow}^*_{\beta_c} \cdot \underset{w}{\rightarrow}^*_{\sigma} N$*. Moreover, the two sequences have the same number of* β_c *steps.*

Proof. In both claims, we use Lemma 4.3.14. Linearity allows us to count the β_c steps. In the proof, we write $\underset{w}{\rightarrow}$ (resp. $\underset{s}{\rightarrow}$) for $\underset{w}{\rightarrow}_{\sigma\beta_c}$ (resp. $\underset{s}{\rightarrow}_{\sigma\beta_c}$).

1. Let $\underset{i}{\rightarrow} = \underset{s}{\rightarrow} \setminus \underset{w}{\rightarrow}$ (*i.e.* $\underset{i}{\rightarrow}$ is a surface step whose redex is in the scope of λ). We prove linear postponement:

$$(\#) \quad \underset{i}{\rightarrow} \cdot \underset{w}{\rightarrow} \subseteq \underset{w}{\rightarrow} \cdot \underset{s}{\rightarrow}$$

Assume $M \underset{i}{\rightarrow} S \underset{w}{\rightarrow} N$. M and S have the same shape, which is not $[U]$, otherwise no weak or surface reduction is possible. We examine the cases.

- The step $S \underset{w}{\rightarrow} N$ has empty context:
  - $S \mapsto_{\beta_c} N$. Then $S = (\lambda x.P')[V] \mapsto_{\beta_c} P'\{V/x\} = N$, and $M = (\lambda x.P)[V] \underset{i}{\rightarrow} (\lambda x.P')[V]$. Therefore $(\lambda x.P)[V] \mapsto_{\beta_c} P\{V/x\} \underset{s}{\rightarrow} P'\{V/x\}$.
  - $S \mapsto_\sigma N$. Then $S = V((\lambda x.P)Q) \mapsto_\sigma (\lambda x.VP)Q$, and $M = V_0((\lambda x.P_0)Q_0) \underset{i}{\rightarrow} V((\lambda x.P)Q)$ (where exactly one among V_0, P_0, Q_0 reduces).
    Therefore, $M = V_0((\lambda x.P_0)Q_0) \mapsto_\sigma (\lambda x.V_0P_0)Q_0 \underset{i}{\rightarrow} (\lambda x.VP)Q$.

- The step $S \underset{w}{\rightarrow} N$ has non-empty context. Necessarily, we have $S = VQ \underset{w}{\rightarrow} VQ'$, with $Q \underset{w}{\rightarrow} Q'$:
  - Case $M = M_V Q \underset{i}{\rightarrow} VQ \underset{w}{\rightarrow} VQ'$, then $M_V Q \underset{w}{\rightarrow} M_V Q' \underset{i}{\rightarrow} VQ'$
  - Case $M = VM_Q \underset{i}{\rightarrow} VQ \underset{w}{\rightarrow} VQ'$. We conclude by *i.h.*.

Observe that we have proved more than (#), namely we proved

$$\underset{\mathsf{i}}{\to}_j \cdot \underset{\mathsf{w}}{\to}_k \subseteq \underset{\mathsf{w}}{\to}_k \cdot \underset{\mathsf{s}}{\to}_j \quad (j, k \in \{\beta_c, \sigma\})$$

Therefore, we conclude that the two sequences have the same number of β_c steps, by Lemma 4.3.14.

2. We prove $\underset{\mathsf{w}}{\to}_\sigma \cdot \underset{\mathsf{w}}{\to}_{\beta_c} \subseteq \underset{\mathsf{w}}{\to}_{\beta_c} \cdot \underset{\mathsf{w}}{\to}_\sigma^=$, and conclude by Lemma 4.3.14. □

Observe that in particular the following holds

$$M \underset{\mathsf{s}}{\to}^*_{\sigma\beta_c} [V] \text{ implies } M \underset{\mathsf{w}}{\to}^*_{\beta_c} \cdot \underset{\mathsf{w}}{\to}^*_\sigma \cdot \underset{\neg\mathsf{w}}{\to}^*_{\sigma\beta_c} [V]$$

and the two sequences have the same number of β_c steps.

4.4 Returning a Value

In this section we focus on *values*, which are the terms of interest in CbV λ-calculus. Recall that for weak reduction, values are exactly the normal forms of closed terms, *i.e.* of *programs*. In a computational setting, we are interested in knowing if a term M *returns* a value, *i.e.* if $M \to^*_{\copyright} [V]$ for some value V.

If M returns a value, is there a *deterministic* reduction which is guaranteed to return a value? The answer is positive. In fact, there are two such reductions: $\underset{\mathsf{w}}{\to}_{\beta_c}$ and $\mapsto_{\beta_c\sigma}$, as we see in Theorem 4.4.2. Recall that $\mapsto_{\sigma\beta_c} = (\mapsto_{\beta_c} \cup \mapsto_\sigma)$, indicating the relations which are directly obtained by orienting the monadic laws ($\to_{\beta_c}$ and $\to_\sigma$ are their contextual closure).

Fact 4.4.1.
- *Reduction $\underset{\mathsf{w}}{\to}_{\beta_c}$ is deterministic.*
- *Reductions $\mapsto_{\beta_c}$, $\mapsto_\sigma$, and their union $\mapsto_{\beta_c\sigma}$ are deterministic.*

Theorem 4.4.2 (Returning a value)**.** *The following are equivalent:*

1. *M returns a value,* i.e. *$M \to^*_{\copyright} [V]$.*
2. *The maximal $\underset{\mathsf{w}}{\to}_{\beta_c}$-sequence from M is* finite *and ends in a term $[W]$.*
3. *The maximal $\mapsto_{\beta_c\sigma}$-sequence from M is* finite *and ends in a term $[W]$.*

Proof. 1. $\Longrightarrow$ 2. is Theorem 4.4.5, which we prove in Section 4.4.1.
2. $\Longrightarrow$ 3. is Proposition 4.4.10, which we prove in Section 4.4.2.
3. $\Longrightarrow$ 1. is trivial. □

In the rest of the section we prove Theorem 4.4.2. Note that our analysis is not restricted to closed terms.

Remark 4.4.3. An open term may well return a value. For example, $[\lambda x.[z]]$.

4.4.1 Values via Weak β_c Steps

The factorization results allow us to prove that $\underset{\mathsf{w}}{\rightarrow}_{\beta_c}$-steps suffice to return a value. This is an immediate consequence of surface factorization (Theorem 4.3.1), ι postponement (Theorem 4.3.12), and weak factorization (Theorem 4.3.15), and the fact that internal steps, ι steps, and σ steps cannot produce a return value.

Lemma 4.4.4. *If $M \rightarrow_{\circledcirc} [V]$ with a step which is* not $M \underset{\mathsf{w}}{\rightarrow}_{\beta_c} [V]$, *then $M = [W]$.*

Proof. Indeed, one can easily check the following

- If $M \rightarrow_{\sigma} [V]$, then $M = [W]$.
- If $M \rightarrow_{\iota} [V]$, then $M = [W]$.
- If $M \underset{\neg\mathsf{w}}{\rightarrow}_{\beta_c} [V]$, then $M = [W]$.

Notice that $\underset{\neg\mathsf{s}}{\rightarrow}_{\circledcirc} \subseteq \underset{\neg\mathsf{w}}{\rightarrow}_{\circledcirc}$, and so $\underset{\neg\mathsf{s}}{\rightarrow}_{\beta_c} \subseteq \underset{\neg\mathsf{w}}{\rightarrow}_{\beta_c}$ □

Theorem 4.4.5 (Values via β_c steps)**.** *The following are equivalent*

1. *$M \rightarrow_{\circledcirc}^{*} [V]$ (for some $V \in Val$);*
2. *$M \underset{\mathsf{w}}{\rightarrow}_{\beta_c}^{*} [U]$ (for some $U \in Val$).*

Proof. Point 2 trivially implies Point 1.
Let us show that Point 1 entails Point 2.

If $M \rightarrow_{\circledcirc}^{*} [V]$ then by Corollary 4.3.13 $M \underset{\mathsf{s}}{\rightarrow}_{\sigma\beta_c}^{*} \cdot \underset{\neg\mathsf{s}}{\rightarrow}_{\sigma\beta_c}^{*} \cdot \rightarrow_{\iota}^{*} [V]$.

By weak factorization (Theorem 4.3.15.1-2), and since $\underset{\neg\mathsf{s}}{\rightarrow} \subseteq \underset{\neg\mathsf{w}}{\rightarrow}$, we have

$$M \underset{\mathsf{w}}{\rightarrow}_{\beta_c}^{*} M' \underset{\mathsf{w}}{\rightarrow}_{\sigma}^{*} \cdot \underset{\neg\mathsf{w}}{\rightarrow}_{\sigma\beta_c}^{*} \cdot \underset{\neg\mathsf{s}}{\rightarrow}_{\sigma\beta_c}^{*} \cdot \rightarrow_{\iota}^{*} [V]$$

By iterating Lemma 4.4.4 from $[V]$ backwards, we have that all terms in the sequence from M' to $[V]$ are returned values. So in particular, M' has shape $[U]$. □

Remark 4.4.6. Theorem 4.4.5 was already claimed in [dT20] and here it is Lemma 9.1.4, for closed terms, involving a convergence predicate. However, the inductive argument in [dT20] did not suffice to produce a proof, here with this operational investigation we have fixed that proof.

4.4.2 Values via $\sigma\beta_c$ Root Steps

We show also an alternative way to evaluate a term in $\lambda_{\circledcirc}$. Let us call *root steps* the relations $\mapsto_{\beta_c}, \mapsto_{\sigma}$ and $\mapsto_{\mathsf{id}}$. They suffice to compute a value (in fact the first two suffice), without need for a contextual closure.

Note that this property holds only because terms are restricted to computations (for example, in Plotkin's CbV λ-calculus, $(\boldsymbol{II})(\boldsymbol{II})$ has a reduction step, but it is not itself a redex, so $(\boldsymbol{II})(\boldsymbol{II}) \not\mapsto_{\beta_v}$).

Closed computations have the following property, which is immediate.

Fact 4.4.7 (Closed Computation)**.** *If M is a closed computation, then*

- *either M has shape $[V]$ (*i.e.* M is weak-normal and surface-normal),*
- *or M is a β_c-redex, or is a σ-redex.*

As a consequence, every closed normal form is a return $[V]$.

More generally, the same holds for any computation which returns a value (Corollary 4.4.9).

Lemma 4.4.8. *Assume $M \underset{\mathsf{w}}{\to}^{*}_{\beta_c} [W]$. Then*

- *either $M = [W]$,*
- *or $M = (\lambda x.P)M'$ and $M' \underset{\mathsf{w}}{\to}^{*}_{\beta_c} [U]$, for some value U.*

Therefore M has shape $V_0(V_1...(V_n[U]))$, where each V_i is an abstraction, and moreover $V_0(V_1...(V_{n-1}(\lambda x_n.P_n)[U])) \underset{\mathsf{w}}{\to}_{\beta_c} V_1(V_2...(V_{n-1}P_n\{U/x_n\})$.

Corollary 4.4.9 (Progression via root steps)**.** *Assume M returns a value ($M \to^{*}_{\copyright} [W]$). Then M is either a β_c-redex, or a σ-redex, or it has shape $[U]$.*

Corollary 4.4.9 states a *progression* results: a $\mapsto_{\sigma\beta_c}$-sequence from M may only end in a return value. We still need to verify that such a sequence terminates.

Proposition 4.4.10 (week steps and root steps)**.** *If $M \underset{\mathsf{w}}{\to}^{*}_{\beta_c} [W]$ then $M \mapsto^{*}_{\beta_c\sigma} [W]$. Moreover, the two sequences have the same number of β_c steps.*

Proof. By induction on the number k of $\underset{\mathsf{w}}{\to}_{\beta_c}$ steps. If $k = 0$ the claim holds trivially. Otherwise, $M \underset{\mathsf{w}}{\to}_{\beta_c} M_1 \underset{\mathsf{w}}{\to}^{*}_{\beta_c} [W]$ and by *i.h.*

$$(\#) \quad M_1 \mapsto^{*}_{\beta_c\sigma} [W].$$

- If M is β_c-redex, then $M \mapsto_{\beta_c} M_1$, and the claim is proved.
- If M is a σ-redex, observe that by Lemma 4.4.8,
  - $M = \lambda x_0.P_0(...(\lambda x_{n-1}.P_{n-1}(\lambda x_n.P_n)[U]))$, and
  - $M_1 = \lambda x_0.P_0(...(\lambda x_{n-1}.P_{n-1}(P_n\{U/x_n\})$.

  We apply to M all possible $\mapsto_\sigma$ steps, obtaining

  $M \mapsto^{*}_{\sigma} (\lambda x_{n-1}.(...P_{n-1}))((\lambda x_n.P_n)[U]) \mapsto_\sigma \lambda x_n.(\lambda x_{n-1}.(...P_{n-1})P_n))[U] = M'$ which is a β_c-redex, so $M' \mapsto_{\beta_c} \lambda x_{n-1}.(...P_{n-1})(P_n\{U/x_n\})$.

  We observe that $M_1 \mapsto^{*}_{\sigma} \lambda x_{n-1}.(...P_{n-1})(P_n\{U/x_n\})$. We conclude, by using (#), and the fact that $\mapsto_{\sigma\beta_c}$ is deterministic. □

The converse is also true and immediate. Putting together Proposition 4.4.10 and Theorem 4.4.5, we can finally prove that *root* steps $\mapsto_{\beta_c}$ and $\mapsto_\sigma$ suffice to compute a value, without need for contextual closure.

Theorem 4.4.11 (Values via root $\beta_c\sigma$ steps)**.** *The following are equivalent*

1. *$M \to^{*}_{\copyright} [V]$ (for some $V \in$ Val);*
2. *$M \mapsto^{*}_{\beta_c\sigma} [U]$ (for some $U \in$ Val).*

4.4.3 Observational Equivalence

In this section we summarize some consequences of Theorem 4.4.5. We recast the notion of observational equivalence introduced by Plotkin for CbV λ-calculus in $\lambda_{©}$. Informally, two terms are observationally equivalent if they can be substituted for each other in all contexts without observing any difference in their overall behaviour. For a computation M, the "behaviour" of interest here is whether the term returns a value. As we have seen, if M is a program, then M returns a value whenever its evaluation by $\underset{\mathsf{w}}{\rightarrow}_{\beta_v}$ terminates. We write this $M \Downarrow$ (the evaluation of M halts). Since $\underset{\mathsf{w}}{\rightarrow}_{\beta_c}$ is a deterministic reduction, halting is deterministic.

Following [Plo75], we define

Definition 4.4.12 (Observational equivalence.). *$M \cong N$ if, for each context C, $\mathsf{C}\langle M\rangle \Downarrow$ if and only if $\mathsf{C}\langle N\rangle \Downarrow$.*

An easy argument, similar to that in [Plo75] gives:

Theorem 4.4.13 (Adequacy). *If $M \rightarrow_{©}^{*} M'$ then $M \Downarrow$ if and only if $M' \Downarrow$.*

Corollary 4.4.14. *If $M =_{©} N$ then $M \cong N$.*

CHAPTER 5

OPERATIONAL PROPERTIES: NORMALIZATION

Normalization and Normalizing Strategies In this chapter we study normalization and normalizing strategies in $\lambda_{\copyright}$.

Reduction $\to_{\copyright}$ is obtained by adding $\to_{\iota}$ and $\to_{\sigma}$ to $\to_{\beta_c}$. What is the role of ι steps and σ steps with respect to normalization in $\lambda_{\copyright}$? Perhaps surprisingly, despite the fact that both $\to_{\iota}$ and $\to_{\sigma}$ are strongly normalizing (Lemma 5.1.3 below), their role is quite different.

1. Unlike the case of terms returning a value, which we studied in Section 4.4, β_c steps do not suffice to capture $\lambda_{\copyright}$-normalization, in that σ steps may turn a β_c-normalizable term into one that is not $\lambda_{\copyright}$-normalizable. That is, σ steps are *essential* to normalization in $\lambda_{\copyright}$ (see Section 5.2).
2. ι steps instead are *irrelevant* for normalization in $\lambda_{\copyright}$, in the sense that they play no role, in the sense that a term has a $\lambda_{\copyright}$-normal form if and only if it has a $\sigma\beta_c$-normal form (see Section 5.1).

Taking into account both Point 1 and Point 2, in 5.3 we define two families of normalizing strategies in $\lambda_{\copyright}$. The first one, quite constrained, relies on an *iteration of weak reduction* $\underset{\mathsf{w}}{\to}_{\copyright}$. The second one, more liberal, is based on an *iteration of surface reduction* $\underset{\mathsf{s}}{\to}_{\copyright}$. The interest of a rather liberal strategy is that it provides *a more versatile framework* to reason about program transformations, or optimization techniques such as parallel implementation.

Technical Lemmas: preservation of normal forms. We collect here some properties of preservation of (full, weak and surface) normal forms, which we will use along the section. The easy proofs are in Appendix B.2.

Lemma 5.0.1. *Assume $M \to_{\iota} N$.*

1. *M is β_c-normal if and only if N is β_c-normal.*

2. If M is σ-normal, so is N.

Lemma 5.0.2. *If $M \to_\sigma N$, then: M is $\underset{\mathsf{w}}{\to}_{\beta_c}$-normal if and only if so is N.*

Lemma 5.0.2 fails if we replace $\underset{\mathsf{w}}{\to}_{\beta_c}$ with $\underset{\mathsf{s}}{\to}_{\beta_c}$. Indeed, $M \to_\sigma N$ for some M $\underset{\mathsf{s}}{\to}_{\beta_c}$-normal does not imply that N is $\underset{\mathsf{s}}{\to}_{\beta_c}$-normal, as we will see in Example 5.2.1.

Lemma 5.0.3. *Let $\to = \to_\sigma \cup \to_{\beta_c}$ and $\mathsf{e} \in \{\mathsf{w}, \mathsf{s}\}$. Assume $M \underset{\neg\mathsf{e}}{\to} N$. Then, M is e-normal if and only if N is e-normal.*

5.1 Irrelevance of ι Steps for Normalization

We show that postponement of ι steps (Theorem 4.3.12) implies that $\to_\iota$ steps have no impact on normalization, *i.e.* whether a term M has or not a $\lambda_{\copyright}$-normal form. Indeed, saying that M has a $\lambda_{\copyright}$-normal form is equivalent to saying that M has a $\beta_c\sigma$-normal form.

On the one hand, if $M \to^*_{\sigma\beta_c} N$ and N is $\sigma\beta_c$-normal, to reach a $\lambda_{\copyright}$-normal form it suffices to extend the reduction with ι steps to a ι-normal form (since $\to_\iota$ is terminating, Lemma 5.1.3). Notice that here we use Lemma 5.0.1. On the other hand, the proof that $\lambda_{\copyright}$-normalization implies $\beta_c\sigma$-normalization is trickier, because σ-normal forms are not preserved by performing a ι step backward (the converse of Lemma 5.0.1.2 is false). Here is a counterexample.

Example 5.1.1. Consider

$$(\lambda x.x[x])(\boldsymbol{I}(z[z])) \to_\iota (\lambda x.x[x])(z[z])$$

where $(\lambda x.x[x])(z[z])$ is σ-normal (actually $\lambda_{\copyright}$-normal) but $(\lambda x.x[x])(\boldsymbol{I}(z[z]))$ is not σ-normal.

Consequently, the fact that M has a $\lambda_{\copyright}$-normal form N means (by postponement of $\to_\iota$) that $M \to^*_{\beta_c\sigma} P \to^*_\iota N$ for some term P that (in Lemma 5.0.1) is guaranteed to be β_c-normal only, not σ-normal. To prove that M has a $\beta_c\sigma$-normal form is not even enough to take the σ-normal form of P, because a σ step can create a β_c-redex. To solve the problem, we need the following technical lemma.

Lemma 5.1.2. *Assume $M \to^k_\iota N$, where $k > 0$, and N is $\sigma\iota$-normal. If M is not σ-normal, then there exist M' and N' such that either $M \to_\sigma M' \to_\iota N' \to^{k-1}_\iota N$ or $M \to_\sigma M' \to_{\beta_c} N' \to^{k-1}_\iota N$.*

We also use the fact that $\to_\sigma$ and $\to_\iota$ are strongly normalizing (Lemma 5.1.3). Instead of proving that $\to_\sigma$ and $\to_\iota$ are —separately— so, we state a more general result (its proof is in Appendix B.2).

Lemma 5.1.3 (Termination of $\sigma\mathsf{id}$)**.** *$\to_{\mathsf{id}} \cup \to_\sigma$ is strongly normalizing.*

Now we have all the elements to prove the following.

Theorem 5.1.4 (Irrelevance of ι for normalization)**.** *The following are equivalent:*

1. *M is $\lambda_{©}$-normalizing;*
2. *M is $\sigma\beta_c$-normalizing.*

Proof. **(1) $\Longrightarrow$ (2):** If M is $\lambda_{©}$-normalizing, then $M \to_{©}^* N$ for some N λ_c-normal. By postponement of ι steps (Theorem 4.3.12), for some P we have

$$M \to_{\beta_c\sigma}^* P \to_{\iota}^* N \tag{5.1}$$

For any sequence of the form (5.1), let $w(P) = (w_\iota(P), w_\sigma(P))$, where $w_\iota(P)$ and $w_\sigma(P)$ are the lengths of the maximal ι-sequence and of the maximal σ-sequence from T, respectively. Both $w_\iota(P)$ and $w_\sigma(P)$ are well-defined because $\to_\iota$ and $\to_\sigma$ are strongly normalizing (Lemma 5.1.3).

We proceed by induction on $w(P)$ ordered lexicographically to prove that $M \to_{\beta_c\sigma}^* P' \to_\iota^* N$ for some P' $\beta_c\sigma$-normal (and so M is $\beta_c\sigma$-normalizing). By Lemma 5.0.1.1, P is β_c-normal in (5.1).

- If $w(P) = (0, h)$ then $P = N$, so P is σ-normal and hence $\beta_c\sigma$-normal.
- If $w(P) = (k, 0)$, then P is σ-normal and hence $\beta_c\sigma$-normal.
- Otherwise, $w(P) = (k, h)$ with $k, h > 0$. By Lemma 5.1.2, $M \to_{\beta_c\sigma}^* P' \to_\iota^* N$ for some P' with $w(P') < w(P)$: indeed, either $w(P') = (k, h-1)$, or $w(P') = (k-1, h)$. By *i.h.*, we can conclude.

(2) $\Longrightarrow$ (1): If M is $\beta_c\sigma$-normalizing, then $M \to_{\beta_c\sigma}^* N$ for some N $\beta_c\sigma$-normal. As $\to_\iota$ is strongly normalizing (Lemma 5.1.3), $N \to_\iota^* P$ for some P ι-normal. By Lemma 5.0.1.1-2, P is also β_c-normal and σ-normal. Summing up, $M \to_{©}^* P$ with P $\lambda_{©}$-normal, that is, M is $\lambda_{©}$-normalizing. □

5.2 The Essential Role of σ Steps for Normalization

In $\lambda_{©}$, for normalization, σ steps play a crucial role, unlike ι steps. Indeed, σ steps can unveil "hidden" β_c-redexes in a term. Let us see this with an example, where we consider a term that is β_c-normal (no further β_c step is possible), but is diverging in $\lambda_{©}$ and this divergence is "unblocked" by a σ step.

Example 5.2.1 (Normalization in $\lambda_{©}$)**.** Let $\Delta = \lambda x.x[x]$. Consider the σ step

$$M_z = \Delta((\lambda y.[\Delta])(z[z])) \to_\sigma (\lambda y.\Delta[\Delta])(z[z]) = N_z$$

M_z is β_c-normal, but not $\lambda_{©}$-normal. In fact, M_z is diverging in $\lambda_{©}$:

$$M_z \to_\sigma N_z \to_{\beta_c} N_z \to_{\beta_c} \ldots$$

Note that the σ step is weak and that N_z is normal for $\underset{\mathsf{w}}{\to}_{\beta_c}$ but not for $\underset{\mathsf{s}}{\to}_{\beta_c}$.

The fact that a σ step can unblock a hidden β_c-redex is not limited to open terms. Indeed, $\lambda z.M_z$ is closed and β_c-normal, but divergent in $\lambda_{©}$:

$$\lambda z.M_z \to_\sigma \lambda z.N_z \to_{\beta_c} \lambda z.N_z \to_{\beta_c} \ldots$$

The example shows that, contrary to ι steps, σ steps are essential to determine whether a term has or not normal form in $\lambda_{©}$. This fact is in accordance with the semantics. First, it can be shown that the term M' above and $\Delta[\Delta]$ are observational equivalent. Second, the denotational models and type systems studied in [Ehr12, dT20] and here in Part II interpret M' in the same ways as $\Delta[\Delta]$, which is a β_c-divergent term. It is then reasonable to expect that the two terms have the same operational behavior in $\lambda_{©}$. Adding σ steps to β_c-reduction is a way to obtain this: both M' and $\Delta[\Delta]$ are divergent in $\lambda_{©}$. Said differently, σ-reduction restricts the set of normal forms in $\lambda_{©}$, so as to exclude some β_c-normal (but not $\sigma\beta_c$-normal) forms that are semantically meaningless.

Actually, σ-reduction *can only restrict* the set of terms having a normal form: it may turn a β_c-normal form into a term that diverges in $\lambda_{©}$, but it cannot turn a β_c-diverging term into a $\lambda_{©}$-normalizing one. To prove this (Proposition 5.2.3), we rely on the following lemma.

Lemma 5.2.2. *If M is not β_c-normal and $M \to_\sigma L$, then L is not β_c-normal and $L \to_{\beta_c} N$ implies $M \to_{\beta_c} \cdot \to_\sigma^= N$.*

Roughly, Lemma 5.2.2 says that σ step on a term that is not β_c-normal cannot erase a β_c-redex, and hence it can be postponed. Lemma 5.2.2 does not contradict Example 5.2.1, because the former talks about a σ step on a term that is not β_c-normal, whereas the start terms in Example 5.2.1 are β_c-normal.

Proposition 5.2.3. *If a term is $\sigma\beta_c$-normalizing (resp. strongly $\sigma\beta_c$-normalizing), then it is β_c-normalizing (resp. strongly β_c-normalizing).*

Proof. As $\to_{\beta_c} \subseteq \to_{\beta_c\sigma}$, any infinite β_c-sequence is an infinite $\sigma\beta_c$-sequence. So, if M is not strongly β_c-normalizing, it is not strongly $\sigma\beta_c$-normalizing.

We prove now the part of the statement about normalization. If M is $\sigma\beta_c$-normalizing, there exists a reduction sequence $\mathfrak{s} : M \to_{\beta_c\sigma}^* N$ with N $\sigma\beta_c$-normal. Let $|\mathfrak{s}|_\sigma$ be the number of steps in s, and let $|\mathfrak{s}|_{\beta_c}$ be the number of β_c steps after the last σ step in s (when $|\mathfrak{s}|_\sigma = 0$, $|\mathfrak{s}|_{\beta_c}$ is just the length of s). We prove by induction on $(|\mathfrak{s}|_\sigma, |\mathfrak{s}|_{\beta_c})$ ordered lexicographically that M is β_c-normalizing. There are three cases.

1. If $\mathfrak{s}$ contains only β_c steps ($|\mathfrak{s}|_\sigma = 0$), then $M \to_{\beta_c}^* N$ and we are done.

2. If $\mathfrak{s} : M \to_{\beta_c\sigma}^* L \to_\sigma^+ N$ ($\mathfrak{s}$ ends with a non-empty sequence of σ steps), then L is β_c-normal by Lemma 5.2.2, as N is β_c-normal; by *i.h.* applied to the sequence $\mathfrak{s}' : M \to_{\beta_c\sigma}^* L$ (as $|\mathfrak{s}'|_\sigma < |\mathfrak{s}|_\sigma$), M is β_c-normalizing.

3. Otherwise, $\mathfrak{s} : M \to_{\beta_c\sigma}^* L \to_\sigma P \to_{\beta_c} Q \to_{\beta_c}^* N$ ($L \to_\sigma P$ is the last σ step in $\mathfrak{s}$, followed by a β_c step). By Lemma 5.2.2, either there is a sequence $\mathfrak{s}' : M \to_{\beta_c\sigma}^* L \to_{\beta_c} R \to_\sigma Q \to_{\beta_c}^* N$, then $|\mathfrak{s}'|_\sigma = |\mathfrak{s}|_\sigma$ and $|\mathfrak{s}'|_{\beta_c} < |\mathfrak{s}|_{\beta_c}$; or $s' : M \to_{\beta_c\sigma}^* L \to_{\beta_c} Q \to_{\beta_c}^* N$ and then $|\mathfrak{s}'|_\sigma < |\mathfrak{s}|_\sigma$. In both cases $(|\mathfrak{s}'|_\sigma, |\mathfrak{s}'|_{\beta_c}) < (|\mathfrak{s}|_\sigma, |\mathfrak{s}|_{\beta_c})$, so by *i.h.* M is β_c-normalizing. ☐

5.3 Normalizing Strategies

Irrelevance of ι steps (Theorem 5.1.4) implies that to define a normalizing strategy for $\lambda_{\copyright}$, it suffices to define a normalizing strategy for $\sigma\beta_c$. We do so by iterating either *surface* or *weak* reduction. Our definition of $\sigma\beta_c$-normalizing strategy and the proof of normalization (Theorem 5.3.4) is *parametric* on either. The difficulty here is that both weak and surface reduction are *non-deterministic.* The key property we need in the proof is that the reduction which we iterate is *uniformly normalizing* (see Definition 1.2.1). In Section 5.3.1 we establish that this hold for both, weak and surface reduction. While uniform normalization is easy to establish for the former, it is *non-trivial* for the latter. The proof is rather sophisticated. Here we recap the fruits of the careful analysis of the number of β_c steps in Section 4.3.3.

In Section 5.3.2 we formalize the strategy and tackle normalization.

Notation. Since we are now only concerned with $\sigma\beta_c$ steps, for the sake of readability in the rest of the section, we often write $\rightarrow$ for $\rightarrow_{\sigma\beta_c}$, $\underset{\mathsf{s}}{\rightarrow}$ and $\underset{\mathsf{w}}{\rightarrow}$ for $\underset{\mathsf{s}}{\rightarrow}_{\sigma\beta_c}$ and $\underset{\mathsf{w}}{\rightarrow}_{\sigma\beta_c}$, respectively.

Understanding uniform normalization The fact that $\underset{\mathsf{s}}{\rightarrow}$ and $\underset{\mathsf{w}}{\rightarrow}$ are uniformly normalizing is key in the definition of normalizing strategy and deserves some discussion.

Assume we iterate surface reduction. The heart of the normalization proof is that if M has a $\rightarrow$-normal form N, and we perform surface steps, the process terminates, that is, we reach a *surface normal form.* Notice that surface factorization only guarantees that *there exists* a $\underset{\mathsf{s}}{\rightarrow}$-sequence such that if $M \rightarrow^* N$ then $M \underset{\mathsf{s}}{\rightarrow}^* U \underset{\neg\mathsf{s}}{\rightarrow}^* N$, where U is $\underset{\mathsf{s}}{\rightarrow}$-normal. The *existential* quantification is crucial here because $\underset{\mathsf{s}}{\rightarrow}$ is *not* a deterministic reduction. *Uniform normalization* of $\underset{\mathsf{s}}{\rightarrow}$ turns the existential into an *universal* quantification. If M has a normal form (and so a fortiori a surface normal form), then *every* sequence of $\underset{\mathsf{s}}{\rightarrow}$-steps will terminate. The normalizing strategy then iterates this process.

5.3.1 Uniform Normalization of Weak and Surface Reduction

We prove that both weak and surface reduction are uniformly normalizing, *i.e.* for $\mathsf{e} \in \{\mathsf{w}, \mathsf{s}\}$, if a term M is $\underset{\mathsf{e}}{\rightarrow}$-normalizing, then it is strongly $\underset{\mathsf{e}}{\rightarrow}$-normalizing. In both cases, the proof relies on the fact that all maximal $\underset{\mathsf{e}}{\rightarrow}$-sequences from a given term M have the same number of β_c steps.

Fact 5.3.1 (Number of β_c steps)**.** *Given a $\rightarrow_{\sigma\beta_c}$-sequence $\mathfrak{s}$, the number of its β_c steps is finite if and only if $\mathfrak{s}$ is finite.*

Proof. The left-to-right implication is obvious. The right-to-left is an immediate consequence of the fact that $\rightarrow_\sigma$ is strongly normalizing (Lemma 5.1.3). □

A *maximal* $\underset{\mathsf{e}}{\to}$-sequence from M is either infinite, or ends in a e-normal form. Theorem 5.3.3 states that for $\mathsf{e} \in \{\mathsf{w}, \mathsf{s}\}$, all *maximal* $\underset{\mathsf{e}}{\to}$-sequence from the same term M have the *same behaviour*, also quantitatively (with respect to the number of β_c steps). The proof relies on the following lemma. Recall that weak reduction is not confluent (see Example 4.2.1); however, $\underset{\mathsf{w}}{\to}_{\beta_c}$ is deterministic.

Lemma 5.3.2 (Invariant)**.** *Given M,* every sequence *$M \underset{\mathsf{w}}{\to}^*_{\sigma\beta_c} S$ where S is $\underset{\mathsf{w}}{\to}_{\beta_c}$-normal has* the same number *k of β_c steps. Moreover*

1. *the unique maximal $\underset{\mathsf{w}}{\to}_{\beta_c}$-sequence from M has length k, and*

2. *there exists a sequence $M \underset{\mathsf{w}}{\to}_{\beta_c}{}^k L \underset{\mathsf{w}}{\to}^*_{\sigma} S$.*

Proof. The argument is illustrated by Figure 5.1. Assume that k is the number of β_c steps in a sequence $\mathfrak{s}\colon M \underset{\mathsf{w}}{\to}^*_{\sigma\beta_c} S$ where S is $\underset{\mathsf{w}}{\to}_{\beta_c}$-normal. By weak factorization (Theorem 4.3.15.2) there is a sequence $M \underset{\mathsf{w}}{\to}_{\beta_c}{}^k L \underset{\mathsf{w}}{\to}^*_{\sigma} S$ which has the same number k of β_c steps. Since S is $\underset{\mathsf{w}}{\to}_{\beta_c}$-normal, so is L (Lemma 5.0.2). Hence, $M \underset{\mathsf{w}}{\to}_{\beta_c}{}^k L$ is a maximal $\underset{\mathsf{w}}{\to}_{\beta_c}$-sequence from M, which is unique because $\underset{\mathsf{w}}{\to}_{\beta_c}$ is deterministic. □

Theorem 5.3.3 (Uniform normalization)**.**

1. *The reduction $\underset{\mathsf{w}}{\to}_{\sigma\beta_c}$ is uniformly normalizing.*

2. *The reduction $\underset{\mathsf{s}}{\to}_{\sigma\beta_c}$ is uniformly normalizing.*

Moreover, all maximal $\underset{\mathsf{w}}{\to}_{\sigma\beta_c}$-sequences (resp. all maximal $\underset{\mathsf{s}}{\to}_{\sigma\beta_c}$-sequences) from the same term M have the same number of β_c steps.

Proof. We write $\to$ (resp. $\underset{\mathsf{w}}{\to}, \underset{\mathsf{s}}{\to}$) for $\to_{\sigma\beta_c}$ (resp. $\underset{\mathsf{w}}{\to}_{\sigma\beta_c}, \underset{\mathsf{s}}{\to}_{\sigma\beta_c}$).

Claim 1. Assume M is a term such that $M \underset{\mathsf{w}}{\to}^* N$, where N is $\underset{\mathsf{w}}{\to}$-normal, and so, in particular $\underset{\mathsf{w}}{\to}_{\beta_c}$-normal. By Lemma 5.3.2, $M \underset{\mathsf{w}}{\to}_{\beta_c}{}^k L \underset{\mathsf{w}}{\to}^*_{\sigma} N$ where $M \underset{\mathsf{w}}{\to}_{\beta_c}{}^k L$ is the (unique) maximal $\underset{\mathsf{w}}{\to}_{\beta_c}$-sequence from M. We prove that no $\underset{\mathsf{w}}{\to}$-sequence from M may have more than k β_c steps. Indeed, every sequence $\mathfrak{s} : M \underset{\mathsf{w}}{\to}^* N'$ can be factorized (Theorem 4.3.15.2) into $M \underset{\mathsf{w}}{\to}^*_{\beta_c} L' \underset{\mathsf{w}}{\to}^*_{\sigma} N'$ with the same number of β_c steps as $\mathfrak{s}$, and $M \underset{\mathsf{w}}{\to}^*_{\beta_c} L'$ is a prefix of $M \underset{\mathsf{w}}{\to}_{\beta_c}{}^k L$ (as $\underset{\mathsf{w}}{\to}_{\beta_c}$ is deterministic).

We deduce that no infinite $\underset{\mathsf{w}}{\to}$-sequence from M is possible (by Fact 5.3.1).

Claim 2. Assume that $M \underset{\mathsf{s}}{\to}^* N$ with N $\underset{\mathsf{s}}{\to}$-normal. Recall that $\underset{\mathsf{s}}{\to}$ is confluent (Proposition 4.2.5.2), so N is the unique $\underset{\mathsf{s}}{\to}$-normal form of M.

First, by induction on N, we prove that given a term M,

(#) all sequences $M \underset{\mathsf{s}}{\to}^* N$ have the same number of β_c steps.

$$\begin{tikzcd} & & S \\ M \arrow[r, two heads, "\mathsf{w}", "\beta_c"'] & \cdot \arrow[ur, "\mathsf{w}", "\sigma"'] \arrow[dr, "\mathsf{w}", "\sigma"'] & \\ & & S' \end{tikzcd}$$

Figure 5.1: Weak reduction

$$\begin{tikzcd} & & S_1 \arrow[d, "\sigma"] \arrow[dr] & \\ M \arrow[r, two heads, "\mathsf{w}", "\beta_c"'] & L \arrow[ur, "\mathsf{w}", "\sigma"'] \arrow[dr, "\mathsf{w}", "\sigma"'] & S_3 \arrow[r, two heads] & N \\ & & S_2 \arrow[u, "\sigma"'] \arrow[ur] & \end{tikzcd}$$

Figure 5.2: Surface reduction

Let $\mathfrak{s}_1, \mathfrak{s}_2$ be two such sequences. Figure 5.2 illustrates the argument.

By weak factorization (Theorem 4.3.15.1), there is a sequence $M \underset{\mathsf{w}}{\rightarrow}^* S_1 \underset{\neg\mathsf{w}}{\rightarrow}^* N$ (resp. $M \underset{\mathsf{w}}{\rightarrow}^* S_2 \underset{\neg\mathsf{w}}{\rightarrow}^* N$) with the same number of β_c steps as $\mathfrak{s}_1$ (resp. $\mathfrak{s}_2$), and whose steps are all surface. Note that S_1 and S_2 are $\underset{\mathsf{w}}{\rightarrow}$-normal (by Lemma 5.0.3, because N is in particular $\underset{\mathsf{w}}{\rightarrow}$-normal), and so in particular $\underset{\mathsf{w}}{\rightarrow}_{\beta_c}$-normal. By Lemma 5.3.2, $M \underset{\mathsf{w}}{\rightarrow}^* S_1$, $M \underset{\mathsf{w}}{\rightarrow}^* S_2$ have the same number k of β_c steps, and so do the sequences $\mathfrak{s}'_1 : M \underset{\mathsf{w}}{\rightarrow}_{\beta_c}{}^k\, L \underset{\mathsf{w}}{\rightarrow}^*_\sigma S_1$ and $\mathfrak{s}'_2 : M \underset{\mathsf{w}}{\rightarrow}_{\beta_c}{}^k\, L \underset{\mathsf{w}}{\rightarrow}^*_\sigma S_2$.

To prove (#), we have to show that the sequences $\mathfrak{s}''_1 : S_1 \underset{\neg\mathsf{w}}{\rightarrow}^* N$ has the same number of β_c steps $\mathfrak{s}''_2 : S_2 \underset{\neg\mathsf{w}}{\rightarrow}^* N$.

By confluence of $\underset{\mathsf{s}}{\rightarrow}_\sigma$ (Proposition 4.2.5.1), $S_1 \underset{\mathsf{s}}{\rightarrow}^*_\sigma S_3 \underset{\mathsf{s}}{\leftarrow}^*_\sigma S_2$, for some S_3, and (by confluence of $\underset{\mathsf{s}}{\rightarrow}$, Proposition 4.2.5.2) there is a sequence $\mathfrak{t}$: $S_3 \underset{\mathsf{s}}{\rightarrow}^* N$. By Lemma 5.0.2, since S_1, S_2 are $\underset{\mathsf{w}}{\rightarrow}$-normal, terms in these sequences are $\underset{\mathsf{w}}{\rightarrow}$-normal, and therefore all steps are not only surface, but also $\underset{\neg\mathsf{w}}{\rightarrow}$ steps. That is, $S_1 \underset{\neg\mathsf{w}}{\rightarrow}^*_\sigma S_3$, $S_2 \underset{\neg\mathsf{w}}{\rightarrow}^*_\sigma S_3$, and $\mathfrak{t}$: $S_3 \underset{\neg\mathsf{w}}{\rightarrow}^* N$. We conclude that S_1, S_2, S_3 and N have the same shape (Fact 4.3.2).

We examine the shape of N, and prove claim (#) by showing that $\mathfrak{s}''_1$ and $\mathfrak{s}''_2$ have the same number of β_c steps as $\mathfrak{t}$ (note that the sequences $S_1 \underset{\neg\mathsf{w}}{\rightarrow}^*_\sigma S_3$ and $S_2 \underset{\neg\mathsf{w}}{\rightarrow}^*_\sigma S_3$ have no β_c steps).

- $N = [V]$. In this case, $N = S_1 = S_2$, and the claim (#) is immediate.
- $N = (\lambda x.P)Q$, and $S_i = (\lambda x.P_i)Q_i$ (for $i \in \{1, 2, 3\}$). We have $P_i \underset{\mathsf{s}}{\rightarrow}^* P$ and $Q_i \underset{\mathsf{s}}{\rightarrow}^* Q$. Since P and Q are $\underset{\mathsf{s}}{\rightarrow}$-normal, by *i.h.* we have:
  - the two sequences $P_1 \underset{\mathsf{s}}{\rightarrow}^* P$ and $P_1 \underset{\mathsf{s}}{\rightarrow}^*_\sigma P_3 \underset{\mathsf{s}}{\rightarrow}^* P$ have the same number of β_c steps, and similarly $Q_1 \underset{\mathsf{s}}{\rightarrow}^* Q$ and $Q_1 \underset{\mathsf{s}}{\rightarrow}^*_\sigma Q_3 \underset{\mathsf{s}}{\rightarrow}^* Q$. Therefore $\mathfrak{s}''_1$ and $\mathfrak{t}$ have the same number of β_c steps.
  - Similarly, $\mathfrak{s}''_2$ and $\mathfrak{t}$ have the same number of β_c steps.

This completes the proof of (#). We now can conclude that $\underset{\mathsf{s}}{\rightarrow}$ is *uniformly normalizing*. If the term M has a sequence $\mathfrak{s} : M \underset{\mathsf{s}}{\rightarrow}^* N$ where N is $\underset{\mathsf{s}}{\rightarrow}$-normal, then no $\underset{\mathsf{s}}{\rightarrow}$-sequence can have more β_c steps than $\mathfrak{s}$, because given any sequence $M \underset{\mathsf{s}}{\rightarrow}^* T$

then (by confluence) $T\underset{\mathsf{s}}{\rightarrow}^* N$, and (by #) $M\underset{\mathsf{s}}{\rightarrow}^* T\underset{\mathsf{s}}{\rightarrow}^* N$ has the same number of β_c steps as $\mathfrak{s}$. Therefore, all $\underset{\mathsf{s}}{\rightarrow}$-sequences from M are finite. □

5.3.2 Normalizing Strategies

We are ready to deal with normalizing strategies for $\lambda_{\copyright}$. Our definition is inspired, and generalizes, the stratified strategy proposed in [Gue15, GPR17], which iterates weak reduction (there called head reduction) according to a more strict discipline.

Iterated e-reduction. We define a family of normalizing strategies, parametrically on the reduction to iterate, which can either be surface or weak reduction. Let $\mathsf{e} \in \{\mathsf{w}, \mathsf{s}\}$. The reduction $\underset{\mathsf{le}}{\rightarrow}$ is defined as follows, by iterating $\underset{\mathsf{e}}{\rightarrow}$-reduction in left to right order.

1. If M has $\underset{\mathsf{e}}{\rightarrow}$-steps:
$$\frac{M\underset{\mathsf{e}}{\rightarrow}M'}{M\underset{\mathsf{le}}{\rightarrow}M'}$$

2. Otherwise, if M is $\underset{\mathsf{e}}{\rightarrow}$-normal:
$$\frac{P\underset{\mathsf{le}}{\rightarrow}P'}{M := [\lambda x.P]\underset{\mathsf{le}}{\rightarrow}[\lambda x.P']} \qquad \frac{P\underset{\mathsf{le}}{\rightarrow}P'}{M := (\lambda x.P)Q\underset{\mathsf{le}}{\rightarrow}(\lambda x.P')Q}$$
$$\frac{V \text{ is } \sigma\beta_c\text{-normal} \quad Q\underset{\mathsf{le}}{\rightarrow}Q'}{M := VQ\underset{\mathsf{le}}{\rightarrow}VQ'}$$

Theorem 5.3.4 (Normalization)**.** *Assume M has a $\rightarrow_{\sigma\beta_c}$-normal form N. Let $\mathsf{e} \in \{\mathsf{w}, \mathsf{s}\}$. Then every maximal $\underset{\mathsf{le}}{\rightarrow}$-sequence from M ends in N.*

Proof. By induction on the term N. We write $\rightarrow$ for $\rightarrow_{\sigma\beta_c}$.

Assume $\mathfrak{s} = M, M_1, M_2, \ldots$ is a maximal $\underset{\mathsf{le}}{\rightarrow}$-sequence from M.

We observe that

(**) every maximal $\underset{\mathsf{e}}{\rightarrow}$-sequence from M is finite.

Indeed, from $M \rightarrow^* N$, by e-factorization, we have that $M\underset{\mathsf{e}}{\rightarrow} U \underset{\neg\mathsf{e}}{\rightarrow}^* N$. Since N is $\underset{\mathsf{e}}{\rightarrow}$-normal, so is U (by Lemma 5.0.3) and (**) follows by uniform normalization of $\underset{\mathsf{e}}{\rightarrow}$ (Theorem 5.3.3).

Let $\mathfrak{s}' \sqsubseteq \mathfrak{s}$ be the maximal prefix of $\mathfrak{s}$ such that $M_i\underset{\mathsf{e}}{\rightarrow}M_{i+1}$. Since it is finite, $\mathfrak{s}'$ is $M, \ldots, M_k$, where M_k is e-normal. Let $\mathfrak{s}'' = M_k, M_{k+1}\ldots$ be the sequence such that $\mathfrak{s} = \mathfrak{s}'\mathfrak{s}''$.

We observe that all terms in $\mathfrak{s}''$ are e-normal (by repeatedly using Lemma 5.0.3 from M_k), so $M_k \underset{\neg\mathsf{e}}{\rightarrow} M_{k+1} \underset{\neg\mathsf{e}}{\rightarrow} \ldots$, and (by shape preservation, Fact 4.3.2) all terms in $\mathfrak{s}''$ have the same shape as M_k.

By confluence of $\rightarrow$ (Proposition 4.1.14), $M_k \rightarrow^* N$. Again, all terms in this sequence are e-normal, by repeatedly using Lemma 5.0.3 from M_k. So, $M_k \underset{\neg\mathsf{e}}{\rightarrow}^* N$, and (by shape preservation, Fact 4.3.2) M_k and N have the same shape.

We have established that M_k and all terms in $\mathfrak{s}'' : M_k, M_{k+1}, \ldots$ have the same shape as N. Now we examine the possible cases for N.

- $N = [x]$, and $M_k = [x]$. Trivially $M \underset{\mathsf{le}}{\to}^* M_k = N$.
- $N = [\lambda x.N_P]$ and $M_k = [\lambda x.P]$ with $P \to^* N_P$. Since N_P is $\sigma\beta_c$-normal, by *i.h.* every maximal $\underset{\mathsf{le}}{\to}$-sequence from P terminates in N_P, and so every maximal $\underset{\mathsf{le}}{\to}$-sequence from $[\lambda x.P]$ terminates in $[\lambda x.N_P] = N$. Since the sequence $\mathfrak{s}'' = M_k, M_{k+1}, \ldots$ is a maximal $\underset{\mathsf{le}}{\to}$-sequence, we have that $\mathfrak{s} = \mathfrak{s}'\mathfrak{s}''$ is as follows
$$M \underset{\mathsf{le}}{\to}^* M_k = [\lambda x.P] \underset{\mathsf{le}}{\to}^* [\lambda x.N_P] = N.$$
- $N = (\lambda x.N_P)N_Q$ and $M_k = (\lambda x.P)Q$, with $P \to^* N_P$ and $Q \to^* N_Q$. Since N_P and N_Q are both $\sigma\beta_c$-normal, by *i.h.*:
  - every maximal $\underset{\mathsf{le}}{\to}$-sequence from P ends in N_P. So every $\underset{\mathsf{le}}{\to}$-sequence from $(\lambda x.P)Q$ eventually reaches $(\lambda x.N_P)Q$;
  - every maximal $\underset{\mathsf{le}}{\to}$-sequence from Q ends in N_Q. So every $\underset{\mathsf{le}}{\to}$-sequence from $(\lambda x.N_P)Q$ eventually reaches $(\lambda x.N_P)N_Q = N$.

  Therefore $\mathfrak{s}$ is as follows
$$M \underset{\mathsf{le}}{\to}^* M_k = (\lambda x.P)Q \underset{\mathsf{le}}{\to}^* (\lambda x.N_P)Q \underset{\mathsf{le}}{\to}^* (\lambda x.N_P)N_Q = N.$$
- $N = xN_Q$ and $M_k = xQ$. Similar to the previous one. □

CHAPTER 6

CONCLUSIONS AND RELATED WORK

On the Computational Core. The computational λ-calculus has been constructed as the theory of a categorical model; similarly in Section 2.1, we base the definition of the $\lambda_{©}$-calculus on strong monads over concrete ccc's possessing a call-by-value reflexive object. As remarked, models having unit and bind as primitive operators are rather different from Moggi's λ_C-models: this is discussed in [Pow00], where in particular the concept of a $\mathcal{C}$-monad seems the appropriate generalization of our functional monad, which is based on a self enriched, concrete ccc.

Since here our focus has been on operational properties and reduction theory, we chose the computational core $\lambda_{©}$ [dT20] among the different variants of computational calculi in the literature inspired by Moggi's seminal work [Mog88, Mog91]. Indeed, the computational core $\lambda_{©}$ has a "minimal" syntax that internalizes Moggi's original idea of deriving a calculus from the categorical model consisting of the Kleisli category of a (strong) monad. For instance, $\lambda_{©}$ does not have to consider both a pure and a (potentially) effectful functional application. So, $\lambda_{©}$ has less syntactic constructors and less reductions rules with respect to other computational calculi, and this simplifies our operational study.

In Section 3.1 we have considered the relationship between $\lambda_{©}$ and λ_{ml^*} from [SW97], proving that there is an equational correspondence provided that the analogous of rule η is added to $\lambda_{©}$. We showed that such a result cannot be strengthened to a Galois connection in the sense of [SW97].

Let us discuss the difference between $\lambda_{©}$ and Moggi's λ_C. As observed in Chapter 1 and Chapter 2, the first formulation of λ_C and of its reduction relation was introduced in [Mog88], where it is formalized through the *let* operator. Indeed, this operator is not just a syntactical sugar for the application of λ-abstraction. In fact, it represents the extension to computations of functions from values to computations, therefore interpreting Kleisli composition. Combining *let* with ordinary abstraction and application is at the origin of the complexity of the reduction rules in [Mog88]. On the other side this allows to internalize extensionality. On the contrary, adding the η-rule to $\lambda_{©}$ breaks confluence.

Beside using *let*, a major difference of $\lambda_{©}$ with respect to λ_C is the neat distinction among the two syntactical sorts of terms, restricting the combination of values and non-values since the very definition of the grammar of the language. In spite of these differences, in [dT19] §9 and here in Appendix C.3, it has been proved that there exists an interpretation of λ_C into $\lambda_{©}$ that preserves the reduction, while there is a reverse translation that preserves convertibility, only.

On the reduction relation. We studied the properties of reduction in the computational core $\lambda_{©}$ focusing on two questions: how to reach values and how to reach normal forms. We faced issues caused by identity and associativity rules, and dealt with them by means of factorization techniques. We assessed the role of associativity as computational and not merely structural, and investigated in depth the structure of normalizing reductions.

We found out that σ-reduction plays at least three distinct, independent roles:

- σ unblocks "premature" β_c-normal forms so as to guarantee that there are not $\lambda_{©}$-normalizable terms whose semantics is the same as diverging terms, as we have seen in Section 5.2;
- it internalizes the associativity of Kleisli composition into the calculus, as a syntactic reduction rule, as explained in Chapter 1 after Equation (1.3);
- it "simulates" the contextual closure of the β_c-rule for terms that reduce to a value, as we have seen in Theorem 4.4.2.

Related Work. Sabry and Wadler [SW97] is the first work on the computational calculus to put on center stage the reduction. Still the focus of the paper are the properties of the translation between that and the monadic metalanguage—the reduction theory itself is not investigated.

In [HZ09] a different refinement of λ_c has been proposed. The reduction rules are divided into a purely operational system, a structural and an observational system. It is proved that the purely operational system suffices to reduce any closed term to a value. This result is similar to Theorem 4.4.5, with weak β_c steps corresponding to head reduction in [HZ09]. Interestingly, the analogous of our rule σ is part of the structural system, while the rule corresponding to our id is generalized and considered as an observational rule. Unlike our work, normalization is not studied in [HZ09].

Surface reduction is a generalization of weak reduction that comes from linear logic. We inherit surface factorization from the linear λ-calculus in [Sim05]. Such a reduction has been recently studied in several variants of the λ-calculus, especially for semantic purposes [AP12, CG14, AG16, EG16, GM19, Gue19].

Regarding the σ-rule, in [CG14] two commutation rules are added to Plotkin's CbV λ-calculus in order to remove meaningless normal forms—the resulting calculus is called *shuffling*. The commutative rule there called σ_3 is literally the same as σ here. In the setting of the *shuffling calculus*, properties such as the fact that all maximal surface $\beta_v\sigma$-reduction sequences from the same term M have the same number of β_v steps, and so such a reduction is uniformly normalizing, were known via semantical tools [CG14, Gue19], namely non-idempotent intersection types. In this part we give the first syntactic proof of such a result.

A relation between the computational calculus, [Sim05] and other linear calculi are well-known in the literature, see for example [EMS09, SW97, MOTW99].

In [dT20], Theorem 8.4 states that any closed term returns a value if and only if it is convergent according to a big-step operational semantics. That proof is incomplete and needs a more complex argument via factorization, as we do here to prove Theorem 4.4.5 (from which that statement in [dT20] and here Lemma 9.1.4 in Part II easily follows).

PART II

CHAPTER 7

INTRODUCTION

The first, main concern of this part is the definition of an intersection type assignment system for the untyped computational λ-calculus $\lambda_{\copyright}$.

Intersection types are an extension of Curry's simple types introduced in the 80's, such that relevant classes of λ-terms are characterized by means of their types: see [BDS13] Part III, and the references there. The motivation for investigating intersection types is that, when including a universal type, usually denoted by ω, that can be assigned to any term, types are invariant under term conversion, instead of just reduction; by this property the term meaning, namely its functional behaviour, is fully characterized by the set of its types. So, having such a system for the computational λ-calculus, opens the way to study by well established mathematical tools the case of (untyped) λ-calculi with effects.

Intersection types are naturally interpreted as predicates over a λ-model, and indeed intersection type systems have been originally conceived as a way to characterize strongly normalizing, weakly normalizing and solvable terms namely having head normal form. In order to develop analogous systems for the computational λ-calculus, we introduce an intersection type assignment system with two sorts of intersection types, namely *value types* ranged over by δ, and *computation types* ranged over by τ, whose intended meanings are subsets of D and TD, respectively. We then define the minimal type theories $\mathit{Th}_{\mathit{Val}}$ and $\mathit{Th}_{\mathit{Com}}$ axiomatizing the preorders over value and computation types, respectively, and construct a type assignment system which is a generalization of the BCD type system for the ordinary λ-calculus in [BCD83]. Then, the subject reduction property smoothly follows, and can be established along the lines of the analogous property of system BCD.

We are looking at BCD type system because it defines a logical semantics of λ-terms, whose meaning are just the sets of types that can be assigned to them, which turn out to be filters of types. Such a model, named *filter model*, has the structure of an algebraic lattice with countable basis. This fact is at the heart of the proof of completeness of the system, namely that the denotation of a term belongs to the interpretation of a type in any model if and only if the type can be assigned to the term in the type system.

However, the type interpretation over a T-model is much more problematic than in case of intersection types and λ-models. The issue consists of ensuring that computation types are closed under the two basic operations of the monad T, that is unit and bind, which we dub *monadic type interpretations*. As we shall see in the technical development, the natural clauses lead to a non inductive definition, hence difficult to handle. To solve the problem we cannot resort to the correspondence of intersection types to compact points in D and TD, because there is no information about the compacts of TD, since the monad T is a parameter.

The solution we propose is to restrict type interpretation to the case of T-models that are (pre-)fixed points of the functor $\mathbf{F}(X) = (X \to TX)$, existing as inverse limit constructions if T and therefore $\mathbf{F}$ is an ω-continuous functor. What one obtains in this way is an instance of Scott's D_∞ model, which is the co-limit of a chain of approximant domains D_n. By interpreting types as admissible subsets of the D_n by induction over n, we obtain admissible subsets of D_∞ and TD_∞ by the very same co-limit construction.

Coming to the filter model construction, we build over the fact that such models can be seen as inverse limit domains, whose structure is determined by the type preorder, that is the type theory one considers: see in particular [DHA03] and [BDS13], Section 16.3. To avoid the rather inelegant shape of domain equations arising from non-extensional filter models, we show here how an extensional T-model can be constructed as a filter model, that is itself a limit model satisfying the domain equation $D = D \to TD$. This eventually leads to the completeness theorem, of which subject expansion is a corollary.

To substantiate our claims, we consider a straightforward adaptation of Abramsky's idea of convergence for the lazy λ-calculus, as the only observable property of terms [Abr90, AO93]. We then relate convergence to reducibility to the trivial computation of a value; eventually we show that convergent terms are exactly those that have a type different than the universal type of computations. We conclude that the filter model is computationally adequate.

Summary. The part is organized as follows: In Chapter 8 we introduce the type assignment system, and in Section 8.2 we establish the subject reduction and expansion theorems. After that we interpret types over $\lambda_{\copyright}$-models. The filter model construction is introduced in Section 8.4 and it is used to prove the soundness and completeness of the type system in Section 8.5.

The convergence predicate is defined in Section 9.1; it is related to the reduction relation and the characterization theorem is established. Finally in section 10 we discuss related work and propose some further developments.

What is presented in this part is a revised version of [dT20] and of the unpublished paper [dT19]. We assume familiarity with λ-calculus, intersection types and domain theory, for which we refer to textbooks such as [AC98] and [BDS13] part III.

CHAPTER 8

INTERSECTION TYPES FOR $\lambda_{©}$

8.1 The Intersection Type Assignment System

Intersection types are an extension of Curry's simple type assignment system to untyped λ-terms, obtained by adding new types $\sigma \wedge \sigma'$ to be assigned to terms that have both type σ and σ'. Intersection type assignment systems form a whole family in the literature; of special interest to us is the system in [BCD83], usually called BCD: see [BDS13] part III. In BCD it is introduced a notion of subtyping together with a universal type ω, that can be assigned to any term, leading to a notion called below *type theory.*

Definition 8.1.1 (Intersection types and Type theories). *A* language of intersection types $\mathcal{T}$ *is a set of expressions* $\sigma, \sigma', \ldots$ *including a constant* ω *and closed under the intersection type constructor:* $\sigma \wedge \sigma'$*.*

An intersection type theory *(shortly a* type theory*) is a pair* $Th = (\mathcal{T}, \leq)$ *where* $\mathcal{T}$ *is a language of intersection types and* $\leq$ *a pre-order over* $\mathcal{T}$ *such that* ω *is the top,* $\wedge$ *is idempotent and commutative, and*

$$\sigma \wedge \sigma' \leq \sigma, \qquad \frac{\sigma \leq \sigma' \quad \sigma \leq \sigma''}{\sigma \leq \sigma' \wedge \sigma''}$$

A type theory is a presentation of a meet-semilattice with ω as top element; in particular $\wedge$ turns out to be monotonic. Different type theories give rise to different structures and hence to different type systems. We adapt BCD type theory to the case of $\lambda_{©}$, where two sorts of types correspond to the two sorts of terms.

Definition 8.1.2 (Intersection types for values and computations). *Let TypeVar be a countable set of* type variables*, ranged over by* α*:*

$$\begin{array}{rrcll} \mathit{ValType}: & \delta & ::= & \alpha \mid \delta \to \tau \mid \delta \wedge \delta \mid \omega_{\mathit{Val}} & \text{(value types)} \\ \mathit{ComType}: & \tau & ::= & T\delta \mid \tau \wedge \tau \mid \omega_{\mathit{Com}} & \text{(computation types)} \end{array}$$

Intersection types from Definition 8.1.2 are better understood as predicates of values and computations, respectively, or as refinement types of the two types of $\lambda_{©}$, that is, using the notation in [MZ15], $\delta \sqsubset D = D \to TD$ in case of values, and $\tau \sqsubset TD$ in case of computations.

In the definition of language *ValType* and, consequently, *ComType*, the set of *TypeVar* (also called *atoms*) is left unspecified and it is a parameter.

Definition 8.1.3 (Type theories Th_{Val} and Th_{Com}). *The intersection type theories $Th_{Val} = (ValType, \leq_{Val})$ and $Th_{Com} = (ComType, \leq_{Com})$ are the least type theories such that:*

$$\delta \leq_{Val} \omega_{Val} \qquad \omega_{Val} \leq_{Val} \omega_{Val} \to \omega_{Com}$$

$$(\delta \to \tau) \wedge (\delta \to \tau') \leq_{Val} \delta \to (\tau \wedge \tau') \qquad \frac{\delta' \leq_{Val} \delta \quad \tau \leq_{Com} \tau'}{\delta \to \tau \leq_{Val} \delta' \to \tau'}$$

$$\tau \leq_{Com} \omega_{Com} \qquad T\delta \wedge T\delta' \leq_{Com} T(\delta \wedge \delta') \qquad \frac{\delta \leq_{Val} \delta'}{T\delta \leq_{Com} T\delta'}$$

Remark 8.1.4. By writing $=_{Val}$ for $\leq_{Val} \cap \leq_{Val}^{-1}$, and similarly $=_{Com}$, we see that all the axioms but $\delta \leq_{Val} \omega_{Val}$ and $\tau \leq_{Com} \omega_{Com}$ are actually equalities. In particular, if $\tau \neq_{Com} \omega_{Com}$ then for a finite set of δ_i we have $\tau =_{Com} \bigwedge_i T\delta_i =_{Com} T(\bigwedge_i \delta_i)$.

Type theories Th_{Val} and Th_{Com} depend on each other. Except for the distinction among value and computation types, theory Th_{Val} is exactly the same as the type theory of BCD. Theory Th_{Com} treats T as a type modality and as a morphism of meet-semilattices: hence it is monotonic and preserves meets. An important feature is that $T\omega_{Val}$ is strictly smaller than ω_{Com} in general; this is consistent with the interpretation of $T\omega_{Val}$ as the largest type of convergent terms. Indeed in Corollary 9.2.4 we shall prove that $\omega_C \leq_{Com} T\omega_{Val}$ is not derivable from the system in Definition 8.1.3 by exhibiting a type interpretation such that this inequality doesn't hold.

Lemma 8.1.5. *If $\tau \in ComType$ is such that $\tau \neq_{Com} \omega_{Com}$ then for some $\delta \in ValType$ we have $\tau =_{Com} T\delta$; hence $\tau \leq_{Com} T\omega_{Val}$.*

Proof. By induction over τ. The only non trivial case is when $\tau \equiv \tau_1 \wedge \tau_2$. From $\tau_1 \wedge \tau_2 \neq_{Com} \omega_{Com}$ it follows that at least one of them is different than ω_{Com}: if say $\tau_1 =_{Com} \omega_{Com}$ then $\tau_1 \wedge \tau_2 =_{Com} \tau_2 \neq_{Com} \omega_{Com}$ so that $\tau_2 =_{Com} T\delta_2$ by induction. Finally, if both τ_1 and τ_2 are not equated to ω_{Com} then by induction $\tau_1 \wedge \tau_2 =_{Com} T\delta_1 \wedge T\delta_2 =_{Com} T(\delta_1 \wedge \delta_2) \leq_{Com} T\omega_{Val}$, for some $\delta_1, \delta_2 \in ValType$. $\square$

We are now in place to introduce the type assignment system for $\lambda_{©}$.

Definition 8.1.6 (Type assignment). *A* basis *is a finite set of typings $\Gamma = \{x_1 : \delta_1, \ldots x_n : \delta_n\}$ with pairwise distinct variables x_i, whose* domain *is the set* $\mathrm{dom}(\Gamma) = \{x_1, \ldots, x_n\}$. *A basis determines a function from variables to types such that $\Gamma(x) = \delta$ if $x : \delta \in \Gamma$, $\Gamma(x) = \omega_{Val}$, otherwise.*

A judgment *is an expression of either shapes:* $\Gamma \vdash V : \delta$ *or* $\Gamma \vdash M : \tau$*. It is* derivable *if it is the conclusion of a derivation according to the rules:*

$$\frac{x : \delta \in \Gamma}{\Gamma \vdash x : \delta}\,(Ax) \qquad \frac{\Gamma, x : \delta \vdash M : \tau}{\Gamma \vdash \lambda x.M : \delta \to \tau}\,(\to I)$$

$$\frac{\Gamma \vdash V : \delta}{\Gamma \vdash [V] : T\delta}\,(\mathit{unit}\, I) \qquad \frac{\Gamma \vdash M : T\delta \quad \Gamma \vdash V : \delta \to \tau}{\Gamma \vdash M \star V : \tau}\,(\to E)$$

where $\Gamma, x : \delta = \Gamma \cup \{x : \delta\}$ *with* $x : \delta \notin \Gamma$*, and the rules:*

$$\frac{}{\Gamma \vdash P : \omega}\,(\omega) \qquad \frac{\Gamma \vdash P : \sigma \quad \Gamma \vdash P : \sigma'}{\Gamma \vdash P : \sigma \wedge \sigma'}\,(\wedge I) \qquad \frac{\Gamma \vdash P : \sigma \quad \sigma \leq \sigma'}{\Gamma \vdash P : \sigma'}\,(\leq)$$

where either $P \in \mathit{Val}$*,* $\omega \equiv \omega_{\mathit{Val}}$*,* $\sigma, \sigma' \in \mathit{ValType}$ *and* $\leq = \leq_{\mathit{Val}}$ *or* $P \in \mathit{Com}$*,* $\omega \equiv \omega_{\mathit{Com}}$*,* $\sigma, \sigma' \in \mathit{ComType}$ *and* $\leq = \leq_{\mathit{Com}}$*.*

In the proof texts we write $\Gamma \vdash V : \delta$ and $\Gamma \vdash M : \tau$ to mean that these judgments are derivable. Because of rule (ω), it is not true in general that if $\Gamma \vdash P : \sigma$ then $\mathsf{fv}(P) \subseteq \mathrm{dom}\,(\Gamma)$; however under the same hypothesis we have that $\Gamma \restriction \mathsf{fv}(P) \vdash P : \sigma$, where, for $X \subseteq \mathit{Var}$, $\Gamma \restriction X = \{x : \delta \mid x : \delta \in \Gamma \;\&\; x \in X\}$ (the restriction of Γ to X).

Among the elementary properties of the system, we state the admissibility of the following rules.

Lemma 8.1.7 (Weakening and Strengthening)**.** *The following rules are admissible:*

$$\frac{\Gamma \vdash P : \sigma \quad \Gamma \subseteq \Gamma'}{\Gamma' \vdash P : \sigma}\,(W) \qquad \frac{\Gamma, x : \delta \vdash P : \sigma \quad \delta' \leq_{\mathit{Val}} \delta}{\Gamma, x : \delta' \vdash P : \sigma}\,(S)$$

where $P \in \mathit{Term}$ *and* σ *is either in ValType or in ComType according to the sort of* P*.*

Proof. Admissibility of rule (W) is proved by a straightforward induction over the derivation of $\Gamma \vdash P : \sigma$. Concerning rule (S) we also reason by induction over the derivation of $\Gamma, x : \delta \vdash P : \sigma$, where x is not bound in P as it is element of $\mathrm{dom}\,(\Gamma, x : \delta)$. It follows that in the derivation the only judgements in which x is the subject of the right hand side typing are either instances of rule $(\leq)$ or instances of (Ax), that is of the shape $\Gamma', x : \delta \vdash x : \delta$ for some $\Gamma' \supseteq \Gamma$. In the first case the thesis follows by induction. In the second case we obtain a new reduction with the same conclusion by replacing (Ax) by the inference

$$\frac{\dfrac{}{\Gamma', x : \delta' \vdash x : \delta'}\,(Ax) \qquad \delta' \leq_{\mathit{Val}} \delta}{\Gamma', x : \delta' \vdash x : \delta}\,(\leq)$$

□

8.2 Subject Reduction and Expansion

In this section we establish the minimal requirement for a sound type system, namely that types are preserved by reductions. Moreover, we prove that types are preserved by subject expansion, which is a characteristic property of intersection type systems with universal type ω.

The next lemma is an extension of the analogous property of BCD type systems, also called Inversion Lemma in [BDS13] 14A.1.

Lemma 8.2.1 (Generation Lemma). *Assume that $\delta \neq \omega_{Val}$ and $\tau \neq \omega_{Com}$, then:*

i) $\Gamma \vdash x : \delta \Leftrightarrow \Gamma(x) \leq_{Val} \delta$

ii) $\Gamma \vdash \lambda x.M : \delta \Leftrightarrow$
$\exists I, \delta_i, \tau_i.\ \forall i \in I.\ \Gamma, x : \delta_i \vdash M : \tau_i\ \ \&\ \ \bigwedge_{i\in I} \delta_i \to \tau_i \leq_{Val} \delta$

iii) $\Gamma \vdash [V] : \tau \Leftrightarrow \exists I, \delta_i\ \forall i \in I.\ \Gamma \vdash V : \delta_i\ \ \&\ \ \bigwedge_{i\in I} T\delta_i \leq_{Com} \tau$

iv) $\Gamma \vdash M \star V : \tau \Leftrightarrow$
$\exists I, \delta_i, \tau_i.\ \forall i \in I.\ \Gamma \vdash M : T\delta_i\ \ \&\ \ \Gamma \vdash V : \delta_i \to \tau_i\ \ \&\ \ \bigwedge_{i\in I} \tau_i \leq_{Com} \tau$

Proof. The implications $\Leftarrow$ are immediate. To prove the implications $\Rightarrow$ we reason by induction over the derivations, by case-distinctions on the last rule. Parts i) and ii) are the same as for ordinary intersection types and λ-calculus; part iii) is immediate by the induction hypothesis, hence we treat part iv) only.

If the last rule in the derivation of $\Gamma \vdash M \star V : \tau$ is ($\to$ E) just take I as a singleton set. If it is ($\leq$) then the thesis follows immediately by induction and the transitivity of $\leq_{Com}$. Finally, suppose that the derivation ends by

$$\frac{\Gamma \vdash M \star V : \tau' \quad \Gamma \vdash M \star V : \tau''}{\Gamma \vdash M \star V : \tau' \wedge \tau''}\ (\wedge I)$$

and $\tau \equiv \tau' \wedge \tau''$. Then by induction we have

$$\exists I, \delta'_i, \tau'_i.\ \forall i \in I.\ \Gamma \vdash M : T\delta'_i\ \ \&\ \ \Gamma \vdash V : \delta'_i \to \tau'_i\ \ \&\ \ \bigwedge_{i\in I} \tau'_i \leq_{Com} \tau'$$

and

$$\exists J, \delta''_j, \tau''_j.\ \forall j \in J.\ \Gamma \vdash M : T\delta''_j\ \ \&\ \ \Gamma \vdash V : \delta''_j \to \tau''_j\ \ \&\ \ \bigwedge_{j\in J} \tau''_j \leq_{Com} \tau''$$

From this the thesis immediately follows by observing that

$$\bigwedge_{i\in I} \tau'_i \leq_{Com} \tau'\ \ \&\ \ \bigwedge_{j\in J} \tau''_j \leq_{Com} \tau'' \Rightarrow \bigwedge_{i\in I} \tau'_i \wedge \bigwedge_{j\in J} \tau''_j \leq_{Com} \tau' \wedge \tau''.$$

$\square$

We observe that the statements of Lemma 8.2.1 could be stronger, since whenever we say that if $\Gamma \vdash P : \sigma$ then $\Gamma' \vdash Q : \sigma'$ it is always the case that the derivation of the latter judgment is a subderivation of the former. Furthermore the inverse of all implications hold.

The following lemma, necessary to the subsequent proofs, establishes a fundamental property of intersection type theories including arrow types, known as *extended applicative type structures*, or EATS [CDHL84]. It has been stated the first time in [BCD83], 2.4 (ii), and it has been widely covered for intersection type theories in [BDS13] with the name β-soundness (Definition 14A.4).

Lemma 8.2.2. *Let $\tau \neq_{Com} \omega_{Com}$, then:*

$$\bigwedge_{i\in I}(\delta_i \to \tau_i) \leq_{Val} \delta \to \tau \Leftrightarrow \exists J \subseteq I.\ J \neq \emptyset \ \ \&\ \ \delta \leq_{Val} \bigwedge_{j\in J} \delta_j \ \ \&\ \ \bigwedge_{j\in J} \tau_j \leq_{Com} \tau$$

Proof. By induction over the definition of $\leq_{Val}$ and $\leq_{Com}$. □

Lemma 8.2.3 (Substitution lemma)**.** *If $\Gamma, x : \delta \vdash M : \tau$ and $\Gamma \vdash V : \delta$ then $\Gamma \vdash M\{V/x\} : \tau$.*

Proof. By induction over the derivation of $\Gamma, x : \delta \vdash M : \tau$. For the induction to go through, one has to show the auxiliary statement that if $\Gamma, x : \delta \vdash W : \delta'$ for some $W \in \mathit{Val}$ then $\Gamma \vdash W\{V/x\} : \delta'$. Details are routine. □

Theorem 8.2.4 (Subject Reduction)**.** *If $\Gamma \vdash M : \tau$ and $M \to N$ then $\Gamma \vdash N : \tau$.*

Proof. Let us assume that $\tau \neq \omega_{Com}$ since the thesis is trivial, otherwise. The proof is by induction over the definition of $M \to N$, using Lemma 8.2.1. We treat just the interesting cases.

Case (β_c): then $M \equiv [V] \star (\lambda x.M')$ and $N \equiv M'\{V/x\}$. From the hypothesis $\Gamma \vdash M : \tau$, by iii) and iv) of Lemma 8.2.1 we have that there exist a finite set I and types δ_i, δ'_i and τ_i for all $i \in I$ such that:

1. $\Gamma \vdash V : \delta'_i$ with $\delta'_i \leq \delta_i$;
2. $\Gamma \vdash \lambda x.M' : \delta_i \to \tau_i$ with $\bigwedge_{i\in I} \tau_i \leq_{Com} \tau$

By ii) of the same lemma for all $i \in I$ there is J_i such that for all $j \in J_i$:

3. $\Gamma, x : \delta_{ij} \vdash M : \tau_{ij}$ with $\bigwedge_{j\in J_i} \delta_{ij} \to \tau_{ij} \leq_{Val} \delta_i \to \tau_i$

In virtue of Lem. 8.2.2, we may assume w.l.o.g that the non empty J_i have been chosen so that $\delta_i \leq_{Val} \bigwedge_{j\in J_i} \delta_{ij}$ and $\bigwedge_{j\in J_i} \tau_{ij} \leq_{Com} \tau_i$ for all $i \in I$.

It follows that $\delta'_i \leq_{Val} \delta_i \leq_{Val} \delta_{ij}$ for all i and j so that by ((a)) we have $\Gamma \vdash V : \delta_{ij}$ by rule ($\leq$). It follows by Lem. 8.2.3 and ((a)) that $\Gamma \vdash M'\{V/x\} : \tau_{ij}$; now $\bigwedge_{i\in I} \bigwedge_{j\in J_i} \tau_{ij} \leq_{Com} \tau$, so that $\Gamma \vdash M'\{V/x\} : \tau$ by repeated applications of rule ($\wedge I$) and ($\leq$).

Case (σ): then $M \equiv (L \star \lambda x.M') \star \lambda y.N'$ and $N \equiv L \star \lambda x.(M' \star \lambda y.N')$ where $x \notin \mathsf{fv}(N')$. As before from $\Gamma \vdash M : \tau$ and Lem. 8.2.1 we know that there exist I, δ_i, τ_i such that for all $i \in I$:

4. $\Gamma \vdash L \star \lambda x.M' : T\delta_i$
5. $\Gamma \vdash \lambda y.N' : \delta_i \to \tau_i$ with $\bigwedge_{i\in I} \tau_i \leq_{\mathit{Com}} \tau$

From (4) it follows that for all $i \in I$ there are $J_i, \delta_{ij}, \tau_{ij}$ such that for all $j \in J_i$:

6. $\Gamma \vdash L : T\delta_{ij}$
7. $\Gamma \vdash \lambda x.M' : \delta_{ij} \to \tau_{ij}$ with $\bigwedge_{j\in J_i} \tau_{ij} \leq_{\mathit{Com}} T\delta_i$

Reasoning as in case (β_c), we obtain from (7) that for all $j \in J_i$ there exist $K_j, \delta_{ijk}, \tau_{ijk}$ s.t.

8. $\Gamma, x : \delta_{ijk} \vdash M' : \tau_{ijk}$ with $\bigwedge_{k\in K_j} \delta_{ijk} \to \tau_{ijk} \leq_{\mathit{Val}} \delta_{ij} \to \tau_{ij}$

Assuming as before that the J_i and the K_j have been suitably chosen, by Lem. 8.2.2 we have that

9. $\delta_i \leq_{\mathit{Val}} \bigwedge_{j\in J_i} \delta_{ij}$ and $\bigwedge_{j\in J_i} \tau_{ij} \leq_{\mathit{Com}} T\delta_i$
10. $\delta_{ij} \leq_{\mathit{Val}} \bigwedge_{k\in K_j} \delta_{ijk}$ and $\bigwedge_{k\in K_j} \tau_{ijk} \leq_{\mathit{Com}} \tau_{ij}$

Therefore, for all i, j, k we have $\delta_{ij} \leq_{\mathit{Val}} \delta_{ijk}$ and $\tau_{ijk} \leq_{\mathit{Com}} T\delta_i$; hence from (8) by (S) and $(\leq)$ we have $\Gamma, x : \delta_{ij} \vdash M' : T\delta_i$. On the other hand by admissibility of rule (W), from (5) and the fact that $x \notin FV(N')$, we have that $\Gamma, x : \delta_{ij} \vdash \lambda y.N' : \delta_i \to \tau_i$. Hence for all i:

$$\dfrac{\Gamma \vdash L : T\delta_{ij} \qquad \dfrac{\dfrac{\Gamma, x : \delta_{ij} \vdash M' : T\delta_i \quad \Gamma, x : \delta_{ij} \vdash \lambda y.N' : \delta_i \to \tau_i}{\Gamma, x : \delta_{ij} \vdash M' \star \lambda y.N' : \tau_i}}{\Gamma \vdash \lambda x.(M' \star \lambda y.N') : \delta_{ij} \to \tau_i}}{\Gamma \vdash L \star \lambda x.(M' \star \lambda y.N') : \tau_i}$$

Now the thesis follows by rules $(\wedge I)$ and $(\leq)$, using $\bigwedge_{i\in I} \tau_i \leq_{\mathit{Com}} \tau$.

Case id is immediate from Lem. 8.2.1; all cases dealing with the compatible closure are easy consequences of the induction hypotheses. □

Toward the proof of subject expansion, we need a lemma that is the inverse of Lem. 8.2.3.

Lemma 8.2.5 (Expansion Lemma). *If $\Gamma \vdash P\{V/x\} : \sigma$ with $V \in \mathit{Val}$, $P \in \mathit{Term}$ and either $\sigma \in \mathit{ValType}$ or $\sigma \in \mathit{ComType}$ according to the sort of P, then then there exists $\delta \in \mathit{ValType}$ such that:*

$$\Gamma \vdash V : \delta \quad \textit{and} \quad \Gamma, x : \delta \vdash P : \sigma$$

Proof. If σ is either ω_{Val} or ω_{Com} then the thesis is trivial. Otherwise, let us assume w.l.o.g. that $x \notin \mathsf{fv}(P\{V/x\}) \cup \mathrm{dom}(\Gamma)$ and that V is free for x in P, that is $\mathsf{fv}(V) \cap BV(P) = \emptyset$, which is a necessary condition for the substitution $P\{V/x\}$ to be capture avoiding. Then we proceed by induction over $P\{V/x\}$, and by cases of P.

Case $P \equiv x$: then σ is some $\delta \in \mathit{ValType}$; since clearly $x\{V/x\} \equiv V$ then $\Gamma \vdash V : \delta$ by the hypothesis, and $\Gamma, x : \delta \vdash x : \delta$ by (Ax), where $\Gamma, x : \delta$ is a basis since $x \notin \mathrm{dom}\,(\Gamma)$.

Case $P \equiv y \not\equiv x$: then we trivially have $y\{V/x\} \equiv y$, so that we obtain the thesis by taking $\delta \equiv \omega_{\mathit{Val}}$.

Case $P \equiv \lambda y.M$: then $P\{V/x\} \equiv \lambda y.(M\{V/x\})$; in particular since free variables in V cannot be caught in $P\{V/x\}$ by the binding λy, we freely assume that $y \notin \mathsf{fv}(V)$. From the hypothesis $\Gamma \vdash \lambda y.(M\{V/x\}) : \sigma$, by ii) of Lem. 8.2.1, it follows that there exist I and δ_i, τ_i such that $\Gamma, y : \delta_i \vdash M\{V/x\} : \tau_i$ for all $i \in I$ and $\bigwedge_{i\in I} \delta_i \to \tau_i \leq_{\mathit{Val}} \sigma$.

By induction for all $i \in I$ there exist δ'_i such that $\Gamma, y : \delta_i \vdash V : \delta'_i$ and $\Gamma, y : \delta_i, x : \delta'_i \vdash M : \tau_i$. Since $y \notin \mathsf{fv}(V)$, from $\Gamma, y : \delta_i \vdash V : \delta'_i$ we obtain $\Gamma \vdash V : \delta'_i$. Taking $\delta = \bigwedge_{i\in I} \delta'_i$ we obtain $\Gamma \vdash V : \delta$ by rule $(\wedge I)$ and $\Gamma, y : \delta_i, x : \delta \vdash M : \tau_i$ by (S). ; on the other hand we get $\Gamma, x : \delta \vdash \lambda y.M : \delta_i \to \tau_i$ for all $i \in I$ by rule $(\to \mathrm{I})$ and $(\wedge I)$. Hence we conclude by rule $(\leq)$.

Case $P \equiv [W]$: then $P\{V/x\} \equiv [W\{V/x\}]$ and the thesis follows by iii) of Lem. 8.2.1 and the induction hypothesis.

Case $P \equiv M \star W$: then $P\{V/x\} \equiv (M\{V/x\}) \star (W\{V/x\})$. By iv) of Lem. 8.2.1 and induction, there exist I and δ_i, τ_i and δ'_1, δ'_2 such that $\Gamma \vdash V : \delta'_j$, $\Gamma, x : \delta'_j \vdash M : T\delta_i$ and $\Gamma, x : \delta'_j \vdash W : \delta_i \to \tau_i$ for all $i \in I$ and $j = 1, 2$, such that $\bigwedge_{i\in I} \tau_i \leq_{\mathit{Com}} \sigma$. Take $\delta = \delta'_1 \wedge \delta'_2$: then $\Gamma \vdash V : \delta$ by $(\wedge I)$ and $\Gamma, x : \delta \vdash M : T\delta_i$ and $\Gamma, x : \delta \vdash W : \delta_i \to \tau_i$ for all $i \in I$ by (S). Now $\Gamma, x : \delta \vdash M \star W : \sigma$ follows by $(\to \mathrm{E})$, $(\wedge I)$ and $(\leq)$.

□

Theorem 8.2.6 (Subject Expansion)**.** *If* $\Gamma \vdash N : \tau$ *and* $M \to N$ *then* $\Gamma \vdash M : \tau$.

Proof. The proof is by induction over $M \to N$, assuming that $\tau \neq_{\mathit{Com}} \omega_{\mathit{Com}}$. The only interesting cases are the following.

Case (β_c): then $M \equiv [V] \star (\lambda x.M')$ and $N \equiv M'\{V/x\}$. By Lem. 8.2.5 there exists δ such that $\Gamma \vdash V : \delta$ and $\Gamma, x : \delta \vdash M' : \tau$. Then $\Gamma \vdash [V] : T\delta$ by rule $(\mathit{unit}\mathrm{I})$ and $\Gamma \vdash \lambda x.M' : \delta \to \tau$ by rule $(\to \mathrm{I})$. We conclude that $\Gamma \vdash [V] \star (\lambda x.M') : \tau$ by rule $(\to \mathrm{E})$.

Case (σ): then $M \equiv (L \star \lambda x.M') \star \lambda y.N'$ and $N \equiv L \star \lambda x.(M' \star \lambda y.N')$.

By iv) of Lem. 8.2.1, from $\Gamma \vdash N : \tau$ there exist I, δ_i, τ_i such that for all $i \in I$ we have $\Gamma \vdash L : T\delta_i$, $\Gamma \vdash \lambda x.(M' \star \lambda y.N') : \delta_i \to \tau_i$ and $\bigwedge_{i\in I} \tau_i \leq_{\mathit{Com}} \tau$. By ii) of Lem. 8.2.1 for all $i \in I$ there exist $J_i, \delta_{ij}, \tau_{ij}$ such that $\Gamma, x : \delta_{ij} \vdash M' \star \lambda y.N' : \tau_{ij}$ and $\bigwedge_{j\in J_i} \delta_{ij} \to \tau_{ij} \leq_{\mathit{Val}} \delta_i \to \tau_i$. From this and by iv) of Lem. 8.2.1, for all $j \in J_i$ there are $K_j, \delta_{ijk}, \tau_{ijk}$ such that for all $k \in K_j$ we have $\Gamma, x : \delta_{ij} \vdash M' : T\delta_{ijk}$ and $\Gamma, x : \delta_{ij} \vdash \lambda y.N' : \delta_{ijk} \to \tau_{ijk}$ and $\bigwedge_{k\in K_j} \tau_{ijk} \leq_{\mathit{Com}} \tau_{ij}$.

Now, by Lem. 8.2.2, from $\bigwedge_{j\in J_i} \delta_{ij} \to \tau_{ij} \leq_{\mathit{Val}} \delta_i \to \tau_i$ we have that there exists $\emptyset \neq J'_i \subseteq J_i$ such that for all $j \in J'_i$, $\delta_i \leq_{\mathit{Val}} \delta_{ij}$ and $\tau_{ij} \leq_{\mathit{Com}} \tau_i$. Hence we obtain that $T\delta_i \leq_{\mathit{Com}} T\delta_{ij}$ so that $\Gamma \vdash L : T\delta_{ij}$, for all i and the appropriate j. On the other hand by rule ($\to$ I) we know that $\Gamma \vdash \lambda x.M' : \delta_{ij} \to T\delta_{ijk}$, so that by rule ($\to$ E) we deduce $\Gamma \vdash L \star \lambda x.M' : T\delta_{ijk}$, for all i and the appropriate j, k.

From $\Gamma, x : \delta_{ij} \vdash \lambda y.N' : \delta_{ijk} \to \tau_{ijk}$ and the fact that $x \notin \mathsf{fv}(\lambda y.N')$ we have that $\Gamma \vdash \lambda y.N' : \delta_{ijk} \to \tau_{ijk}$, that combined with the above, yields $\Gamma \vdash (L \star \lambda x.M') \star \lambda y.N' : \tau_{ijk}$ by rule ($\to$ E). But we know that $\bigwedge_{j\in J'_i, k\in K_j} \tau_{ijk} \leq_{\mathit{Com}} \bigwedge_{j\in J'_i} \tau_{ij} \leq_{\mathit{Com}} \tau_i$, for all $i \in I$: thus the thesis follows by $(\wedge I)$ and $(\leq)$ since $\bigwedge_{i\in I} \tau_i \leq_{\mathit{Com}} \tau$.

□

8.3 Type Interpretation over $\lambda_{©}$-Models

To interpret value and computation types we extend the usual interpretation of intersection types over λ-models to $\lambda_{©}$-models. Let (D, T, Φ, Ψ) be such a model, as defined in Definition 2.3.1; then for $d, d' \in D$ we abbreviate: $d \cdot d' = \Phi(d)(d')$. Also, if $X \subseteq D$ and $Y \subseteq TD$ then $X \Rightarrow Y = \{d \in D \mid \forall d' \in X.\ d \cdot d' \in Y\}$.

Definition 8.3.1. *Let $\xi \in \mathit{TypeEnv}_D = \mathit{TypeVar} \to 2^D$ a type variable interpretation; the maps $[\![\cdot]\!]^D : \mathit{ValType} \times \mathit{TypeEnv}_D \to 2^D$ and $[\![\cdot]\!]^{TD} : \mathit{ComType} \times \mathit{TypeEnv}_D \to 2^{TD}$ are* type interpretations *if:*

$$\begin{array}{rclcrcl}
[\![\alpha]\!]^D_\xi & = & \xi(\alpha) & \qquad & [\![\delta \to \tau]\!]^D_\xi & = & [\![\delta]\!]^D_\xi \Rightarrow [\![\tau]\!]^{TD}_\xi \\
[\![\omega_{\mathit{Val}}]\!]^D_\xi & = & D & & [\![\delta \wedge \delta']\!]^D_\xi & = & [\![\delta]\!]^D_\xi \cap [\![\delta']\!]^D_\xi \\
[\![\omega_{\mathit{Com}}]\!]^{TD}_\xi & = & TD & & [\![\tau \wedge \tau']\!]^{TD}_\xi & = & [\![\tau]\!]^{TD}_\xi \cap [\![\tau']\!]^{TD}_\xi
\end{array}$$

Moreover the following implication holds:

$$d \in [\![\delta]\!]^D_\xi \Rightarrow \mathit{unit}\, d \in [\![T\delta]\!]^{TD}_\xi$$

We say that $[\![\cdot]\!]^{TD}$ is strict *if*

$$[\![T\delta]\!]^{TD}_\xi = \{\mathit{unit}\, d \mid d \in [\![\delta]\!]_\xi\}$$

In the following, if D is a T-model then we assume that $[\![\cdot]\!]^D$ and $[\![\cdot]\!]^{TD}$ are type interpretations satisfying all conditions in Definition 8.3.1 but are not necessarily strict.

Lemma 8.3.2. *Let D, $[\![\cdot]\!]^{TD}$, and $\xi \in \mathit{TypeEnv}_D$ as above.*
The couple (D, ξ) preserves *$\leq_{\mathit{Val}}$ and $\leq_{\mathit{Com}}$, that is: for all $\delta, \delta' \in \mathit{ValType}$ and for all $\tau, \tau' \in \mathit{ComType}$, one has:*

$$\delta \leq_{\mathit{Val}} \delta' \;\Rightarrow\; [\![\delta]\!]^D_\xi \subseteq [\![\delta']\!]^D_\xi \quad \text{and} \quad \tau \leq_{\mathit{Com}} \tau' \;\Rightarrow\; [\![\tau]\!]^{TD}_\xi \subseteq [\![\tau']\!]^{TD}_\xi$$

Clearly any strict type interpretation is a type interpretation. Let us define (strict) truth and (strict) validity of typing judgments.

Definition 8.3.3 (Truth and validity). *We say that $\Gamma \vdash V : \delta$ and $\Gamma \vdash M : \tau$ are* true *in a T-model D w.r.t. the type interpretations $[\![\cdot]\!]^D$ and $[\![\cdot]\!]^{TD}$ if*

1. $\rho, \xi \models^D \Gamma$ *if* $\rho(x) \in [\![\Gamma(x)]\!]^D_\xi$ *for all* $x \in \mathrm{dom}\,(\Gamma)$
2. $\Gamma \models^D V : \delta$ *if* $\rho, \xi \models^D \Gamma$ *implies* $[\![V]\!]^D_\rho \in [\![\delta]\!]^D_\xi$
3. $\Gamma \models^D M : \tau$ *if* $\rho, \xi \models^D \Gamma$ *implies* $[\![M]\!]^{TD}_\rho \in [\![\tau]\!]^{TD}_\xi$

We say that $\Gamma \vdash V : \delta$ and $\Gamma \vdash M : \tau$ are valid*, written $\Gamma \models V : \delta$ and $\Gamma \models M : \tau$, respectively, if $\Gamma \models^D V : \delta$ and $\Gamma \models^D M : \tau$ for any T-model D and type interpretations $[\![\cdot]\!]^D$ and $[\![\cdot]\!]^{TD}$.*

Finally we say that $\Gamma \vdash V : \delta$ and $\Gamma \vdash M : \tau$ are strictly true *in D, written $\Gamma \models^D_s V : \delta$ and $\Gamma \models^D_s M : \tau$, respectively, if the interpretations $[\![\cdot]\!]^D$ and $[\![\cdot]\!]^{TD}$ are strict; also they are* strictly valid*, written $\Gamma \models_s V : \delta$ and $\Gamma \models_s M : \tau$, if strictly true for any D.*

Unfortunately, the type system in Definition 8.1.6 is not sound w.r.t. arbitrary type interpretations. The difficulty comes from rule ($\to$ E), because in general $[\![T\delta]\!]^{TD}_\xi \supseteq \{\mathit{unit}\, d \mid d \in [\![\delta]\!]_\xi\}$ and inclusion might be proper. In case of strict type interpretations, however, this is an equality and the soundness of type assignment is easily established.

Theorem 8.3.4 (Soundness w.r.t. strict interpretations).

$$\Gamma \vdash V : \delta \;\Rightarrow\; \Gamma \models_s V : \delta \quad \textit{and} \quad \Gamma \vdash M : \tau \;\Rightarrow\; \Gamma \models_s M : \tau$$

Proof. By simultaneous induction on the derivations of $\Gamma \vdash V : \delta$ and $\Gamma \vdash M : \tau$. Rules ($\wedge I$) and (ω) are sound by the definition of type interpretation. Rule ($\leq$) is sound by Lemma 8.3.2, as we proved that (D, ξ) preserves $\leq_{Val}$ and $\leq_{Com}$; rule (unitI) is immediate by induction.

Rule ($\to$ I): if $\rho, \xi \models^D_s \Gamma$ and $d \in [\![\delta]\!]^D_\xi$ then $\rho[x \mapsto d], \xi \models^D_s \Gamma, x : \delta$; by induction $\Gamma, x : \delta \models^D_s M : \tau$ which implies

$$[\![\lambda x.M]\!]^D_\rho \cdot d = [\![M]\!]^{TD}_{\rho[x \mapsto d]} \in [\![\tau]\!]^{TD}_\xi$$

Hence $[\![\lambda x.M]\!]^D_\rho \in [\![\delta \to \tau]\!]^D_\xi$ by the arbitrary choice of d.

Rule ($\to$ E): let $\rho, \xi \models^D_s \Gamma$ and assume by induction that $[\![M]\!]^{TD}_\rho \in [\![T\delta]\!]^{TD}_\xi$; because of strictness we have that $[\![M]\!]^{TD}_\rho = \mathit{unit}\, d$ for some $d \in [\![\delta]\!]^D_\xi$, therefore

$$[\![M \star V]\!]^{TD}_\rho = [\![M]\!]^{TD}_\rho \star [\![V]\!]^D_\rho = \mathit{unit}\, d \star [\![V]\!]^D_\rho = [\![V]\!]^D_\rho \cdot d \in [\![\tau]\!]^{TD}_\xi$$

since $[\![V]\!]^D_\rho \in [\![\delta \to \tau]\!]^D_\xi$ by induction.

□

8.3.1 Beyond Strictness

The hypothesis of strictness of type interpretation is quite restrictive (although it suffices for proving computational adequacy: see Section 4.4.3). Consider for example the state monad $\mathbb{S}\,X = [S \to (X \times S)_\perp]$, where S is some domain of states (e.g. $S \subseteq \mathbf{L} \to D$ where $\mathbf{L}$ is some set of locations, and it is ordered by $s \sqsubseteq_S s'$ if $s(\ell) \sqsubseteq_D s'(\ell)$ for all $\ell \in \mathbf{L}$), and according to [Mog91] define:

$$\mathit{unit}_{\mathbb{S}}\, d = \lambda\!\!\lambda\, s \in S.(d, s) \qquad a \star_{\mathbb{S}} f = \lambda\!\!\lambda\, s \in S.\ \mathsf{let}\,(d, s') := a\, s \,\mathsf{in}\, f\, d\, s'$$

Given some $d \in D$ and $\ell \in \mathbf{L}$, let $a = \mathit{unit}_S\, d \in \mathbb{S}D$ and

$$f = \lambda\!\!\lambda\, d\, s.\, (d, \mathit{upd}_\ell(s, d)) \in D \to \mathbb{S}D,$$

where $\mathit{upd}_\ell(s, d)$ is the update of s in ℓ to d. Now $a\, s = (d, s)$ and therefore

$$(a \star_S f)s = f\, d\, s = (d, \mathit{upd}_\ell(s, d))$$

namely $a \star_S f = \lambda\!\!\lambda\, s.(d, \mathit{upd}_\ell(s, d))$ which is a defined computation that is not $\perp_{\mathbb{S}D}$ but different than $\mathit{unit}_S\, d$, therefore it does not belong to the strict type interpretation of any non trivial type $\mathbb{S}\,\delta$.

We will discuss with deep details the case of state monad in Part III.

Definition 8.3.5 (Monadic type interpretation)**.** *Let D be a T-model; then type interpretations $[\![\cdot]\!]^D$ and $[\![\cdot]\!]^{TD}$ are* monadic *if for any $d \in D$ and $a \in TD$*

$$\exists \delta'.\, d \in [\![\delta' \to T\delta]\!]^D_\xi \ \ \&\ \ a \in [\![T\delta']\!]^{TD}_\xi \Rightarrow a \star d \in [\![T\delta]\!]^{TD}_\xi.$$

By the very definition, monadic type interpretations are not inductive; in particular $[\![T\delta]\!]^{TD}_\xi$ depends on itself and also on $[\![T\delta']\!]^{TD}_\xi$ for any δ'. To turn this definition into an inductive one, we make essential use of the correspondence among intersection types and the category of ω-algebraic lattices.

Henceforth $\mathcal{D}$ is the category of ω-algebraic lattices, which is a particular category of domains (see e.g. [AJ94] and [AC98], ch. 5). Objects of $\mathcal{D}$ are lattices whose elements are directed sup of the *compact points* $c \in \mathcal{K}(D) \subseteq D$ they dominate, where c is *compact* if whenever X is directed and $c \sqsubseteq \bigsqcup X$ there is some $x \in X$ s.t. $c \sqsubseteq x$; moreover $\mathcal{K}(D)$ is countable.

Suppose that the monad T is an ω-continuous functor over $\mathcal{D}$, so that the functor $\mathbf{F}(X) = X \to TX$ is such. Let $D_\infty = \lim_{\leftarrow} D_n$ where D_0 is some fixed domain, and $D_{n+1} = \mathbf{F}(D_n) = [D_n \to TD_n]$ is such that for all n, $D_n \lhd D_{n+1}$ is an embedding. As a consequence we have $D_\infty \simeq \mathbf{F}(D_\infty) = [D_\infty \to TD_\infty]$ and we call Φ the isomorphism. Also, by continuity of T, we have that $TD_\infty = \lim_{\leftarrow} TD_n$. If $x \in D_\infty$, we write x_n for its projection to D_n; if $x \in D_n$ then we identify x with its injection into D_∞. We fix the notation for the standard notions from inverse limit construction. For all $n \in \mathbb{N}$ the following are injection-projection pairs:

- $\varepsilon_n : D_n \to D_{n+1}$, $\pi_n : D_{n+1} \to D_n$
- Let $m > n$. $\varepsilon_{n,m} : D_n \to D_m$, $\pi_{m,n} : D_m \to D_n$
- $\varepsilon_{n,\infty} : D_n \to D_\infty$, $\pi_{\infty,n} : D_\infty \to D_n$

For definitions see any standard text on domain theory, e.g. [AC98]. The following lemma lists some well known facts. We set the following abbreviation $x_n = \pi_{\infty,n}(x)$.

Lemma 8.3.6. *Let $x, y \in D_\infty$:*

1. $x = \bigsqcup_n x_n$
2. *if* $x \in D_n$ *then* $x = x_n$
3. $x \cdot y = \bigsqcup_n x_{n+1}(y_n)$
4. *if* $y \in D_n$ *then* $x \cdot y = x_{n+1}(y)$

where $x_n = \pi_{\infty,n}(x)$.

To these we add:

Lemma 8.3.7. *Let $x \in D_\infty$ and $y \in TD_\infty$:*

1. $(\mathit{unit}\, x)_n = \mathit{unit}_{n+1} x$
2. $(y \star x)_n = y_n \star x_n$
3. $a \star d = \bigsqcup_n (a_n \star d_{n+1})$.

If $X \subseteq D_\infty$ we write $X_n = \{x_n \mid x \in X\}$, similarly for Y_n when $Y \subseteq TD_\infty$. By means of this we define a notion of approximated type interpretation such that each type turns out to denote certain well behaved subset either of D_n or TD_n according to its kind, called admissible subset ([AJ94] sec. 2.1.6).

Definition 8.3.8. *A predicate on (i.e. a subset of) a domain D is* admissible *if it contains $\perp$ and is closed under directed suprema. We write as $Adm(D)$ the set of all admissible predicates on a domain D. Fix a domain D and a subset $X \subseteq D$; we define the following operator:*

$$cl_D(X) = \bigcap \{P \in Adm(D) \mid P \supseteq \mathcal{K}(X)\}$$

where $\mathcal{K}(X)$ are the elements of $\mathcal{K}(D)$ bounded above by the elements of X.

Our goal is to show that the type interpretations are admissible subsets of either D_n or TD_n provided that the $\xi(\alpha)$ are such. To enforce admissibility of the $\xi(\alpha)$ we have introduced the cl_D operator.

Lemma 8.3.9. *The operator $cl_D : \mathscr{P}D \to \mathscr{P}D$ is a closure, and $cl_D(X)$ is an object of $\mathcal{D}$ for all $X \subseteq D$, hence it is admissible and algebraic.*

Next we define the notion of approximated type interpretation, that in the limit is a monadic type interpretation (Theorem 8.3.14).

Definition 8.3.10. *Let $\xi \in$ Term-Env$_{D_\infty}$; then we define a family of* approximated type interpretations $[\![\delta]\!]^{D_n}_\xi \subseteq D_n$ *and* $[\![\tau]\!]^{TD_n}_\xi \subseteq TD_n$ *inductively over $n \in \mathbb{N}$, and then over types:*

$$[\![\alpha]\!]^{D_n}_\xi = cl_{D_n}(\xi(\alpha)_n) \qquad [\![\omega_{Val}]\!]^{D_n}_\xi = D_n \qquad [\![\omega_{Com}]\!]^{TD_n}_\xi = TD_n$$

$$[\![\delta \wedge \delta']\!]^{D_n}_\xi = [\![\delta]\!]^{D_n}_\xi \cap [\![\delta']\!]^{D_n}_\xi \qquad [\![\tau \wedge \tau']\!]^{TD_n}_\xi = [\![\tau]\!]^{TD_n}_\xi \cap [\![\tau']\!]^{TD_n}_\xi$$

$$[\![\delta \to \tau]\!]^{D_0}_\xi = D_0 \qquad [\![\delta \to \tau]\!]^{D_{n+1}}_\xi = \{d \in D_{n+1} \mid \forall d' \in [\![\delta]\!]^{D_n}_\xi .\, d(d') \in [\![\tau]\!]^{TD_n}_\xi\}$$

$$\begin{array}{rcl} [\![T\delta]\!]^{TD_0}_\xi & = & TD_0 \\ [\![T\delta]\!]^{TD_{n+1}}_\xi & = & \{unit\, d \in TD_n \mid d \in [\![\delta]\!]^{D_n}_\xi\} \cup \\ & & \{a \star d \in TD_{n+1} \mid \exists \delta' .\, d \in [\![\delta' \to T\delta]\!]^{D_{n+1}}_\xi \ \& \ a \in [\![T\delta']\!]^{TD_n}_\xi\} \end{array}$$

The use of cl_{D_n} in the definition of $[\![\alpha]\!]^{D_n}_\xi$ can be avoided if $\xi(\alpha)_n$ is admissible, for which it suffices that $\xi(\alpha)$ is admissible. We say that ξ is admissible if all $\xi(\alpha)$ are such. Clearly $cl_D(\xi(\alpha)) = \xi(\alpha)$ if ξ is admissible.

Lemma 8.3.11. *Let $\xi \in$ Term-Env$_{D_\infty}$. For all $n \in \mathbb{N}$, every element of the families $[\![\delta]\!]^{D_n}_\xi$ and $[\![\tau]\!]^{TD_n}_\xi$ is an admissible predicate on D_n and TD_n, respectively. Moreover, they are algebraic domains. In addition, every such admissible predicate is an ω-algebraic sublattice of D_n and TD_n, respectively.*

Lemma 8.3.12. *Fix a $\xi \in$ Term-Env$_{D_\infty}$.*
Define $\varepsilon^\delta_n = \varepsilon_n \restriction [\![\delta]\!]^{D_n}_\xi$ and $\pi^\delta_n = \pi_n \restriction [\![\delta]\!]^{D_n}_\xi$, where $\varepsilon_n \restriction [\![\delta]\!]^{D_n}_\xi = \varepsilon_n([\![\delta]\!]^{D_n}_\xi)$, that is the image of $[\![\delta]\!]^{D_n}_\xi$ via ε_n; similarly for $\pi_n \restriction [\![\delta]\!]^{D_n}_\xi$.
Then, for all $\delta \in$ ValType $(\varepsilon^\delta_n, \pi^\delta_n)$ is injection-projection pair.

Define $\varepsilon^{T\delta}_n = T\varepsilon^\delta_n$ and $\pi^{T\delta}_n = T\pi^\delta_n$. By functoriality of T and Lemma 8.3.12 they are an injection-projection pair, too. This implies that the following subsets of D_∞ and TD_∞ do exist.

Definition 8.3.13.

1. $[\![\delta]\!]^{D_\infty}_\xi = \lim_{\leftarrow} [\![\delta]\!]^{D_n}_\xi$

2. $[\![T\delta]\!]^{TD_\infty}_\xi = \lim_{\leftarrow} [\![T\delta]\!]^{TD_n}_\xi$

Theorem 8.3.14. *Let $\xi \in$ Term-Env$_{D_\infty}$ be admissible. Then the mappings $[\![\cdot]\!]^{D_\infty}_\xi$ and $[\![\cdot]\!]^{TD_\infty}_\xi$ are monadic type interpretations; in particular:*

1. $d \in [\![\delta]\!]^{D_\infty}_\xi \Rightarrow unit\, d \in [\![T\delta]\!]^{TD_\infty}_\xi$

2. $\exists \delta' .\, d \in [\![\delta' \to T\delta]\!]^{D_\infty}_\xi \ \& \ a \in [\![T\delta']\!]^{TD_\infty}_\xi \Rightarrow a \star d \in [\![T\delta]\!]^{TD_\infty}_\xi$

Proof. By Definition 8.3.10 $[\![\delta \to \tau]\!]^{D_{n+1}}_\xi = [\![\delta]\!]^{D_n}_\xi \Rightarrow [\![\tau]\!]^{TD_n}_\xi$. Let $d \in [\![\delta \to \tau]\!]^{D_{n+1}}_\xi$ if and only if $d = \bigsqcup_n d_{n+1}$ where $d_{n+1} \in [\![\delta \to \tau]\!]^{D_{n+1}}_\xi$ for all n. Similarly,$d' \in [\![\delta]\!]^{D_\infty}_\xi$ if and only if $d' = \bigsqcup_n d'_{n+1}$ where $d'_{n+1} \in [\![\delta \to \tau]\!]^{D_{n+1}}_\xi$ for all n. Now, by Lemma 8.3.6, $d \cdot d' = \bigsqcup_n d_{n+1}(d'_n)$ and as seen so far $d_{n+1}(d'_n) \in [\![\tau]\!]^{TD_n}_\xi$. Hence,

$\bigsqcup_n d_{n+1}(d'_n) \in \lim_{\leftarrow} [\![\tau]\!]_\xi^{TD_n} = [\![\tau]\!]_\xi^{TD_\infty}$.
Now we prove the *monadicity* of the type interpretation. Let $d \in [\![\delta]\!]_\xi^{D_\infty}$. In particular, $[\![\delta]\!]_\xi^{D_\infty} = \lim_{\leftarrow} [\![\delta]\!]_\xi^{D_n}$, thus $d = \bigsqcup_n d_n$ where $d_n \in [\![\delta]\!]_\xi^{D_n}$. Since $unit\, d_n \in [\![T\delta]\!]_\xi^{TD_n}$ by definition, and since *unit* is continuous, $unit\, d = unit(\bigsqcup_n d_n) = \bigsqcup_n unit\, d_n$. As built so far, $\bigsqcup_n unit\, d_n \in \lim_{\leftarrow} [\![T\delta]\!]_\xi^{TD_n} = [\![T\delta]\!]_\xi^{TD_\infty}$. So clause (i) is satisfied.
Let $a \in [\![T\delta']\!]_\xi^{TD_\infty}$ and $d \in [\![\delta' \to T\delta]\!]_\xi^{D_{n+1}}$. According to (iii) in Lemma 8.3.7, we have $a \star d = \bigsqcup_n (a_n \star d_{n+1})$. By construction $a_n \in [\![T\delta']\!]_\xi^{TD_n}$ and $d_{n+1} \in [\![\delta' \to T\delta]\!]_\xi^{D_{n+1}}$. By definition $a_n \star d_{n+1} \in [\![T\delta]\!]_\xi^{TD_{n+1}}$, thus $a \star d \in \lim_{\leftarrow} [\![T\delta]\!]_\xi^{TD_n} = [\![T\delta]\!]_\xi^{TD_\infty}$. So clause (ii) is satisfied. □

Remark 8.3.15. As long as the monad T is generic and no algebraic operations are considered, the only effects are produced by the trivial computations $[V]$; hence in the term model for the pure core calculus the type interpretation is actually strict. However the closure properties of the monadic interpretations are necessary conditions for type interpretation in the general case, and do actually hold in case of the state monad studied in Part III.

8.4 The Filter-Model Construction

Let $(L, \leq)$ be an inf-semilattice; a non empty $F \subseteq L$ is a *filter* of L if it is upward closed and closed under finite infs; $\mathcal{F}(L)$ is the set of filters of L. The next proposition and theorem are known from the literature, e.g. [DP90, AC98]:

Proposition 8.4.1. *If $(L, \leq)$ is an inf-semilattice then $\mathcal{F}(L) = (\mathcal{F}(L), \subseteq)$ is an algebraic lattice, whose compact elements are the filters $\uparrow a = \{a' \in L \mid a \leq a'\}$. Hence $\mathcal{F}(L)$ is ω-algebraic if L is denumerable.*

Any $D \in |\mathcal{D}|$ arises by ideal completion of $\mathcal{K}(D)$ taken with the restriction of the order $\sqsubseteq$ over D; dually it is isomorphic to the *filter completion* of $\mathcal{K}^{op}(D)$, that is $\mathcal{K}(D)$ ordered by $\sqsubseteq^{op}$, the inverse of $\sqsubseteq$.

Theorem 8.4.2 (Representation theorem)**.** *Let $D \in |\mathcal{D}|$; then $\mathcal{K}^{op}(D)$ is an inf-semilattice and $D \simeq \mathcal{F}(\mathcal{K}^{op}(D))$ is an isomorphism in $\mathcal{D}$.*

Let $Th = (\mathcal{T}, \leq)$ be a type theory. Elements of $\mathcal{F}(\mathcal{T})$ are the non empty subsets $F \subseteq \mathcal{T}$ which are upward closed w.r.t. the preorder $\leq$ and such that if $\sigma, \sigma' \in F$ then $\sigma \wedge \sigma' \in F$. This definition, coming from [BCD83], is not the same as that of $\mathcal{F}(L)$ because $\leq$ is just a preorder, and infs do exist only in the quotient $\mathcal{T}_{/\leq}$, which is indeed an inf-semilattice. The resulting partial order $\mathcal{F}(Th) = (\mathcal{F}(\mathcal{T}), \subseteq)$, however, is isomorphic to $(\mathcal{F}(\mathcal{T}_{/\leq}), \subseteq)$.

Lemma 8.4.3. *Let $\mathcal{T}_{/\leq}$ be the quotient of the pre-order $Th = (\mathcal{T}, \leq)$, whose elements are the equivalence classes $[\sigma] = \{\sigma' \in \mathcal{T} \mid \sigma \leq \sigma' \leq \sigma\}$, ordered by the relation $[\sigma] \lesssim [\sigma'] \Leftrightarrow \sigma \leq \sigma'$. Then $(\mathcal{T}_{/\leq}, \lesssim)$ is an inf-semilattice; moreover $\mathcal{F}(Th) \simeq \mathcal{F}(\mathcal{T}_{/\leq})$ is an isomorphism in $\mathcal{D}$.*

Proof. Proving that $\lesssim$ is well defined and $(\mathcal{T}_{/\leq}, \lesssim)$ is an inf-semilattice is routine. The isomorphism $\mathcal{F}(Th) \simeq \mathcal{F}(\mathcal{T}_{/\leq})$ is given by the map $F \mapsto \{[\sigma] \mid \sigma \in F\}$. □

By Theorem 8.4.2 and Lemma 8.4.3, any $D \in |\mathcal{D}|$ is isomorphic to the filter domain $\mathcal{F}_D = \mathcal{F}(\mathcal{T}_D)$ of some intersection type theory $Th_D = (\mathcal{T}_D, \leq_D)$, called the Lindenbaum algebra of $\mathcal{K}(D)$ in [Abr91].

Definition 8.4.4.
For $D, E \in |\mathcal{D}|$, the functional type theory $Th_{D \to E} = (\mathcal{T}_{D\to E}, \leq_{D\to E})$ *is the least type theory such that $\mathcal{T}_{D\to E}$ includes all expressions of the form $\delta \to \varepsilon$ for $\delta \in \mathcal{T}_D$ and $\varepsilon \in \mathcal{T}_E$ and $\leq_{D\to E}$ is such that*

$$\omega_{D\to E} \leq_{D\to E} \omega_D \to \omega_E \qquad (\delta \to \varepsilon) \wedge (\delta \to \varepsilon') \leq_{D\to E} \delta \to (\varepsilon \wedge \varepsilon')$$

$$\frac{\delta' \leq_{D\to E} \delta \quad \varepsilon \leq_E \varepsilon'}{\delta \to \varepsilon \leq_{D\to E} \delta' \to \varepsilon'}$$

Also $Th_{D\to E}$ is continuous *if*

$$\bigwedge_{i\in I}(\delta_i \to \varepsilon_i) \leq_{D\to E} \delta \to \varepsilon \Rightarrow \bigwedge\{\varepsilon_i \mid i \in I \;\;\&\;\; \delta \leq_D \delta_i\} \leq_E \varepsilon \tag{8.1}$$

Remark 8.4.5. The theory $Th_{D\to E}$ is an *extended abstract type system*, shortly *eats* (see e.g. [AC98] ch. 3), but for the sorts of type expressions. It is continuous if it is a continuous eats.

Proposition 8.4.6. *Let $Th_{D\to E}$ be a continuous functional type theory. Then the domain $\mathcal{F}_{D\to E} = \mathcal{F}(Th_{D\to E})$ is isomorphic to $[\mathcal{F}_D \to \mathcal{F}_E]$, namely the domain of Scott continuous functions from D to E.*

Proof. First if $u \in \mathcal{F}_{D\to E}$ and $d \in \mathcal{F}_D$ then

$$u \cdot d = \{\varepsilon \in \mathcal{T}_E \mid \exists\, \delta \to \varepsilon \in u.\ \delta \in d\} \in \mathcal{F}_E$$

Then the isomorphism $\mathcal{F}_{D\to E} \simeq [\mathcal{F}_D \to \mathcal{F}_E]$ is given by

$$\Phi^{\mathcal{F}}(u) = \lambda\!\!\lambda\, d \in \mathcal{F}_D.\ u \cdot d \qquad \Psi^{\mathcal{F}}(f) = \{\bigwedge_{i\in I}(\delta_i \to \varepsilon_i) \mid \forall i \in I.\ \varepsilon_i \in f(\uparrow\!\delta_i)\} \tag{8.2}$$

To prove that $\Phi^{\mathcal{F}} \circ \Psi^{\mathcal{F}} = \mathrm{id}_{[\mathcal{F}_D \to \mathcal{F}_E]}$ and $\Psi^{\mathcal{F}} \circ \Phi^{\mathcal{F}} = \mathrm{id}_{\mathcal{F}_{D\to E}}$ it is enough to show this for compact elements, which is obtained by simple calculations. $\square$

Definition 8.4.7. *Let $Th = (\mathcal{T}, \leq)$ be a type theory and T a unary symbol; then $Th^T = (\mathcal{T}^T, \leq^T)$ is the least type theory such that $\mathcal{T}^T$ is defined by the grammar*

$$\varphi ::= T\sigma \mid \varphi \wedge \varphi' \mid \omega_T$$

and, for $\sigma, \sigma' \in \mathcal{T}$

$$\varphi \leq^T \omega_T \qquad \sigma \leq \sigma' \Rightarrow T\sigma \leq^T T\sigma' \qquad T\sigma \wedge T\sigma' \leq^T T(\sigma \wedge \sigma')$$

Clearly $Th_{Com} = (Th_{Val})^T$ that we abbreviate by Th^T_{Val}; also, by Theorem 8.4.2, if Th_D is the theory of some $D \in |\mathcal{D}|$ then $\mathcal{F}_{TD} = \mathcal{F}(Th^T_D)$ is again a domain with theory Th^T_D, and $\mathbf{T}D = \mathcal{F}_{TD}$ is a well defined, total map $\mathbf{T} : |\mathcal{D}| \to |\mathcal{D}|$.

In the following we abbreviate $\mathcal{F}_D = \mathcal{F}(Th_D)$ and $\mathcal{F}_{TD} = \mathcal{F}(Th_D^T)$. We also suppose that the set of atoms $TypeVar_0$ is non empty and fixed, and in one-to-one correspondence to the compacts $\mathcal{K}(D_0)$ of some fixed domain D_0; also assume that $\alpha_d \leq_{Val} \alpha_e$ in the theory Th_{Val} if and only if $e \sqsubseteq d$ in D_0.

Finally, we enforce extensionality of the resulting T-model (see below) by adding to Th_{Val} some axioms that equate each atomic type α to an arrow type, for which we have similar choices that is either all (in-)equations $\alpha =_{Val} \omega_{Val} \to T\alpha$ by analogy to Scott's models, or $\alpha =_{Val} \alpha_{Val} \to T\alpha$ by analogy to Park's. No matter which is the actual choice, we ambiguously call Th^η_{Val} the resulting theory and $D_* = \mathcal{F}(Th^\eta_{Val})$ its filter domain.

Lemma 8.4.8. *We have that $Th^\eta_{Val} = Th_{D_* \to \mathbf{T}D_*}$ and it is a continuous functional theory. Therefore $D_* \simeq [D_* \to \mathbf{T}D_*]$.*

Proof. By hypothesis $Th^\eta_{Val} = Th_{D_*}$; by definition $Th_{Com} = (Th^\eta_{Val})^T = Th_{\mathbf{T}D_*}$. By Definition 8.1.3 we have that $Th_{D_*} = Th_{D_* \to \mathbf{T}D_*}$. To see that $Th_{D_* \to \mathbf{T}D_*}$ is a continuous functional type theory it suffices to show that condition (Equation (8.1)) holds in Th_{Val} (similar to the analogous proof for system BCD). Hence

$$D_* \simeq \mathcal{F}(Th_{D_* \to \mathbf{T}D_*}) \simeq [\mathcal{F}(Th_{D_*}) \to \mathcal{F}(Th_{\mathbf{T}D_*})] \simeq [D_* \to \mathbf{T}D_*]$$

by Proposition 8.4.6. □

Lemma 8.4.9. *Let $\mathbf{T} : |\mathcal{D}| \to |\mathcal{D}|$ be as above. Define $unit^{\mathcal{F}}_D : \mathcal{F}_D \to \mathcal{F}_{TD}$ and $\star^{\mathcal{F}}_{D,E} : \mathcal{F}_{TD} \times \mathcal{F}_{D \to \mathbf{T}E} \to \mathcal{F}_{\mathbf{T}E}$ such that:*

$$unit^{\mathcal{F}}_D\, d = \ \uparrow\{T\delta \in \mathcal{T}_{\mathbf{T}D} \mid \delta \in d\} \qquad t \star^{\mathcal{F}}_{D,E} e = \ \uparrow\{\tau \in \mathcal{T}_{\mathbf{T}E} \mid \exists\, \delta \to \tau \in e.\, T\delta \in t\}$$

Then $(\mathbf{T}, unit^{\mathcal{F}}, \star^{\mathcal{F}})$ is a monad over $\mathcal{D}$.

Corollary 8.4.10. *Let $\Phi^{\mathcal{F}}, \Psi^{\mathcal{F}}$ be defined by (Equation (8.2)). Then $(D_*, \mathbf{T}, \Phi^{\mathcal{F}}, \Psi^{\mathcal{F}})$ is a T-model.*

Next we show that D_* is a limit T-model, hence it admits a monadic type interpretation by Theorem 8.3.14. Let's stratify types according to the rank map: $r(\alpha) = r(\omega_{Val}) = r(\omega_{Com}) = 0$, $r(\sigma \wedge \sigma') = \max(r(\sigma), r(\sigma'))$ (for $\sigma \wedge \sigma' \in ValType \cup ComType$), $r(\delta \to \tau) = \max(r(\delta) + 1, r(\tau))$ and $r(T\delta) = r(\delta) + 1$. If $\mathcal{T}$ is any language of intersection types, we set: $\mathcal{T}_n = \{\sigma \in \mathcal{T} \mid r(\sigma) \leq n\}$.

Lemma 8.4.11. *Let $\leq_n = \ \leq \restriction \mathcal{T}_n \times \mathcal{T}_n$ (for both $\leq_{Val}$ and $\leq_{Com}$) and Th_n and $D_n = \mathcal{F}(Th_n)$ be the respective type theories and filter domains. Then the D_n form a denumerable chain of domains such that $D_* = \lim_{\leftarrow} D_n$ is a limit T-model.*

Proof. Recall that $D_n = \mathcal{F}(Th^\eta_n)$. Define $\varepsilon_n : D_n \to D_{n+1}$, $\pi_n : D_{n+1} \to D_n$ as follows:

$$\begin{array}{rcll} \varepsilon_n(d) & = & \{\delta \in ValType_{n+1} \mid \exists \delta' \in d.\, \delta' \leq_{n+1} \delta\} & \text{for } d \in D_n = \mathcal{F}(Th^\eta_n) \\ \pi_n(e) & = & \{\delta \in e \mid r(\delta) \leq n\} & \text{for } e \in D_{n+1} = \mathcal{F}(Th^\eta_{n+1}) \end{array}$$

Let $d \in D_n$ and $e = \varepsilon_n(d)$. Both $\varepsilon_n(d)$ an $\pi_n(e)$ are filters. Clearly $d \subseteq \varepsilon_n(d) \cap ValType_n$, hence $d \subseteq \pi_n(e)$.

Vice versa if $\delta \in \pi_n(e)$ then $r(\delta) \leq n$ and for some $\delta' \in d$, $\delta' \leq_{n+1} \delta$. But $\leq_{n+1}\restriction ValType_n \times ValType_n = \ \leq_n$, hence $\delta' \leq_n \delta$ which implies $\delta \in d$ as the latter is a filter. We conclude that $\pi_n(e) = d$, and therefore $\pi_n \circ \varepsilon_n = \mathrm{id}_{D_n}$ by the arbitrary choice of d.

On the other hand if $d = \pi_n(e)$ for some arbitrary $e \in D_{n+1}$ and $\delta \in \varepsilon_n(d)$ it follows that there exists $\delta' \in e$ such that $r(\delta') \leq n$ and $\delta' \leq_{n+1} \delta$, which implies that $\delta \in e$ being e a filter w.r.t. $\leq_{n+1}$.

In conclusion $\varepsilon_n \circ \pi_n \sqsubseteq \mathrm{id}_{D_{n+1}}$ where $\sqsubseteq$ is the point-wise ordering induced by subset inclusion. Combining with the previous equation we conclude that (ε_n, π_n) is an injection-projection pair, and hence $D_* \subseteq \lim_{\leftarrow} D_n$. To see the inverse inclusion just note that any filter $d \in D_*$ is the union of all its restrictions to the rank n, namely in D_n, and hence is a filter in D_∞. □

Remark 8.4.12. It is not the case that $D_n \subseteq D_{n+1}$: indeed let $\delta \to \tau \in d \in D_n$ and take any δ' such that $r(\delta') = n$; hence $r(\delta \wedge \delta' \to \tau) = n+1$ and $\delta \wedge \delta' \to \tau \notin d$. But $\delta \to \tau \leq_{n+1} \delta \wedge \delta' \to \tau$, therefore the latter type belongs to any filter in D_{n+1} including d. Then we see that ε_n cannot be just set theoretic inclusion. The very same example shows that $\pi_n(e) \subseteq e$ is a proper inclusion in general.

8.5 Soundness and Completeness of the Type System

In the following, let us fix a T-model D and assume that $[\![\cdot]\!]^{TD}$ is a monadic type interpretation. Also we assume $\xi \in TypeEnv_D$ is admissible and $\rho \in Term\text{-}Env_D$.

Lemma 8.5.1. *Let D, $[\![\cdot]\!]^{TD}$, and $\xi \in TypeEnv_D$ as above.*
The couple (D, ξ) preserves *$\leq_{Val}$ and $\leq_{Com}$, that is: for all $\delta, \delta' \in ValType$ and for all $\tau, \tau' \in ComType$, one has:*

$$\delta \leq_{Val} \delta' \ \Rightarrow\ [\![\delta]\!]^D_\xi \subseteq [\![\delta']\!]^D_\xi \quad \textit{and} \quad \tau \leq_{Com} \tau' \ \Rightarrow\ [\![\tau]\!]^{TD}_\xi \subseteq [\![\tau']\!]^{TD}_\xi$$

Theorem 8.5.2 (Soundness)**.**

$$\Gamma \vdash V : \delta \ \Rightarrow\ \Gamma \models V : \delta \quad \textit{and} \quad \Gamma \vdash M : \tau \ \Rightarrow\ \Gamma \models M : \tau$$

Proof. By simultaneous induction on the derivations of $\Gamma \vdash V : \delta$ and $\Gamma \vdash M : \tau$. Rules $(\wedge I)$ and (ω) are sound by a quick inspection on the definition of type interpretation. Rule $(\leq)$ is sound by Lemma 8.5.1, as we proved that (D, ξ) preserves $\leq_{Val}$ and $\leq_{Com}$.

In order to prove soundness of rule $(\to \mathrm{I})$, assume $\Gamma, x : \delta \vdash M : \tau$ and $\rho, \xi \models \Gamma$, one has to show that $[\![\lambda x.M]\!]^D_\rho \in [\![\delta \to \tau]\!]^D_\xi$. Let $d \in [\![\delta]\!]^D_\xi$, the thesis is equivalent to say that $[\![M]\!]^{TD}_{\xi[x \mapsto d]} \in [\![\tau]\!]^{TD}_\xi$, by the arbitrary choice of d, because in any model this is the same as $[\![\lambda x.M]\!]^D_\rho \cdot d \in [\![\tau]\!]^{TD}_\xi$. If we set $F_D = \{e \in D \mid \exists M, \rho.\ e = [\![\lambda x.M]\!]^D_\rho\}$ then the semantic interpretation of rule $(\to \mathrm{I})$ says that $\{e \in F_D \mid \forall d.; d \in [\![\delta]\!]^D_\xi \Rightarrow e \cdot d \in [\![\tau]\!]^{TD}_\xi\} \subseteq [\![\delta \to \tau]\!]^d_\xi$.
For what concerns rule $(unit\,\mathrm{I})$, suppose that the assertion is true for the derivation of $\Gamma \vdash V : \delta$, namely $\Gamma \models V : \delta$, and $\rho, \xi \models \Gamma$. To prove that $\Gamma \models [V]$, one has to

show that $[\![[V]]\!]^{TD}_\rho \in [\![T\delta]\!]^{TD}_\xi$, that is equivalent to $\mathit{unit}[\![V]\!]^D_\rho \in [\![T\delta]\!]^{TD}_\xi$ and this true by (i) of Theorem 8.3.14 Soundness of rule ($\to$ E) follows a very similar path, invoking propriety (ii) of of the same theorem. $\square$

Recall that $D_* = \mathcal{F}(\mathit{Th}^\eta_{\mathit{Val}})$. Let $\xi_0 \in \mathit{TypeEnv}_{D_*}$ be defined by $\xi_0(\alpha) = \{d \in D_* \mid \alpha \in d\} \cup \{\uparrow \omega_{\mathit{Val}}\}$, that is admissible. Also let $[\![\cdot]\!]^{D_*}$ and $[\![\cdot]\!]^{TD_*}$ be the monadic interpretations of Definition 8.3.13.

Lemma 8.5.3.

$$[\![\delta]\!]^{D_*}_{\xi_0} = \{d \in D_* \mid \delta \in d\} \quad \mathit{and} \quad [\![\tau]\!]^{TD_*}_{\xi_0} = \{a \in TD_* \mid \tau \in a\}.$$

Proof. D_* is a T-model by Corollary 8.4.10, and $[\![\cdot]\!]^{D_*}$ and $[\![\cdot]\!]^{TD_*}$ are monadic, hence Lemma 8.5.1 applies. $\square$

Lemma 8.5.4 (Type Semantics Theorem)**.** *For any $\rho \in \mathit{Term\text{-}Env}_{D_*}$:*

1. $[\![V]\!]^{D_*}_\rho = \{\delta \in \mathit{ValType} \mid \exists\Gamma.\ \rho, \xi_0 \models \Gamma \ \ \&\ \ \Gamma \vdash V : \delta\}$
2. $[\![M]\!]^{TD_*}_\rho = \{\tau \in \mathit{ComType} \mid \exists\Gamma.\ \rho, \xi_0 \models \Gamma \ \ \&\ \ \Gamma \vdash M : \tau\}$

Proof. By Theorem 8.5.2 and the fact that D_* is a T-model and type interpretations are monadic, both inclusions $\supseteq$ follow by Lemma 8.5.3. To see inclusions $\subseteq$ we reason by induction over V and M.

Case $V \equiv x$: if $\delta \in [\![x]\!]^{D_*}_\rho = \rho(x)$ take $\Gamma = x : \delta$. Then clearly $\Gamma \vdash x : \delta$. On the other hand $\rho, \xi_0 \models x : \delta$ if $\rho(x) \in [\![\delta]\!]^{D_*}_{\xi_0}$ that is if $\delta \in \rho(x)$ by Lemma 8.5.3, which holds by hypothesis.

Case $V \equiv \lambda x.M$: if $\delta \in [\![\lambda x.M]\!]^{D_*}_\rho$, recall from Definition 2.3.1and Proposition 8.4.6 that:

$$\begin{array}{rcl} [\![\lambda x.M]\!]^{D_*}_\rho & = & \Psi^{\mathcal{F}}(\lambda\!\!\!\lambda\, d.\ [\![M]\!]^{TD_*}_{\rho[x\mapsto d]}) \\ & = & \uparrow\{\bigwedge_{i\in I}(\delta_i \to \tau_i) \mid \forall i \in I.\ \tau_i \in [\![M]\!]^{TD_*}_{\rho[x\mapsto \uparrow\delta_i]}\} \end{array}$$

By induction for all $i \in I$ there exists Γ_i s.t. $\rho[x \mapsto\ \uparrow \delta_i], \xi_0 \models \Gamma_i$ and $\Gamma_i \vdash M : \tau_i$. This implies that $\rho(x) \in\ \uparrow \delta_i$ hence there is no theoretical loss in supposing that $\Gamma_i = \Gamma'_i, x : \delta_i$. Let $\Gamma' = \bigwedge_{i\in I} \Gamma'_i$ be the pointwise intersection of the Γ'_i; it follows that $\Gamma', x : \delta_i \vdash M : \tau_i$ for all $i \in I$. Therefore by ($\to$ I) we have $\Gamma' \vdash \lambda x.M : \delta_i \to \tau_i$ for all $i \in I$, so that by $(\wedge I)$ we conclude that $\Gamma' \vdash \lambda x.M : \bigwedge_{i\in I}(\delta_i \to \tau_i)$, and the thesis follows by $\bigwedge_{i\in I}(\delta_i \to \tau_i) \leq_{\mathit{Val}} \delta$ and rule $(\leq)$.

Case $M \equiv [V]$: if $\tau \in [\![[V]]\!]^{TD_*}_\rho = \mathit{unit}[\![V]\!]^{D_*}_\rho =\ \uparrow\{T\delta \mid \delta \in [\![V]\!]^{D_*}_\rho\}$. By induction there is Γ such that $\rho, \xi_0 \models \Gamma$ and $\Gamma \vdash V : \delta$, from which it follows that $\Gamma \vdash [V] : T\delta$ by (unitI).

Case $M \equiv M' \star V$: if $\tau \in [\![M' \star V]\!]^{TD_*}_\rho$ where:

$$[\![M' \star V]\!]^{TD_*}_\rho = [\![M']\!]^{TD_*}_\rho \star [\![V]\!]^{D_*}_\rho =\ \uparrow\{\tau \mid \exists \delta \to \tau \in [\![V]\!]^{D_*}_\rho.\ T\delta \in [\![M']\!]^{TD_*}_\rho\}$$

By induction there exist Γ' and Γ'' such that:

$$\rho, \xi_0 \models \Gamma' \ \& \ \Gamma' \vdash M' : T\delta \quad \text{and} \quad \rho, \xi_0 \models \Gamma'' \ \& \ \Gamma'' \vdash V : \delta \to \tau$$

Now let $\Gamma = \Gamma' \wedge \Gamma''$; we have that $\rho, \xi_0 \models \Gamma'$ implies that $\Gamma'(x) \in \rho(x)$ for all $x \in \mathrm{dom}\,(\Gamma')$ by Lemma 8.5.3, and similarly $\Gamma''(y) \in \rho(y)$ or all $y \in \mathrm{dom}\,(\Gamma'')$; hence for all $z \in \mathrm{dom}\,(\Gamma) = \mathrm{dom}\,(\Gamma') \cup \mathrm{dom}\,(\Gamma'')$ we have $\Gamma(z) = \Gamma'(z) \wedge \Gamma''(z) \in \rho(z)$ since $\rho(z)$ is a filter. It follows that $\rho, \xi_0 \models \Gamma$ and, since $\Gamma \leq_{\mathit{Val}} \Gamma', \Gamma''$ that both $\Gamma \vdash M' : T\delta$ and $\Gamma \vdash V : \delta \to \tau$, from which we obtain $\Gamma \vdash M' \star V : \tau$ by ($\to$ E).

□

Theorem 8.5.5 (Completeness).

$$\Gamma \models V : \delta \ \Rightarrow \ \Gamma \vdash V : \delta \quad \textit{and} \quad \Gamma \models M : \tau \ \Rightarrow \ \Gamma \vdash M : \tau.$$

Proof. We show the second implication as the first one is similar. Assume that $\Gamma \models M : \tau$, then in particular we have $\Gamma \models^{D_*} M : \tau$. Let $\rho_\Gamma \in \textit{Term-Env}_{D_*}$ be defined by $\rho_\Gamma(x) = \ \uparrow\Gamma(x)$. By construction, we have $\rho_\Gamma, \xi_0 \models \Gamma$ and hence $[\![M]\!]^{TD_*}_{\rho_\Gamma} \in [\![\tau]\!]^{TD_*}_{\xi_0}$. Thus, $\tau \in [\![M]\!]^{TD_*}_{\rho_\Gamma}$ by Lemma 8.5.3. Therefore, there exists Γ' such that $\rho_\Gamma, \xi_0 \models \Gamma'$ and $\Gamma' \vdash M : \tau$ by Lemma 8.5.4.
Without loss of theoretical generality, we can assume $X := \mathrm{dom}\,\Gamma = \mathrm{dom}\,\Gamma' = \mathsf{fv}(M)$. For all $x \in X$, $\Gamma'(x) \in \rho_\Gamma(x) = \ \uparrow\Gamma(x)$ implies that $\Gamma \leq_{\mathit{Val}} \Gamma'$. From this last consideration we conclude that $\Gamma \vdash M : \tau$. □

Subject expansion, already proved in Theorem 8.2.6, can be also obtained as a corollary of the above theorem:

Corollary 8.5.6 (Subject expansion). *If* $\Gamma \vdash M : \tau$ *and* $N \to M$ *then* $\Gamma \vdash N : \tau$.

Proof. Since $M \to N$, for all model D and every ρ, $[\![N]\!]^D_\rho = [\![M]\!]^D_\rho$ by Proposition 2.3.3. In particular, by assuming $D = D_*$, $[\![N]\!]^{TD_*}_{\rho_\Gamma} = [\![M]\!]^{TD_*}_{\rho_\Gamma}$.

$$\begin{array}{lll}
\Gamma \vdash M : \tau & \Rightarrow \ \tau \in [\![M]\!]^{TD_*}_{\rho_\Gamma} & \text{by Theorem 8.5.2 and Lemma 8.5.3} \\
& \Rightarrow \ \tau \in [\![N]\!]^{TD_*}_{\rho_\Gamma} & \\
& \Rightarrow \ \exists\Gamma'.\ \rho_\Gamma, \xi_0 \models \Gamma' \text{ and } \Gamma' \vdash M : \tau & \text{by Lemma 8.5.4} \\
& \Rightarrow \ \Gamma \vdash N : \tau & \text{as in proof of Theorem 8.5.5.}
\end{array}$$

□

CHAPTER 9

CHARACTERIZATION OF CONVERGENCE

In this chapter we introduce a big-step operational semantics that is proved to be equivalent to the reduction relation introduced in Section 2.4. By means of this convergence predicate, we reach the main result of this part, that is a characterization of convergent programs by non trivial typings, Theorem 9.2.1. Methodologically, we obtain this result using the Tait's computability method.

9.1 Equivalence Between Big-Step and Small-Step Operational Semantics

Operational semantics of λ-calculi, as well as of programming languages, consists of giving meaning to terms via a definition of their execution. This can be done either by a small-step reduction relation (more precisely by considering some reduction strategy), or via an evaluation relation of terms to their values, often called *convergence* predicate. Both are examples of structural operational semantics in Plotkin's sense [Plo04], but serve different purposes. Instead of describing the evaluation process in detail, which is also defined on open (sub)-terms, convergence is a relation among "programs", that are closed terms and the closed values to which programs evaluate. Having treated reduction for $\lambda_{©}$ in Part I, we now introduce a convergence predicate, whose definition is inspired by the analogous relation for the call-by-value λ-calculus.

In denotational semantics using domain theoretic models, computational adequacy is the property that divergent programs are precisely those interpreted by $\perp$. This notion is relative to the operational semantics at hand and to the observational notion of convergence. For this to make sense in the present setting, we introduce a notion of convergence inductively, and relate it to the reduction relation, that is the small-step operational semantics that we have introduced and deeply discussed in Part I.

Henceforth in this section, terms are closed if not otherwise stated. Let Com^0 and Val^0 be the set of closed computations and values, respectively.

Definition 9.1.1 (Convergence)**.** *The* convergence relation $\Downarrow \subseteq Com^0 \times \mathbb{N} \times Val^0$, *is defined as follows:*

$$[V] \Downarrow_0 V \qquad\qquad \frac{M \Downarrow_m V' \qquad N\{V'/x\} \Downarrow_n V}{M \star (\lambda x.N) \Downarrow_{m+n+1} V}$$

Notation:

$$\begin{array}{lcl} M \Downarrow V & \Leftrightarrow & \exists n.\, M \Downarrow_n V \\ M \Downarrow & \Leftrightarrow & \exists V.\, M \Downarrow V \end{array}$$

The notation $M \Downarrow_n V$ could have been written as $M \Downarrow_n [V]$, in the sense that a term of the shape $[V]$ is a sort of weak normal form of M w.r.t. the reduction $\rightarrow$ which is defined among computations, not among computation and values. By choosing the definition above we intend to emphasize that $[V]$ is the "trivial" computation of V, that is terminated; in the context of λ_c this is not the same as V, and we want to save the idea that convergence relates programs, namely computations, to their results, that are values. The two concepts are related as stated in the lemma:

Lemma 9.1.2. $M \Downarrow V \Rightarrow M \rightarrow^* [V]$

Proof. By hypotheses $M \Downarrow V$, that is $M \Downarrow_n V$ for some $n \in \mathbb{N}$. Then we reason by induction over n. If $n = 0$ then $M \equiv [V]$ and trivially $[V] \rightarrow^* [V]$.

Otherwise, $M \equiv M' \star \lambda y.N \Downarrow_n V$ and for some $W \in Val^0$:

$$\frac{M' \Downarrow_p W \qquad N\{W/y\} \Downarrow_q V}{M' \star \lambda y.\, N \Downarrow_n V} \quad \text{where } n = p + q + 1$$

By induction $M' \rightarrow^* [W]$ and $N\{W/y\} \rightarrow^* [V]$ so that

$$M' \star \lambda y.\, N \rightarrow^* [W] \star \lambda y.\, N \rightarrow N\{W/y\} \rightarrow^* [V].$$

□

The inverse implication does not hold. Indeed if $N \rightarrow N'$ then $[\lambda x.\, N] \rightarrow [\lambda x.\, N']$, that is $\lambda x.\, N'$ is a more refined value than $\lambda x.\, N$; however, $[\lambda x.\, N] \not\Downarrow \lambda x.\, N'$. We can only prove the weaker statement as in Lemma 9.1.4, which nonetheless suffices. Before doing so, let us establish an useful equivalence lemma between weak β_c-reduction and the big-step predicate. This equivalence can be proved following the sufficient requirements that are established in [Cio13], § 3. In our case, we prefer to prove it directly.

Lemma 9.1.3. $M \underset{\mathsf{w}}{\rightarrow}{}_{\beta_c}^{*} [W] \Rightarrow M \Downarrow W$

Proof. By induction over the length of $M \underset{\mathsf{w}}{\rightarrow}{}_{\beta_c}^{*} [U]$.

If $M \underset{\mathsf{w}}{\rightarrow}{}_{\beta_c}^{0} [W]$, then $M \equiv [W]$, and the thesis is trivial since $[W] \Downarrow_0 W$. Otherwise, suppose that $M \underset{\mathsf{w}}{\rightarrow}_{\beta_c} N \underset{\mathsf{w}}{\rightarrow}_{\beta_c}^{n-1}[W]$. Now we proceed by induction on the contextual closure.

The first case to consider is if $M \underset{\mathsf{w}}{\rightarrow}_{\beta_c} N$ where $\underset{\mathsf{w}}{\rightarrow}_{\beta_c}$ is a root step. Then $M \equiv [V] \star \lambda x.P$ and $N \equiv P\{V/x\}$. We conclude as follows

$$\frac{[V] \Downarrow V \qquad N \equiv P\{V/x\} \Downarrow W \text{ by i.h.}}{M \equiv [V] \star \lambda x.P \Downarrow W}$$

The second case is the case in which the reduction $M \underset{\mathsf{w}}{\rightarrow}_{\beta_c} N$ is not a root step. That is $M \equiv M' \star \lambda x.P$ and there exists N' such that $M' \underset{\mathsf{w}}{\rightarrow}_{\beta_c} N'$ and $N \equiv N' \star \lambda x.P$. By first induction hypothesis $N \equiv N' \star \lambda x.P \Downarrow W$, and it is possible if

$$\frac{N' \Downarrow U' \qquad P\{U'/x\} \Downarrow W}{N' \star \lambda x.P \Downarrow W}$$

Since $M' \underset{\mathsf{w}}{\rightarrow}_{\beta_c} N'$ by second induction hypothesis $M' \Downarrow U'$ and then $M \equiv M' \star \lambda x.P \Downarrow W$ □

Lemma 9.1.4. *$M \rightarrow^* N$ and $N \Downarrow V \;\Rightarrow\; \exists W.\, M \Downarrow W$ and $[W] \rightarrow^* [V]$.*

Proof. If $M \rightarrow^* N$ and $N \Downarrow V$, by Lemma 9.1.2 it means that $M \rightarrow^* N \rightarrow^* [V]$, that is $M \rightarrow^* [V]$. By a very similar argument as in Theorem 4.4.5, there exists a reduction path from M to $[V]$ and a value W such that $M \underset{\mathsf{w}}{\rightarrow}_{\beta_c}^* [W]$ and $[W] \rightarrow^* [V]$. It remains to prove that $M \underset{\mathsf{w}}{\rightarrow}_{\beta_c}^* [W] \Rightarrow M \Downarrow W$ and this follows by Lemma 9.1.3. □

Theorem 9.1.5. *$M \Downarrow \;\Leftrightarrow\; \exists V.\, M \rightarrow^* [V]$*

Proof. Immediate consequence of Lemma 9.1.2 and Lemma 9.1.4. □

9.2 Characterization of Convergence

In view of Theorem 9.1.5, the predicate $M \Downarrow$ is non trivial. Indeed consider the closed term:

$$\Omega_{Com} \equiv [(\lambda x.[x] \star x)] \star (\lambda x.[x] \star x)$$

that is a translation of the well known term $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$ from ordinary λ-calculus. Then the only reduction out of Ω_{Com} is

$$\Omega_{Com} \rightarrow ([x] \star x)\{\lambda x.[x] \star x/x\} \equiv \Omega_{Com} \qquad \text{by } \beta_c$$

which is not of the shape $[V]$ for any $V \in \mathit{Val}$, hence $\Omega_{Com} \not\Downarrow$. The main purpose of this section is to show that typing in our system characterizes convergent terms. We say that $\tau \in \mathit{ComType}$ is *non trivial* if $\tau \neq_{Com} \omega_{Com}$. Then we want to show:

Theorem 9.2.1 (Convergence characterization)**.** *For all $M \in Com^0$ we have:*

$$M \Downarrow \;\Leftrightarrow\; \exists \text{ non trivial } \tau.\ \vdash M : \tau.$$

Towards the proof, and following the pattern of Tait's computability method, we introduce some auxiliary notions.

Definition 9.2.2. *Let $\mathcal{I} : \mathit{TypeVar} \to \mathscr{P}\,\mathit{Val}^0$ be a map; then define $|\delta|_{\mathcal{I}} \subseteq \mathit{Val}^0$ and $|\tau|_{\mathcal{I}} \subseteq \mathit{Com}^0$ by induction as follows:*

i) $|\alpha|_{\mathcal{I}} = \mathcal{I}(\alpha)$

ii) $|\delta \to \tau|_{\mathcal{I}} = \{V \in \mathit{Val}^0 \mid \forall M \in |T\delta|_{\mathcal{I}}.\ M \star V \in |\tau|_{\mathcal{I}}\}$

iii) $|T\delta|_{\mathcal{I}} = \{M \in \mathit{Com}^0 \mid \exists V \in |\delta|_{\mathcal{I}}.\, M \Downarrow V\}$

iv) $|\omega_{\mathit{Val}}|_{\mathcal{I}} = \mathit{Val}^0$ *and* $|\omega_{\mathit{Com}}|_{\mathcal{I}} = \mathit{Com}^0$

v) $|\delta \wedge \delta'|_{\mathcal{I}} = |\delta|_{\mathcal{I}} \cap |\delta'|_{\mathcal{I}}$ *and* $|\tau \wedge \tau'|_{\mathcal{I}} = |\tau|_{\mathcal{I}} \cap |\tau'|_{\mathcal{I}}$.

Lemma 9.2.3. *Let $\mathcal{I}$ be arbitrary. Then:*

i) $\delta \leq_{\mathit{Val}} \delta' \Rightarrow |\delta|_{\mathcal{I}} \subseteq |\delta'|_{\mathcal{I}}$

ii) $\tau \leq_{\mathit{Com}} \tau' \Rightarrow |\tau|_{\mathcal{I}} \subseteq |\tau'|_{\mathcal{I}}$

Proof. By checking axioms and rules in Definition 8.1.3. The only non trivial cases concern the arrow and T-types.

Let $V \in |(\delta \to \tau_1) \wedge (\delta \to \tau_2)|_{\mathcal{I}} = |\delta \to \tau_1|_{\mathcal{I}} \cap |\delta \to \tau_2|_{\mathcal{I}}$, then for all $M \in |T\delta|_{\mathcal{I}}$ we have $M \star V \in |\tau_i|_{\mathcal{I}}$ for both $i = 1, 2$; hence $M \star V \in |\tau_1|_{\mathcal{I}} \cap |\tau_2|_{\mathcal{I}} = |\tau_1 \wedge \tau_2|_{\mathcal{I}}$.

Suppose that $\delta_1 \leq_{\mathit{Val}} \delta_2$ and let $M \in |T\delta_1|_{\mathcal{I}}$; then there exists $V \in |\delta_1|_{\mathcal{I}}$ such that $M \Downarrow V$. By induction $|\delta_1|_{\mathcal{I}} \subseteq |\delta_2|_{\mathcal{I}}$ so that immediately we have $M \in |T\delta_2|_{\mathcal{I}}$.

Let $M \in |T\delta_1 \wedge T\delta_2|_{\mathcal{I}} = |T\delta_1|_{\mathcal{I}} \cap |T\delta_2|_{\mathcal{I}}$. Then there exists $V_1 \in |\delta_1|_{\mathcal{I}}$ and $V_2 \in |\delta_2|_{\mathcal{I}}$ such that $M \Downarrow V_1$ and $M \Downarrow V_2$. By Lemma 9.1.2 we have $M \to^* [V]_i$ for both $i = 1, 2$ and these terms are in normal form; hence $V_1 \equiv V_2$ by Theorem 4.1.13. It follows that there exists a unique $V \in |\delta_1|_{\mathcal{I}} \cap |\delta_2|_{\mathcal{I}} = |\delta_1 \wedge \delta_2|_{\mathcal{I}}$ such that $M \to^* [V]$, hence $M \in |T(\delta_1 \wedge \delta_2)|_{\mathcal{I}}$.

Suppose that $\delta_2 \leq_{\mathit{Val}} \delta_1$ and $\tau_1 \leq_{\mathit{Com}} \tau_2$. Let $V \in |\delta_1 \to \tau_1|_{\mathcal{I}}$ and $M \in |T\delta_2|_{\mathcal{I}}$; by the above $M \in |T\delta_1|_{\mathcal{I}}$ so that $M \star V \in |\tau_1|_{\mathcal{I}}$. By induction $|\tau_1|_{\mathcal{I}} \subseteq |\tau_2|_{\mathcal{I}}$ hence $M \star V \in |\tau_2|_{\mathcal{I}}$ so that $V \in |\delta_2 \to \tau_2|_{\mathcal{I}}$ by the choice of M.

□

Corollary 9.2.4. *A type $\tau \in \mathit{ComType}$ is non trivial if and only if $\tau \leq_{\mathit{Com}} T\omega_{\mathit{Val}}$.*

Proof. By contradiction, if $T\omega_{\mathit{Val}} =_{\mathit{Com}} \omega_{\mathit{Com}}$ then $|T\omega_{\mathit{Val}}|_{\mathcal{I}} = |\omega_{\mathit{Com}}|_{\mathcal{I}}$ for any $\mathcal{I}$ by ii) of Lem. 9.2.3. But $|T\omega_{\mathit{Val}}|_{\mathcal{I}} = \{M \in \mathit{Com}^0 \mid M\Downarrow\} \neq \mathit{Com}^0 = |\omega_{\mathit{Com}}|_{\mathcal{I}}$ since $\Omega_{\mathit{Com}} \not\Downarrow$. The remaining part of the thesis now follows from Lem. 8.1.5. □

We are now in place to show the only if part of Theorem 9.2.1.

Lemma 9.2.5. $M\Downarrow \;\Rightarrow\; \exists\ \tau$ *non trivial* . $\vdash M : \tau$

Proof. If $M \Downarrow$ then $M \to^* [V]$ for some $V \in \mathit{Val}^0$ by Lemma 9.1.2; now $\vdash V : \omega_{\mathit{Val}}$ so that $\vdash [V] : T\omega_{\mathit{Val}}$ by rule (*unit*I); it follows that $\vdash M : T\omega_{\mathit{Val}}$ by Theorem 8.2.6, where $T\omega_{\mathit{Val}}$ is non trivial by Cor. 9.2.4. $\square$

We say that a subset $X \subseteq \mathit{Com}^0$ is *saturated* if for all $M \in \mathit{Com}^0$, $M \to N$ and $N \in X$ imply $M \in X$.

Lemma 9.2.6. *For all $\tau \in \mathit{ComType}$ and $\mathcal{I}$ the set $|\tau|_{\mathcal{I}}$ is saturated.*

Proof. By induction over τ. The case $\tau \equiv \omega_{\mathit{Com}}$ is trivial; the case $\tau \equiv \tau_1 \wedge \tau_2$ is immediate by induction. Let $\tau \equiv T\delta$: then by hypothesis there exists $V \in |\delta|_{\mathcal{I}}$ such that $N \Downarrow V$. By Lem. 9.1.2 we have that $N \to^* [V]$ so that $M \to^* [V]$ and we conclude by Lem. 9.1.4. $\square$

Lemma 9.2.7. *Let $\Gamma \vdash M : \tau$ where $\Gamma = \{x_1 : \delta_1, \ldots, x_k : \delta_k\}$ and $M \in \mathit{Com}$. For any $V_1, \ldots, V_k \in \mathit{Val}^0$ and $\mathcal{I}$, if $V_i \in |\delta_i|_{\mathcal{I}}$ for all $i = 1, \ldots, k$ then $M\{V_1/x_1\} \cdots \{V_k/x_k\} \in |\tau|_{\mathcal{I}}$.*

Proof. We strength the thesis by adding that if $\Gamma \vdash W : \delta$ for $W \in \mathit{Val}$, then $W\{V_1/x_1\} \cdots \{V_k/x_k\} \in |\delta|_{\mathcal{I}}$ under the same hypotheses. Then we reason by simultaneous induction over the derivations of $\Gamma \vdash M : \tau$ and $\Gamma \vdash W : \delta$. The cases of (Ax) and (ω) are straightforward; cases (*unit*I) and $(\wedge I)$ are immediate by induction; case $(\leq)$ follows by induction and Lemma 9.2.3. Let us abbreviate $M\{\vec{V}/\vec{x}\} \equiv M\{V_1/x_1\} \cdots \{V_k/x_k\}$ and similarly for $W\{\vec{V}/\vec{x}\}$.

Case ($\to$ I): then the derivation ends by:

$$\frac{\Gamma, y : \delta' \vdash M' : \tau'}{\Gamma \vdash \lambda y.M' : \delta' \to \tau'} \ (\to \mathrm{I})$$

where $W \equiv \lambda y.M'$ and $\delta \equiv \delta' \to \tau'$. Let $M'' \equiv M'\{\vec{V}/\vec{x}\}$ and assume that $y \notin \vec{x}$; to prove that $(\lambda y.M')\{\vec{V}/\vec{x}\} \equiv \lambda y.M'' \in |\delta' \to \tau'|_{\mathcal{I}}$ we have to show that $N \star \lambda y.M'' \in |\tau'|_{\mathcal{I}}$ for all $N \in |T\delta'|_{\mathcal{I}}$.

Now if $N \in |T\delta'|_{\mathcal{I}}$ then there exists $V' \in |\delta'|_{\mathcal{I}}$ such that $N \Downarrow V'$. This implies that the hypothesis that $V_i \in |\delta_i|_{\mathcal{I}}$ for all $x_i : \delta_i \in \Gamma$ now holds for the larger basis $\Gamma, y : \delta'$ so that by induction we have $M''\{V'/y\} \in |\tau'|_I$. But

$$N \star \lambda y.M'' \to^* ([V]') \star \lambda y.M'' \to M''\{V'/y\}$$

and the thesis follows since $|\tau'|_{\mathcal{I}}$ is saturated by Lemma 9.2.6.

Case ($\to$ E): then the derivation ends by:

$$\frac{\Gamma \vdash M' : T\delta \quad \Gamma \vdash W' : \delta \to \tau}{\Gamma \vdash M' \star W' : \tau}$$

where $M \equiv M' \star W'$. Let $M'' \equiv M'\{\vec{V}/\vec{x}\}$ and $W'' \equiv W'\{\vec{V}/\vec{x}\}$, so that $(M' \star W')\{\vec{V}/\vec{x}\} \equiv M'' \star W''$. By induction $M'' \in |T\delta|_{\mathcal{I}}$ and $W'' \in |\delta \to \tau|_{\mathcal{I}}$ and the thesis follows by definition of the set $|\delta \to \tau|_{\mathcal{I}}$.

□

We can now finish the proof of Theorem 9.2.1.

Proof. By Lem. 9.2.5 it remains to show that if $\vdash M : \tau$ for some non trivial τ then $M\Downarrow$. Since $M \in \mathit{Com}^0$ and the basis is empty, the hypothesis of Lem. 9.2.7 are vacuously true, so that we have $M \in |\tau|_{\mathcal{I}}$ for all $\mathcal{I}$. On the other hand, by Cor. 9.2.4, we know that $\tau \leq_{\mathit{Com}} T\omega_{\mathit{Val}}$ since τ is non trivial. By Lem. 9.2.3 it follows that $M \in |\tau|_{\mathcal{I}} \subseteq |T\omega_{\mathit{Val}}|_{\mathcal{I}} = \{N \in \mathit{Com}^0 \mid N\Downarrow\}$ and we conclude. □

In the next corollary we write $[\![M]\!]^{TD_*}$ for $M \in \mathit{Com}^0$ omitting the term environment ρ which is irrelevant.

Corollary 9.2.8 (Computational Adequacy). *In the model D_* we have that for any $M \in \mathit{Com}^0$:*

$$M\Downarrow \;\;\Leftrightarrow\;\; [\![M]\!]^{TD_*} \neq \bot_{TD_*}$$

Proof. By Lemma 8.5.4, $[\![M]\!]^{TD_*} = \{\tau \in \mathit{ComType} \mid \vdash M : \tau\}$. By Theorem 9.2.1 $M\Downarrow$ if and only if $\vdash M : \tau$ for some non trivial τ; but

$$\bot_{TD_*} = \uparrow\omega_{\mathit{Com}} \subset \uparrow\tau \subseteq [\![M]\!]^{TD_*}$$

where the inclusion $\uparrow\omega_{\mathit{Com}} \subset \uparrow\tau$ is strict since τ is non trivial. □

CHAPTER 10

CONCLUSION AND RELATED WORK

10.1 Related Work

The main inspiration for our intersection type system is [BDS13]. In [dT19] we study the type interpretation over a $\lambda_{©}$-model, which is problematic since it is not inductive. We show there that if the Equation (2.5) is solved in a category of algebraic domains by the inverse limit construction, then such interpretation, which we call monadic, exists, and the type system is sound and complete w.r.t. monadic type interpretations.

Intersection types have been used in [DP00] in a λ-calculus with side effects and reference types. In their work a problem appears, since left distributivity of the arrow over intersection (a rule in [BDS13], that is an axiom of the theory Th_{Val} in Definition 8.1.3 above) is unsound. This is remedied by restricting intersection introduction to values. However, Davies and Pfenning's work is not concerned with monads, so that value and non-value terms and types are of the same sorts. On the contrary, by working in a system like in Definition 8.1.6, these types are distinct: we conjecture that, if actual definitions of unit and bind for the state monad are consistent with their typings, a type system can be constructed that is an instance of ours, such that it is sound without imposing any ad hoc constraint.

The convergence relation in Section 9.1 is the adaptation of a similar concept introduced in [Abr90, AO93] for the lazy λ-calculus, which represents the only observable property in the definition of the applicative bisimulation. It shares some similarity with the convergence relation considered in [DGL17], where Abramsky's idea is extended to a computational λ-calculus very similar to $\lambda_{©}$. However, differently than in Abramsky's work and the relation in Definition 9.1.1, the authors define their predicate as a relation among syntax and semantics, co-inductively defining the interpretation of terms.

Theorem 9.2.1, characterizing convergent terms by non trivial typings, is evidence of the expressive power of our system. However, since convergence is undecidable, non-trivial typability in the system is also undecidable. If the system should be useful in practice, say as a method for abstract interpretation and static analysis or program synthesis, then restricted subsystems should be considered, like bounded intersection type systems recently proposed in [DMRU12, DR17a, DR17b].

10.2 Further Developments

Concerning future developments, we see at least three lines of research. The type system we have presented is about a generic monad T: what about typing calculi with specific monads, like partiality, exceptions, state (developed in Part III), or input-output? The question itself of what means to instantiate the $\lambda_{\copyright}$-calculus and the type assignment system to a particular one, knowing of sufficient conditions guaranteeing that good properties studied here are inherited, is both of theoretical and practical interest.

The $\lambda_{\copyright}$-calculus is pure, namely without constants. To formalize data types we need algebraic terms and suitable typing rules; in the framework of intersection type systems, types provide a logical semantics to terms, which is the consequence of type invariance under conversion, and, at the same time, can be seen as a denotational semantics in the category of algebraic domains: see the filter model construction in [BDS13] and Abramsky's domain logic theory [Abr91]. A natural question is then what kind of algebraic and co-algebraic specifications and principles are sound in the logical semantics, when induced by an intersection type system with monads.

The computational λ-calculus has been proposed as a foundation for the static analysis of effectful calculi and programming languages. In [WT03], extended version of the previously published [Wad98], this is compared to Lucassen and Gifford [LG88] and Talpin and Jouvelot [TJ94] type and effect discipline (see [NNH99] chap. 5 for an exposition, and the relation to other static analysis techniques). The same topic has been treated by Benton and others in [BHM02] and [BKHB06]. While it is known that intersection types and abstract interpretation are related, see [Jen95] and [CF93], we do not know of any research work relating intersection types to effect systems. Now that we have introduced intersection types for the computational λ-calculi, we have the right theoretical framework to investigate this topic.

10.3 Conclusion

Starting with the general domain equation of the type-free call-by-value computational λ-calculus we have tailored term and type syntax and defined a type theory and an intersection type assignment system that is sound and complete w.r.t. the interpretation of terms and types in a class of models. This class is parametrically defined w.r.t. the monad at hand, as well as the system itself that induce a family of filter models. If the monad is the trivial one, originating from the identity functor, then our system collapses to BCD. The filter model we have obtained is for a generic monad; we conjecture that the model is initial in a suitable category, to which all its instances belong. We leave this question to further research. In the end we have proved that our Curry style, simple type assignment system, is expressive enough to characterize convergent terms by their typings.

PART III

CHAPTER 11

INTRODUCTION TO THE CASE OF STORE MONAD

The problem of integrating non functional aspects into functional programming languages goes back to the early days of functional programming; nowadays, even procedural and object-oriented languages embody more and more features from declarative languages, renewing interest and motivations in the investigation of higher-order effectful computations.

Since Strachey and Scott's work in the 60's, λ-calculus and denotational semantics, together with logic and type theory, have been recognized as the mathematical foundations of programming languages. Nonetheless, there are aspects of actual programming languages that have shown to be quite hard to treat, at least with the same elegance as the theory of recursive functions and of algebraic data structures; a prominent case is surely side-effects.

Focusing on side-effects, the method used in the early studies to treat the store, e.g. [FF89, Tof90, WF94] and [SRI91], is essentially additive: update and dereferentiation primitives are added to a (often typed) λ-calculus, possibly with constructs to dynamically create and initialize new locations. Then, a posteriori, tools to reason about such calculi are built either by extending the type discipline or by means of denotational and operational semantics, or both.

The introduction of notions of computation as monads and of the computational λ-calculus by Moggi in [Mog91] greatly improved the understanding of "impure", namely non functional features in the semantics of programming languages, by providing a unified framework for treating computational effects: see [Wad92, Wad95], the introductory [BHM02], and the large body of bibliography thereafter; a gentle introduction and further references can be found e.g. in [Gav19]. The key idea is to model effectful computations as morphisms of the Kleisli category of a monad in [Mog91], starting a very rich thread in the investigation of programming language foundations based on categorical semantics, which is still flourishing. The methodological advantage is that we have a uniform and abstract way of speaking of various kinds of effects, and the definition of equational logics to reason about them. At the same time computational effects, including side-effects, can be safely embodied into purely functional languages without disrupting their declarative nature, as shown by Wadler's [Wad92, Wad95] together with a long series of papers by the same author and others.

Monads alone model how morphisms from values to computations compose, but do not tell anything about how the computational effects are produced. In the theory of *algebraic effects* [PP02, PP03, Pow06], Plotkin and Power have shown under which conditions effect operators live in the category of algebras of a computational monad, which is equivalent to the category of models of certain equational specifications, namely varieties in the sense of universal algebra [HP07].

Initiating with Parts I and II, we have approached the issue of modelling effects in an untyped computational λ-calculus, studying its operational semantics by introducing a reduction relation and a syntactical convergence predicate that is characterized in terms of an intersection type assignment system.

Our approach was limited to a calculus without constants, where the unit and bind operations are axiomatized by the monadic laws from [Wad95].

In the present part, we add to the calculus syntax denumerably many operations get_ℓ and set_ℓ, indexed over an infinite set of locations, to access a global store, and define an operational semantics in SOS style, which turns out to be the small-step correspondent to the big-step operational semantics proposed in [Gav19], chap. 3. From there, we call the calculus λ_{imp}.

Reasoning about such a calculus is challenging. One possible approach has been investigated in [DGL17], generalizing Abramsky's functional bisimulation to computational λ-calculi. In the present work, we advocate a more abstract method based on the denotational semantics and domain theory.

Following [Mog88], λ_{imp} can be modelled into a domain D satisfying the equation $D = D \to \mathbb{S}D$, clearly reminiscent of Scott's $D = D \to D$ reflexive object, where $\mathbb{S}$ is instantiated to a variant of the state monad in [Mog91], called the partiality and state monad in [DGL17].

The method we follow is to solve such equation in the category of ω-algebraic lattices, whose objects and morphisms can be described via intersection type theories, that are the "logic" of such domains in the sense of [Abr91]: see also [ADCH04]. Intersection types denote compact points in such a domain, therefore we can recover from the domain theoretic definition of the monad all the information needed to build a sound and complete type assignment system.

Exploiting the above approach, subtyping and typing rules are derived in a uniform way, leading to a type assignment system which enjoys the "type-semantics" property, namely that denotational semantics of terms is fully characterized by their typings in the system. Then, the results we obtain are type invariance under reduction and expansion, and the characterization by a single type of *convergent computations*, where we say that a term converges if it evaluates to a value and a final state, whatever the initial state is.

Summary and results. In Section 12.1 we outline the syntax of the untyped imperative λ-calculus λ_{imp} and in Section 12.2 we illustrate its reduction relation. The convergence predicate and its relation w.r.t. evaluation is shown in Section 12.3. In Section 13.1 we deal with solving the domain equation thoroughly. To do this we consider the state and partiality monad $\mathbb{S}$. In the same section, we tackle the solution of the domain equation by breaking the circularity that arises out of the definition of the monad $\mathbb{S}$ itself. We conclude the section by modelling algebraic operators over the monad $\mathbb{S}$ and formalizing the model of λ_{imp}. In Section 13.2, we solve the domain equation by inductively constructing type theories that induce a filter-model.

Section 13.3 explains how to derive the type assignment system from the filter-model construction. We conclude establishing the type semantics theorem for our calculus.

In Section 13.4 we present how to type configurations and illustrate the type system w.r.t. the operational semantics. Its type invariance is proved in Section 14.1. Then, in Section 14.2 we treat the second main result of this part, namely the characterization of the convergence predicate by a single type.

Finally, Section 15.1 is devoted to the discussion of our results and to related works.

We assume familiarity with λ-calculus and intersection types; a comprehensive reference is [BDS13], Part III. Notions from domain theory, computational monads, and algebraic effects, are shorty recalled in the following; for further references see e.g. [AC98], [BHM02], and [Gav19] chap. 3.

What follows is a revised and extended version of [dT21b] and [dT21a].

CHAPTER 12

AN IMPERATIVE λ-CALCULUS

12.1 An Untyped Imperative λ-calculus

Imperative extensions of the λ-calculus, both typed and type-free, are usually based on the call-by-value λ-calculus, enriched with constructs for reading and writing to the store. Aiming at exploring the semantics of side effects in computational calculi, where "impure" functions are modeled by pure ones sending values to computations in the sense of [Mog91], we consider the computational core $\lambda_{©}$, to which we add syntax denoting algebraic effect operations à la Plotkin and Power [PP02, PP03, Pow06] over a suitable state monad.

Let $\mathbf{L} = \{\ell_0, \ell_1, \ldots\}$ be a denumerable set of abstract *locations*. Borrowing notation from [Gav19], chap. 3, we consider denumerably many operator symbols get_ℓ and set_ℓ, obtaining:

Definition 12.1.1 (Term syntax)**.**

$$\begin{array}{rrcl} \mathit{Val}: & V, W & ::= & x \mid \lambda x.M \\ \mathit{Com}: & M, N & ::= & [V] \mid M \star V \\ & & \mid & get_\ell(\lambda x.M) \mid set_\ell(V, M) \quad (\ell \in \boldsymbol{L}) \end{array}$$

As for $\lambda_{©}$, terms are of sorts either *Val* or *Com*, representing values and computations, respectively. The new constructs are $get_\ell(\lambda x.M)$ and $set_\ell(V, M)$. The variable x is bound in $\lambda x.M$ and $get_\ell(\lambda x.M)$; terms are identified up to renaming of bound variables so that the capture avoiding substitution $M\{V/x\}$ is always well defined; $FV(M)$ denotes the set of free variables in M. We call Val^0 the subset of closed $V \in \mathit{Val}$; similarly for Com^0.

With respect to the syntax of the "imperative λ-calculus" in [Gav19], we do not have the *let* construct, nor the application VW among values. The justification is the same as for the computational core. These constructs are definable:

$$\mathsf{let}\, x := M\, \mathsf{in}\, N \;\equiv\; M \star (\lambda x.N) \qquad VW \;\equiv\; [W] \star V$$

where $\equiv$ is syntactic identity. In general, application among computations can be encoded by $MN \equiv M \star (\lambda z.\ N \star z)$, where z is fresh.

In a sugared notation from functional programming languages, we could have written:

$$\mathsf{let}\, x := !\ell \,\mathsf{in}\, M \;\equiv\; \mathit{get}_\ell(\lambda x.M) \qquad \ell := V; M \;\equiv\; \mathit{set}_\ell(V, M)$$

representing location dereferentiation and assignment. Observe that we do not consider locations as values; consequently they cannot be dynamically created like with the **ref** operator from ML, nor is it possible to model aliasing. On the other hand, since the calculus is untyped, "strong updates" are allowed. In fact, when evaluating $\mathit{set}_\ell(V, M)$, the value V to which the location ℓ is updated bears no relation to the previous value, say W, of ℓ in the store: indeed, in our type assignment system W and V may well have completely different types. This will be, of course, a major challenge when designing the type assignment system in the next sections.

12.2 Operational Semantics

We define the operational semantics of our calculus via a reduction relation $(M, s) \to (N, t)$, where $M, N \in \mathit{Com}$ and s, t are *store* terms, which are defined below.

Definition 12.2.1 (Store and Lookup terms)**.** *Let V vary over Val; then define:*

$$\begin{array}{rcll} \mathit{Store} \ni s & ::= & \mathit{emp} \mid \mathit{upd}_\ell(u, s) & \\ \mathit{Lkp} \ni u & ::= & V \mid \mathit{lkp}_\ell(s) & \qquad \ell \in \mathit{dom}(s) \end{array}$$

$$\begin{array}{rcl} \mathit{dom}(\mathit{emp}) & = & \emptyset \\ \mathit{dom}(\mathit{upd}_\ell(u, s)) & = & \{\ell\} \cup \mathit{dom}(s) \end{array}$$

Store terms represent finite mappings from **L** to *Val*. *emp* is the *empty store*, that is the everywhere undefined map; $\mathit{upd}_\ell(V, s)$ is the *update* of s, representing the same map as s, but for ℓ where it holds V.

To compute the value of the store s at location ℓ we add the expressions $\mathit{lkp}_\ell(s) \in \mathit{Lkp}$, whose intended meaning is the *lookup* (partial) function searching the value of ℓ in the store s; more precisely $\mathit{lkp}_\ell(s)$ picks the value V from the leftmost outermost occurrence in s of a subterm of the shape $\mathit{upd}_\ell(V, s')$, if any.

To avoid dealing with undefined expressions like $\mathit{lkp}_\ell(\mathit{emp})$, we have asked that $\ell \in \mathit{dom}(s)$ for the lookup expression $\mathit{lkp}_\ell(s)$ to be well formed. Then the meaning of store and lookup terms can be defined axiomatically as follows.

Definition 12.2.2 (Store and Lookup Axioms)**.** *The axioms of the algebra of store terms and well formed lookup expressions are the following:*

1. $\mathit{lkp}_\ell(\mathit{upd}_\ell(u, s)) = u$

2. $\mathit{lkp}_\ell(\mathit{upd}_{\ell'}(u, s)) = \mathit{lkp}_\ell(s)$ *if* $\ell \neq \ell'$

3. $\mathit{upd}_\ell(\mathit{lkp}_\ell(s), s) = s$

4. $upd_\ell(U, upd_\ell(W, s)) = upd_\ell(U, s)$

5. $upd_\ell(U, upd_{\ell'}(W, s)) = upd_{\ell'}(W, upd_\ell(U, s))$ *if* $\ell \neq \ell'$

We write $\vdash s = t$ to stress that the equality $s = t$ has been derived from the above axioms using reflexivity, symmetry, transitivity, and congruence of equational logic.

The equalities in Definition 12.2.2 are folklore for terms representing the store in the literature: see e.g. [Mit96], chap. 6; also they are essentially, albeit not literally, the same as those ones for global state in [PP02]. This is a non trivial and decidable theory; since we have not found any good reference to establish the properties we shall use in the subsequent sections, we devote the next paragraphs to the study of this theory. The main results are Theorem 12.2.10 and Corollary 12.2.11.

Lemma 12.2.3. $\ell \in dom(s) \Rightarrow \exists V \in Val.\ lkp_\ell(s) = V$

Proof. By induction over s; since $dom(s) \neq \emptyset$ we have that $s \not\equiv emp$ and the lookup expression $lkp_\ell(s)$ is wellformed, so that there are two cases to consider:

$s \equiv upd_\ell(u, s')$: then by axiom 12.2.2.1 we have $lkp_\ell(upd_\ell(u, s')) = u$. If $u \equiv V \in Val$ then we are done; otherwise, $u \equiv lkp_\ell(s'')$, and the size of s'' is strictly smaller than that of s, so that the thesis follows by induction.

$s \equiv upd_{\ell'}(u, s')$, with $\ell' \neq \ell$: then $lkp_\ell(upd_{\ell'}(u, s')) = lkp_\ell(s')$ by axiom 12.2.2.2, and the thesis follows by induction.

□

As expected, the relation $\vdash lkp_\ell(s) = V$ is functional:

Lemma 12.2.4. *If* $\vdash lkp_\ell(s) = V$ *and* $\vdash lkp_\ell(s) = W$ *then* $V \equiv W$.

Proof. By induction over the derivations of $lkp_\ell(s) = V$. □

Definition 12.2.5. *We say that* $s, t \in Store$ *are* extensionally equivalent*, written* $s \simeq t$*, if there exists* $L \subseteq \boldsymbol{L}$ *such that:*

1. $dom(s) = L = dom(t)$

2. $\forall \ell \in L.\ lkp_\ell(s) = lkp_\ell(t)$

Lemma 12.2.6. $\vdash s = t \Rightarrow s \simeq t$

Proof. The thesis holds immediately of the axioms 12.2.2.3-12.2.2.4; then the proof is a straightforward induction over the derivation of $s = t$. □

Toward establishing the converse of Lemma 12.2.6 let us define $s \setminus \ell \in Store$ by:

$$\begin{array}{rcl} emp \setminus \ell & \equiv & emp \\ upd_\ell(u, s) \setminus \ell & \equiv & s \setminus \ell \\ upd_{\ell'}(u, s) \setminus \ell & \equiv & upd_{\ell'}(u, s \setminus \ell) \quad \text{if } \ell' \neq \ell \end{array}$$

Lemma 12.2.7.

1. $s \setminus \ell \equiv s \Leftrightarrow \ell \notin \mathit{dom}(s)$
2. $\mathit{dom}(s \setminus \ell) = \mathit{dom}(s) \setminus \{\ell\}$
3. $s \simeq t \Rightarrow (s \setminus \ell) \simeq (t \setminus \ell)$

Proof. Parts (1) and (2) are immediate consequences of the definitions; part (3) follows from (1) and (2). □

Lemma 12.2.8. $\ell \in \mathit{dom}(s) \Rightarrow \vdash s = \mathit{upd}_\ell(\mathit{lkp}_\ell(s), s \setminus \ell)$

Proof. By induction over s. By the hypothesis $\ell \in \mathit{dom}(s)$ we have that $\mathit{lkp}_\ell(s)$ is well-formed, and $s \not\equiv \mathit{emp}$; so that we have the cases:

$s \equiv \mathit{upd}_\ell(V, s')$: then we have $\vdash \mathit{lkp}_\ell(s) = V$ by Definition 12.2.2.1 so that

$$\vdash \mathit{upd}_\ell(V, s') = \mathit{upd}_\ell(\mathit{lkp}_\ell(s), s')$$

Now if $\ell \notin \mathit{dom}(s')$ we have

$$\begin{array}{rcll} \mathit{upd}_\ell(\mathit{lkp}_\ell(s), s') & \equiv & \mathit{upd}_\ell(\mathit{lkp}_\ell(s), s' \setminus \ell) & \text{by Lemma 12.2.7.1} \\ & \equiv & \mathit{upd}_\ell(\mathit{lkp}_\ell(s), s \setminus \ell) & \text{by } s \setminus \ell \equiv \mathit{upd}_\ell(V, s') \setminus \ell \equiv s' \setminus \ell \end{array}$$

If instead $\ell \in \mathit{dom}(s')$ then $\mathit{lkp}_\ell(s')$ is well formed and

$$\begin{array}{rcll} \mathit{upd}_\ell(\mathit{lkp}_\ell(s), s') & = & \mathit{upd}_\ell(\mathit{lkp}_\ell(s), \mathit{upd}_\ell(\mathit{lkp}_\ell(s'), s' \setminus \ell)) & \text{by induction} \\ & = & \mathit{upd}_\ell(\mathit{lkp}_\ell(s), s' \setminus \ell) & \text{by axiom 12.2.2.4} \end{array}$$

$s \equiv \mathit{upd}_{\ell'}(V, s')$ and $\ell' \neq \ell$: in this case $\ell \in \mathit{dom}(s) = \mathit{dom}(\mathit{upd}_{\ell'}(V, s'))$ implies $\ell \in \mathit{dom}(s')$, so that

$$\begin{array}{rcll} \mathit{upd}_{\ell'}(V, s') & = & \mathit{upd}_{\ell'}(V, \mathit{upd}_\ell(\mathit{lkp}_\ell(s'), s' \setminus \ell)) & \text{by induction} \\ & = & \mathit{upd}_\ell(\mathit{lkp}_\ell(s'), \mathit{upd}_{\ell'}(V, s' \setminus \ell)) & \text{by axiom 12.2.2.5} \\ & \equiv & \mathit{upd}_\ell(\mathit{lkp}_\ell(s'), \mathit{upd}_{\ell'}(V, s') \setminus \ell) & \text{by definition of } \mathit{upd}_{\ell'}(V, s') \setminus \ell \\ & = & \mathit{upd}_\ell(\mathit{lkp}_\ell(s), \mathit{upd}_{\ell'}(V, s') \setminus \ell) & \text{as } \vdash \mathit{lkp}_\ell(\mathit{upd}_{\ell'}(V, s')) = \mathit{lkp}_\ell(s') \\ & & & \text{by axiom 12.2.2.2} \end{array}$$

□

Next, we define $\mathrm{nf}(s) \in \mathit{Store}$, a normal form of s, by:

$$\begin{array}{rcl} \mathrm{nf}(\mathit{emp}) & \equiv & \mathit{emp} \\ \mathrm{nf}(\mathit{upd}_\ell(u, s)) & \equiv & \mathit{upd}_\ell(u, \mathrm{nf}(s \setminus \ell)) \end{array}$$

In $\mathrm{nf}(s)$ each $\ell \in \mathit{dom}(s)$ occurs just once.

Corollary 12.2.9. $\vdash s = \mathrm{nf}(s)$

Proof. By simultaneous induction over s and the cardinality of $dom(s)$. If $dom(s) = \emptyset$ then $s \equiv emp \equiv \mathrm{nf}(emp)$. Otherwise, suppose $s \equiv upd_\ell(u, s')$, then

$$\mathrm{nf}(upd_\ell(u, s')) \quad \equiv \quad upd_\ell(u, \mathrm{nf}(s' \setminus \ell)) \quad \text{by definition of nf}$$

Now if $\ell \notin dom(s')$ then $s' \setminus \ell \equiv s'$ by Lemma 12.2.7.1 and $\vdash s' = \mathrm{nf}(s')$ by induction, and we are done.

If $\ell \in dom(s')$ then, recalling that $s \equiv upd_\ell(u, s')$:

$$\begin{array}{rcll} s & = & upd_\ell(lkp_\ell(s), s \setminus \ell) & \text{by Lemma 12.2.8} \\ & = & upd_\ell(lkp_\ell(s), \mathrm{nf}(s \setminus \ell)) & \text{by 2 and by ind. since } |s| > |dom(s \setminus \ell)| \\ & \equiv & upd_\ell(lkp_\ell(s), \mathrm{nf}(s' \setminus \ell)) & \text{since } s \setminus \ell \equiv upd_\ell(u, s') \setminus \ell \equiv s' \setminus \ell \\ & = & upd_\ell(u, \mathrm{nf}(s' \setminus \ell)) & \text{since } lkp_\ell(s) \equiv lkp_\ell(upd_\ell(u, s')) = u \\ & \equiv & \mathrm{nf}(s) & \text{by definition of nf} \end{array}$$

□

Theorem 12.2.10 (Completeness of Store Axioms)**.** $\vdash s = t \Leftrightarrow s \simeq t$

Proof. The only if part is proved by Lemma 12.2.6. To prove the if part, assume $s \simeq t$ and let $L = dom(s) = dom(t)$, which is finite; then we reason by induction over the cardinality of L. If $L = \emptyset$ then $s \equiv emp \equiv t$ and the thesis follows by reflexivity. Otherwise, let $\ell \in L$ be arbitrary; by Lemma 12.2.7.3 we have that $s \setminus \ell \simeq t \setminus \ell$, therefore we may assume by induction $\vdash s \setminus \ell = t \setminus \ell$ since $L \setminus \{\ell\} = dom(s \setminus \ell) = dom(t \setminus \ell)$ has cardinality $|L| - 1$. Now

$$\begin{array}{rcll} s & = & upd_\ell(lkp_\ell(s), s \setminus \ell) & \text{by Lemma 12.2.7.3} \\ & = & upd_\ell(lkp_\ell(t), s \setminus \ell) & \text{by the hypothesis } s \simeq t \\ & = & upd_\ell(lkp_\ell(t), t \setminus \ell) & \text{by induction} \\ & = & t & \text{by Lemma 12.2.7.3} \end{array}$$

□

The consequence of Theorem 12.2.10 is that each store term is equated to a normal form:

Corollary 12.2.11. *If $s \in Store$ with non empty $dom(s) = \{\ell_1, \ldots, \ell_n\}$ then there exist $V_1, \ldots, V_n \in Val$ such that*

$$\vdash s = upd_{\ell_1}(V_1, \cdots upd_{\ell_n}(V_n, emp) \cdots)$$

Therefore the algebra of stores and lookup terms is decidable.

Proof. Observe that, if $dom(s) = \{\ell_1, \ldots, \ell_n\}$ is non empty then

$$\mathrm{nf}(s) \equiv upd_{\ell_1}(u_1, \cdots upd_{\ell_n}(u_n, emp) \cdots)$$

By Lemma 12.2.3 we know that there exists $V_i \in Val$ such that $\vdash u_i = V_i$ for all $i = 1, \ldots, n$, and these are unique for each u_i by Lemma 12.2.4. Then we have

$$\vdash \mathrm{nf}(s) = upd_{\ell_1}(V_1, \cdots upd_{\ell_n}(V_n, emp) \cdots)$$

$$([V] \star (\lambda x.M), s) \to (M\{V/x\}, s) \quad (\beta_c)$$

$$\frac{(M, s) \to (N, t)}{(M \star V, s) \to (N \star V, t)} \quad (\star\text{-}red)$$

$$\frac{lkp_\ell(s) = V}{(get_\ell(\lambda x.M), s) \to (M\{V/x\}, s)} \quad (get\text{-}red)$$

$$(set_\ell(V, M), s) \to (M, upd_\ell(V, s)) \quad (set\text{-}red)$$

Figure 12.1: Reduction relation.

where the only differences among the left and the right hand sides are in the ordering of the ℓ_i, which is immaterial by axiom 12.2.2.5. By Corollary 12.2.9 we conclude that

$$\vdash s = upd_{\ell_1}(V_1, \cdots upd_{\ell_n}(V_n, emp) \cdots)$$

Combining this with Theorem 12.2.10, we conclude that $\vdash s = t$ if and only if $dom(s) = L = dom(t)$ and $\mathrm{nf}(s) \simeq \mathrm{nf}(t)$, which is decidable as nf is computable and extensional equality is decidable.

$\square$

The reduction relation. A *configuration* is a pair (M, s), with $M \in Com$ and $s \in Store$; then the *one-step reduction* is the binary relation over configurations inductively defined by the rules in Figure 12.1.

The reduction relation is deterministic, reflecting the strictly sequential nature of evaluation for programming languages with side-effects. A configuration of the shape $([V], t)$ is irreducible, and it is the result of the evaluation of (M, s) whenever $(M, s) \to^* ([V], t)$, where $\to^*$ is the reflexive and transitive closure of $\to$. Infinite reductions exist; consider the term:

$$\Omega_c \equiv [(\lambda x.[x] \star x)] \star (\lambda x.[x] \star x)$$

which is such that $(\Omega_c, s) \to (\Omega_c, s)$, for any s.

Not every irreducible configuration represents some properly terminating computation; the simplest example is $(get_\ell(\lambda x.[x]), emp)$, because $lkp_\ell(emp)$ is undefined. In general, the set of *blocked configurations* can be inductively defined by:

$$\begin{array}{rcll} (B, s) & ::= & (get_\ell(\lambda x.M), s) & \text{for } \ell \notin dom(s) \\ & \mid & (B \star V, s) & \end{array}$$

Example 12.2.12. To see how the mutation of the value associated to some location ℓ is modelled in the operational semantics, consider the following reduction, where we omit external brackets of configurations for readability:

$$\begin{array}{crl} & set_\ell(W, set_\ell(V, get_\ell(\lambda x.[x]))), & s \\ \to & set_\ell(V, get_\ell(\lambda x.[x])), & upd_\ell(W, s) \\ \to & get_\ell(\lambda x.[x]), & upd_\ell(V, upd_\ell(W, s)) \\ \to & [V], & upd_\ell(V, upd_\ell(W, s)) \end{array}$$

where the last step is justified by the equation

$$lkp_\ell(upd_\ell(V, upd_\ell(W, s))) = V$$

Then we say that the value W, formerly associated to ℓ, has been *overridden* by V in $upd_\ell(V, upd_\ell(W, s))$. Indeed $\vdash upd_\ell(V, upd_\ell(W, s)) = upd_\ell(V, s)$.

Example 12.2.13. Define the abbreviation: $M; N \equiv M \star \lambda\_.N$ for *sequential composition*, where _ is a dummy variable not occurring in N; then, omitting external brackets of configurations as before:

$$\begin{array}{rrl} & set_\ell(V, [W])\,;\, get_\ell(\lambda x.N), & s \\ \equiv & set_\ell(V, [W]) \star \lambda\_.get_\ell(\lambda x.N), & s \\ \to & [W] \star \lambda\_.get_\ell(\lambda x.N), & upd_\ell(V, s) \\ \to & get_\ell(\lambda x.N), & upd_\ell(V, s) \\ \to & N\{V/x\}, & upd_\ell(V, s) \end{array}$$

Notice that, while the value W is discarded according to the semantics of sequential composition, the side-effect of saving V to the location ℓ binds x to V in N.

12.3 Convergence

Following [PS98], in [Gav19] the operational semantics of the imperative lambda calculus is defined via a *convergence predicate*. This is a relation among configurations and their results, which in [Gav19] are semantical objects. To adapt such definition to our syntactical setting, we use store terms instead.

Recall that Val^0 and Com^0 are the sets of closed values and computations, respectively. We say that $s \in Store$ is *closed* if $V \in Val^0$ for all the value terms V occurring in s; we denote the set of *closed store terms* by $Store^0$. We say that the configuration (M, s) is *closed* if both M and s are such; we call a *result* any pair (V, s) of closed V and s.

Definition 12.3.1 (Big-step)**.** *The relation $(M, s) \Downarrow (V, t)$ among the closed configuration (M, s) and the result (V, t) is inductively defined by the rules in Figure 12.2.*

As suggested by the name used in 12.3.1, convergence is nothing else than the big-step semantics corresponding to the small-step semantics we have defined via the reduction relation in Figure 12.1.

Lemma 12.3.2. $(M, s) \to^* (N, t) \Rightarrow (M \star V, s) \to^* (N \star V, t)$

Proof. By induction over the definition of $(M, s) \to^* (N, t)$. The base case $(M, s) \equiv (N, t)$ is obvious. Otherwise, $(M, s) \to (M', s') \to^* (N, t)$ for some M', s'; then $(M \star V, s) \to (M' \star V, s')$ by rule ($\star$-*red*), and $(M' \star V, s') \to^* (N \star V, t)$ by induction. $\square$

Lemma 12.3.3. $(M, s) \to (N, s')\ \ \&\ \ (N, s') \Downarrow (V, t) \Rightarrow (M, s) \Downarrow (V, t)$

$$\frac{}{([V], s) \Downarrow (V, s)} \quad (\mathit{Val\text{-}conv})$$

$$\frac{(M, s) \Downarrow (V, s') \quad (N\{V/x\}, s') \Downarrow (W, t)}{(M \star (\lambda x.N), s) \Downarrow (W, t)} \quad (\star\text{-}\mathit{conv})$$

$$\frac{\mathit{lkp}_\ell(s) = V \quad (M\{V/x\}, s) \Downarrow (W, t)}{(\mathit{get}_\ell(\lambda x.M), s) \Downarrow (W, t)} \quad (\mathit{get\text{-}conv})$$

$$\frac{(M, \mathit{upd}_\ell(V, s)) \Downarrow (W, t)}{(\mathit{set}_\ell(V, M), s) \Downarrow (W, t)} \quad (\mathit{set\text{-}conv})$$

Figure 12.2: Convergence predicate.

Proof. By induction over $(M, s) \to (N, s')$.

Case (β_c): $(M \equiv [W] \star (\lambda x.M'), s) \to (M'\{W/x\}, s)$; by hypothesis we know that $(M'\{W/y\}, s) \Downarrow (V, t)$, then:

$$\frac{\dfrac{}{([W], s) \Downarrow (W, s)}\ (\mathit{Val\text{-}conv}) \qquad (M'\{W/y\}, s) \Downarrow (V, t)}{([W] \star \lambda y.M', s) \Downarrow (V, t)} \quad (\star\text{-}\mathit{conv})$$

Case $\star$-red: $(M' \star W, s) \to (N' \star W, s')$ because $(M', s) \to (N', s')$.

In this case we have $W \equiv \lambda x.L$, since $W \in \mathit{Val}^0$. By hypothesis $(N' \star \lambda x.L, s') \Downarrow (V, t)$, so there exist W', s'' such that

$$\frac{(N', s') \Downarrow (W', s'') \quad (L\{W'/x\}, s'') \Downarrow (V, t)}{(M' \star \lambda x.L, s') \Downarrow (V, t)} \quad (\star\text{-}\mathit{conv})$$

Since $(M', s) \to (N', s')$ and $(N', s') \Downarrow (W', s'')$, by induction hypothesis $(M', s') \Downarrow (W', s'')$ and therefore we have the derivation

$$\frac{(M', s') \Downarrow (W', s'') \quad (L\{W'/x\}, s'') \Downarrow (V, t)}{(M' \star \lambda x.L, s') \Downarrow (V, t)} \quad (\star\text{-}\mathit{conv})$$

The remaining cases $(\mathit{get}_\ell(\lambda x.M'), s) \to (M'\{W/x\}, s)$ where $\mathit{lkp}_\ell(s) = W$, and $(\mathit{set}_\ell(W, M'), s) \to (M', \mathit{upd}_\ell(W, s) \equiv s')$, easily follow by induction.

□

Proposition 12.3.4. *For all $M \in \mathit{Com}^0$, $V \in \mathit{Val}^0$, and $s, t \in \mathit{Store}^0$, we have:*

$$(M, s) \Downarrow (V, t) \iff (M, s) \to^* ([V], t)$$

Proof. The only if part is proved by induction over the definition of $(M, s) \Downarrow (V, t)$, the case $(\mathit{Val\text{-}conv})$ is trivial. Consider the case $(\star\text{-}\mathit{conv})$. We know by induction that

HI_1 $(M,S) \to^* ([V],s')$

HI_2 $(N\{V/x\},s') \to^* ([W],t)$

$$\begin{array}{rcll}
(M \star \lambda x.N, s) & \to^* & ([V] \star \lambda x.N, s') & \text{by } (HI_1) \text{ and } \textit{Lemma } 12.3.2 \\
& \to & (N\{V/x\}, s') & \text{by } \beta_c \\
& \to^* & ([W], t) & \text{by } (HI_2)
\end{array}$$

The cases of (*get-conv*) and (*set-conv*) are immediate by induction hypothesis.

The if part is proved by induction over $(M,s) \to^* ([V],t)$. The base case is $(M \equiv [V], s \equiv t)$ then $([V],s) \Downarrow (V,s)$ by (*Val-conv*). Otherwise, there exists (N,s') such that $(M,s) \to (N,s') \to^* ([V],t)$. By induction $(N,s') \Downarrow (V,t)$ so that $(M,s) \Downarrow (V,t)$ by Lemma 12.3.3.

□

Proposition 12.3.4 justifies the name "result" we have given to the pairs (V,t); indeed, since the reduction relation is deterministic, the same holds for convergence, so that computations can be seen as partial functions from stores to results:

$$M(s) = \begin{cases} (V,t) & \text{if } (M,s) \Downarrow (V,t) \\ \text{undefined} & \text{else} \end{cases} \tag{12.1}$$

Notice that $M(s)$ is undefined if either the reduction out of (M,s) is infinite, or if it reaches some blocked configuration (B,t).

The convergence predicate essentially involves the stores, that are part of configurations and results, and are dynamic entities, much as when executing imperative programs. However, when reasoning about programs, namely (closed) computations, we abstract from the infinitely many stores that can be fed together with inputs to the program, and returned together with their outputs. This motivates the following definition:

Definition 12.3.5 (Convergence). *For $M \in Com^0$ and $s \in Store^0$ we set:*

1. $(M,s) \Downarrow \iff \exists V,t.\ (M,s) \Downarrow (V,t)$
2. $M \Downarrow \iff \forall s \in Store^0.\ (M,s) \Downarrow$

The definition of $(M,s) \Downarrow$ is similar to that in [PS98], but simpler since we do not treat local states. The definition of $M \Downarrow$ is equivalent to $(M, emp) \Downarrow$. To see why, consider the term $P \equiv get_\ell(\lambda x.[x])$; the configuration (P,s) converges if and only if $lkp_\ell(s)$ is defined; in particular, the configuration (P, emp) is blocked and it does not yield any result. Conversely, if a given configuration (M, emp) converges, either no *get* operation occurs as the main operator of a configuration (N,s') in the convergent out of (M, emp), or if any such operation does, with say $N \equiv get_\ell(\lambda x.M')$, then there exists a term of shape $set_\ell(V,N')$ in a configuration that precedes (N,s') in the reduction path, "initializing" to V the value of ℓ. Therefore, if $(M, emp) \Downarrow$ then $(M,s) \Downarrow$, for any s.

We say that (M,s) is *divergent*, written $(M,s) \Uparrow$, if not $(M,s) \Downarrow$. By Proposition 12.3.4 a divergent configuration (M,s) either reduces to a blocked configuration or the unique reduction out of (M,s) is infinite.

CHAPTER 13

STORE MONAD

13.1 Denotational Semantics

We illustrate a denotational semantics of the calculus λ_{imp} introduced in Section 12.1 in the category of domains. The model is based on the solution of the domain equation:

$$D = [D \to [S \to (D \times S)_\bot]] \tag{13.1}$$

where S is a suitable space of stores over D, and $\mathbb{S}\,D = [S \to (D \times S)_\bot]$ is a variant of the state monad in [Mog91], which is called the partiality and state monad in [DGL17].

Before embarking on solving the equation (13.1), let us recall definitions of monad $\mathbb{S}$ (for a general definition of monad see Definition 2.1.1), algebraic operators, and fix notations.

The partiality and state monad. We recall Wadler's type-theoretic definition of monads [Wad92, Wad95], that is at the basis of their successful implementation in Haskell language, a natural interpretation of the calculus is into a cartesian closed category (ccc), such that two families of combinators, or a pair of polymorphic operators called the "unit" and the "bind", exist satisfying the *monad laws.* In what follows, $\mathcal{C}$ will be a *concrete ccc*, namely with sets as objects and certain maps as morphisms. Examples that are relevant to us are the category **Dom** of Scott domains with continuous functions, and its full subcategory ω-**ALG** of algebraic lattices with a countable basis.

Now, let us look closer how monads model effectful computations. Let X be an object in **Dom**; then TX is the domain, or the type, of computations with values in X. In general TX has a richer structure than X itself, modeling partial computations, exceptions, non-determinism etcetera, including side-effects. The mapping $unit_X : X \to TX$ is interpreted as the trivial computation $unit_X(x)$ just returning the value x, and it is an embedding if T satisfies the requirement that all the $unit_X$ are monos. Now a function $f : X \to TY$ models an "impure" program P with input in X and output in Y via a pure function returning the computation $f(x) \in TY$.

To relate Equation (13.1) to the monad $\mathbb{S}$ we have to be more precise about the domain S of stores. Given a domain X, let $X_\bot$ be the lifting of X, namely the poset $X \cup \{\bot\}$ (with $\bot \notin X$) where $x \sqsubseteq_{X_\bot} x'$ if either $x = \bot$ or $x \sqsubseteq_X x'$. Then we define $(X_\bot)^{\mathbf{L}}$ as the domain of *stores* over X with locations in $\mathbf{L}$. Such a poset, ranged over by ς, can be seen as the domain of partial maps from locations in $\mathbf{L}$ to points of X, where $\varsigma(\ell) = \bot$ represents the fact that $\ell \notin \mathit{dom}(\varsigma)$. It is ordered pointwise, that is $\varsigma \sqsubseteq_{(X_\bot)^{\mathbf{L}}} \varsigma'$ if $\varsigma(\ell) \sqsubseteq_{X_\bot} \varsigma'(\ell)$ for all $\ell \in \mathbf{L}$.

Remark 13.1.1. In [dT21a] the domain of stores over X was just defined as $X^{\mathbf{L}}$. This definition is sound w.r.t. the interpretation of terms, but it equates the everywhere undefined store, which is the interpretation of *emp* in Definition 13.4.1, with any store ς such that $\varsigma(\ell) = \bot_X$ for any ℓ. As we will see such a distinction is crucial when establishing the characterization of convergence in Theorem 14.2.6, while it was erroneously stated in [dT21b].

We also notice that the similar definition of the store space over X as $(X^{\mathbf{L}})_\bot$ would not be sufficient for our proposes. Indeed, although the everywhere undefined store is now distinct from all the others, the stores $\ell \mapsto \bot_X$ and $\ell' \mapsto \bot_X$ with $\bot_X$ being the infimum of X, would be identified, even if $\ell \neq \ell'$, which is equally wrong.

The definition of $FX = (X_\bot)^{\mathbf{L}}$ is clearly functorial by setting $F(g) = g_\bot \circ \_ : (X_\bot)^{\mathbf{L}} \to (Y_\bot)^{\mathbf{L}}$, where for any $g : X \to Y$ $g_\bot(x) = g(x)$ if $x \in X$, and $g_\bot(\bot) = \bot$ else. The functor F is locally continuous, being the composition of lifting and exponentiation, that are both such.

We are now in place to properly define the partiality and state monad $\mathbb{S}$ in terms of Definition 2.1.1.

Definition 13.1.2. (Partiality and state monad) *Given the domain $S = (X_\bot)^{\mathbf{L}}$ representing a notion of state, we define the partiality and state monad $(\mathbb{S}, \mathit{unit}, \star)$, as the mapping*

$$\mathbb{S}\, X = [S \to (X \times S)_\bot]$$

where $(X \times S)_\bot$ is the lifting of the cartesian product $X \times S$, equipped with two (families of) operators unit and $\star$ defined as follows:

$$\begin{array}{l} \mathit{unit}\; x ::= \lambda\!\!\lambda \varsigma.(x, \varsigma) \\ (c \star f)\varsigma = f^\dagger(c)(\varsigma) ::= \begin{cases} f\,(x)(\varsigma') & \text{if } c(\varsigma) = (x, \varsigma') \neq \bot \\ \bot & \text{if } c(\varsigma) = \bot \end{cases} \end{array}$$

where we omit subscripts.

Remark 13.1.3. A function $f : X \to \mathbb{S}\, Y$ has type $X \to S \to (Y \times S)_\bot$, which is isomorphic to $(X \times S) \to (Y \times S)_\bot$; if f is the interpretation of an "imperative program" P, and it is the currying of f', then we expect that $f\, x\, \varsigma = f'(x, \varsigma) = (y, \varsigma')$ if y and ς' are, respectively, the value and the final store obtained from the evaluation of $P(x)$ starting in ς, if it terminates; $f\, x\, \varsigma = f'(x, \varsigma) = \bot$, otherwise. Clearly, f' is the familiar interpretation of the imperative program P as a state transformation mapping.

A domain equation. Evidently, the domain $FX = (X_\perp)^{\mathbf{L}}$ depends on X itself; in contrast, while defining $\mathbb{S}\,X$ the domain S has to be fixed, since otherwise the definition of the $\star$ operator does not make sense, and $\mathbb{S}$ is not even a functor. A solution would be to take $S = FD$, where $D \cong [D \to \mathbb{S}\,D]$, but this is clearly circular.

To break the circularity, we define the mixed-variant bi-functor $G : \mathbf{Dom}^{op} \times \mathbf{Dom} \to \mathbf{Dom}$ by

$$G(X, Y) = [FX \to (Y \times FY)_\perp]$$

whose action on morphisms is illustrated by the diagram:

$$\begin{tikzcd}
FX' \arrow[r, "Ff"] \arrow[d, dashed, "G(f,g)(\alpha)"'] & FX \arrow[d, "\alpha"] \\
(Y' \times FY')_\perp & (Y \times FY)_\perp \arrow[l, "(g \times Fg)_\perp"]
\end{tikzcd}$$

where $f : X' \to X$, $g : Y \to Y'$ and $\alpha \in G(X, Y)$. Now it is routine to prove that G is locally continuous so that, by the inverse limit technique, we can find the initial solution to the domain equation (which is the same as Equation (13.1)):

$$D = [D \to G(D, D)] \tag{13.2}$$

In summary we have:

Theorem 13.1.4. *There exists a domain D such that the state monad $\mathbb{S}$ with state domain $S = (D_\perp)^{\mathbf{L}}$ is a solution in **Dom** to the domain equation:*

$$D = [D \to \mathbb{S}\,D]$$

Moreover, it is initial among all solutions to such equation.

Proof. Take D to be the (initial) solution to Equation (13.2); now if $S = FD = (D_\perp)^{\mathbf{L}}$ then $\mathbb{S}\,D = G(D, D)$. □

Algebraic operations over $\mathbb{S}$. Monads are about composition of morphisms of some special kind. However, thinking of $f : X \to \mathbb{S}X$ as the meaning of a program with side-effects does not tell anything about side-effects themselves, that are produced by reading and writing values from and to stores in S.

To model effects associated to a monad, Plotkin and Power have proposed in [PP02, PP03, Pow06] a theory of *algebraic operations.* A gentle introduction to the theory can be found in [Gav19], chap. 3, from which we borrow the notation.

Suppose that T is a monad over a category $\mathcal{C}$ with terminal object $\mathbf{1}$ and all finite products; then an algebraic operation $\mathbf{op}$ with arity n is a family of morphisms $\mathbf{op}_X : (TX)^n \to TX$, where $(TX)^n = TX \times \cdots \times TX$ is the n-times product of TX with itself, such that

$$\mathbf{op}_X \circ \Pi_n f^\dagger = f^\dagger \circ \mathbf{op}_X \tag{13.3}$$

where $f : X \to TX$ and $\Pi_n f^\dagger = f^\dagger \times \cdots \times f^\dagger : (TX)^n \to (TX)^n$. In case of the concrete category **Dom**, **1** is a singleton and products are sets of tuples, ordered componentwise. Then $\mathbf{op}_X$ is an operation of arity n of an algebra with carrier TX; since $f^\dagger : TX \to TX$, we have that $(\Pi_n f^\dagger)\langle x_1, \ldots, x_n\rangle = \langle f^\dagger(x_1), \ldots, f^\dagger(x_n)\rangle$, and Equation (13.3) reads as:

$$\mathbf{op}_X(f^\dagger(x_1), \ldots, f^\dagger(x_n)) = f^\dagger(\mathbf{op}_X(x_1, \ldots, x_n))$$

namely the functions $f^\dagger$ are homomorphisms w.r.t. $\mathbf{op}_X$.

Algebraic operations $\mathbf{op}_X : (TX)^n \to TX$ do suffice in case T is, say, the nondeterminism or the output monad, but cannot model side-effects operations in case of the store monad. This is because read and write operations implement a bidirectional action of stores to programs and of programs to stores. What is needed instead is the generalized notion of operation proposed in [PP02] (and further studied in [HP06]); such construction, that is carried out in a suitable enriched category, can be instantiated to the case of **Dom** as follows:

$$\mathbf{op} : P \times (TX)^A \to TX \cong (TX)^A \to (TX)^P$$

where P is the domain of parameters and A of generalized arities, and the rightmost "type" is the interpretation in **Dom** of Def. 1 in [PP02]. For such operations Equation (13.3) is generalized as follows (see [Gav19], Def. 13):

$$\mathbf{op}(p, k) \star f = f^\dagger(\mathbf{op}(p, k)) = \mathbf{op}(p, f^\dagger \circ k) = \mathbf{op}(p, \lambda\!\!\lambda x.(k(x) \star f)) \qquad (13.4)$$

where $f : X \to TX$, $p \in P$ and $k : A \to TX$.

Denotational semantics of terms. Given the monad $(\mathbb{S}, \eta, \_^\dagger)$ and the domain D from Theorem 13.1.4, we first interpret the constants get_ℓ and set_ℓ as generalized operations over $\mathbb{S}$. By taking $P = \mathbf{1}$ and $A = D$ we define:

$$[\![get_\ell]\!] : \mathbf{1} \times (\mathbb{S}D)^D \to \mathbb{S}D \simeq (\mathbb{S}D)^D \to \mathbb{S}D \quad \text{by} \quad [\![get_\ell]\!]\, d\, \varsigma = d\,(\varsigma\,(\ell))\,\varsigma$$

where $d \in D$ is identified with its image in $D \to \mathbb{S}\, D$, and $\varsigma \in S = D^{\mathbf{L}}$. On the other hand, taking $P = D$ and $A = \mathbf{1}$, we define:

$$[\![set_\ell]\!] : D \times (\mathbb{S}D)^{\mathbf{1}} \to \mathbb{S}D \simeq D \times \mathbb{S}D \to \mathbb{S}D \quad \text{by } [\![set_\ell]\!](d, c)\, \varsigma = c(\varsigma[\ell \mapsto d])$$

where $c \in \mathbb{S}D = S \to (D \times S)_\perp$ and $\varsigma[\ell \mapsto d]$ is the store sending ℓ to d and it is equal to ς, otherwise.

Then we interpret values from *Val* in D and computations from *Com* in $\mathbb{S}D$. More precisely we define the maps $[\![\cdot]\!]^D : \mathit{Val} \to \mathit{Term\text{-}Env} \to D$ and $[\![\cdot]\!]^{\mathbb{S}D} : \mathit{Com} \to \mathit{Term\text{-}Env} \to \mathbb{S}D$, where $\mathit{Term\text{-}Env} = \mathit{Var} \to D$ is the set of environments interpreting term variables:

Definition 13.1.5. *A λ_{imp}-**model** is a structure $\mathcal{D} = (D, \mathbb{S}, [\![\cdot]\!]^D, [\![\cdot]\!]^{\mathbb{S}D})$ such that:*

1. *D is a domain s.t. $D \cong D \to \mathbb{S}D$ via (Φ, Ψ), where $\mathbb{S}$ is the partiality and state monad of stores over D;*

2. for all $e \in$ Term-Env, $V \in$ Val and $M \in$ Com:

$$\begin{array}{rcl}
[\![x]\!]^D e & = & e(x) \\
[\![\lambda x.M]\!]^D e & = & \Psi(\lambda\!\!\!\lambda\, d \in D.\, [\![M]\!]^{\mathbb{S}D} e[x \mapsto d]) \\
[\![\,[V]\,]\!]^{\mathbb{S}D} e & = & unit([\![V]\!]^D e) \\
[\![M \star V]\!]^{\mathbb{S}D} e & = & ([\![M]\!]^{\mathbb{S}D} e) \star \Phi([\![V]\!]^D e) \\
[\![get_\ell(\lambda x.M)]\!]^{\mathbb{S}D} e & = & [\![get_\ell]\!]\, \Phi([\![\lambda x.M]\!]^D e) \\
[\![set_\ell(V, M)]\!]^{\mathbb{S}D} e & = & [\![set_\ell]\!]([\![V]\!]^D e, [\![M]\!]^{\mathbb{S}D} e)
\end{array}$$

By unravelling definitions and applying to an arbitrary store $\varsigma \in S$, the last two clauses can be written:

$$\begin{array}{rcl}
[\![get_\ell(\lambda x.M)]\!]^{\mathbb{S}D} e\, \varsigma & = & [\![M]\!]^{\mathbb{S}D} (e[x \mapsto \varsigma(\ell)])\, \varsigma \\
[\![set_\ell(V, M)]\!]^{\mathbb{S}D} e\, \varsigma & = & [\![M]\!]^{\mathbb{S}D} e\, (\varsigma[\ell \mapsto [\![V]\!]^D e])
\end{array}$$

We say that the equation $M = N$ is *true* in $\mathcal{D}$, written $\mathcal{D} \models M = N$, if $[\![M]\!]^{\mathbb{S}D} e = [\![N]\!]^{\mathbb{S}D} e$ for all $e \in$ *Term-Env*.

Proposition 13.1.6. *The following equations are true in $\mathcal{D}$:*

1. $[V] \star (\lambda x.M) = M\{V/x\}$

2. $M \star \lambda x.[x] = M$

3. $(L \star \lambda x.M) \star \lambda y.N = L \star \lambda x.(M \star \lambda y.N)$

4. $get_\ell(\lambda x.M) \star W = get_\ell(\lambda x.(M \star W))$

5. $set_\ell(V, M) \star W = set_\ell(V, M \star W)$

where $x \notin FV(\lambda y.N)$ in (3) and $x \notin FV(W)$ in (4).

Proof. By definition and straightforward inferences. For example, to see (4), let $\varsigma \in S$ be arbitrary and $e' = e[x \mapsto \varsigma(\ell)]$; then, omitting the apices of the interpretation mappings $[\![\cdot]\!]$:

$$[\![get_\ell(\lambda x.M) \star W]\!] e\, \varsigma = ([\![W]\!] e)^\dagger ([\![M]\!] e')\, \varsigma$$

On the other hand:

$$[\![get_\ell(\lambda x.(M \star W))]\!] e\, \varsigma = ([\![\lambda x.(M \star W)]\!] e)\, \varsigma(\ell)\, \varsigma = [\![M \star W]\!] e'\, \varsigma = ([\![W]\!] e')^\dagger ([\![M]\!] e')\, \varsigma$$

But $x \notin FV(W)$ implies $[\![W]\!] e' = [\![W]\!] e$, and we are done. □

Equations (1)-(3) in Proposition 13.1.6 are the *monadic laws* from Definition 2.1.1, Equations (4)-(5) are instances of Equation (13.4).

13.2 The Filter-Model Construction

In this section we recap some basic facts about intersection types and algebraic lattices with countable basis, which are well known from the literature; then we apply these results to the domain equation studied in the previous section and build a filter-model of λ_{imp}.

Recall from domain theory that a non-empty subset $X \subseteq D$ of a partial order $(D, \sqsubseteq_D)$ is *directed* if for all $x, y \in X$ there exists $z \in X$ with $x \sqsubseteq_D z \;{}_D\!\sqsupseteq y$. D is said to have all *directed sups* if the sup $\bigsqcup X \in D$ exists whenever X is directed. We often write $\bigsqcup^{\uparrow} X$ to stress $\bigsqcup X$ is directed. A point $e \in D$ is *compact* if for all directed X, $e \sqsubseteq_D \bigsqcup^{\uparrow} X$ implies $e \sqsubseteq_D x$ for some $x \in X$; the set of compact points of D is denoted $K(D)$.

Here we consider the category ω-**ALG** of ω-algebraic lattices. ω-**ALG** is the full subcategory of **Dom** of complete lattices $(D, \sqsubseteq_D)$ such that $K(D)$ is countable and D is *algebraic*, namely $d = \bigsqcup^{\uparrow}\{e \in K(D) \mid e \sqsubseteq_D d\}$ is a directed sup for all $d \in D$. In case of algebraic domains the upward cones of compact points form a basis for the Scott topology over D; therefore abusing terminology the set $K(D)$ is often referred to as a “basis” for such a topology.

As a complete lattice, D has arbitrary sups, but a morphism $f : D \to E \in \omega$-**ALG** is just a *Scott-continuous* map preserving directed sups, that is $f(\bigsqcup^{\uparrow} X) = \bigsqcup^{\uparrow}_{x \in X} f(x)$; a continuous function does not necessarily preserve the sup $\bigsqcup Y$ for arbitrary $Y \subseteq D$.

Definition 13.2.1. *An* intersection type theory*, shortly* itt*, is a pair $Th_A = (\mathcal{L}_A, \leq_A)$ where $\mathcal{L}_A$, the* language *of Th_A, is a countable set of type expressions closed under $\wedge$, and $\omega_A \in \mathcal{L}_A$ is a special constant; $\leq_A$ is a pre-order over $\mathcal{L}_A$ closed under the following rules:*

$$\alpha \leq_A \omega_A \qquad \alpha \wedge \beta \leq_A \alpha \qquad \alpha \wedge \beta \leq_A \beta \qquad \frac{\alpha \leq_A \alpha' \quad \beta \leq_A \beta'}{\alpha \wedge \beta \leq_A \alpha' \wedge \beta'}$$

In the literature, the operator $\wedge$ is called *intersection*, and ω_A the *universal type*. Informally, Th_A is identified with the set of inequalities $\alpha \leq_A \beta$, so that, abusing terminology, we shall also say that $\leq_A$ is a type theory. Clearly, the quotient $\mathcal{L}_A/_{\leq_A}$ is an inf-semilattice, with $\wedge$ as inf and ω_A as top element. Being the meet commutative and associative, we shall write $\alpha_1 \wedge \cdots \wedge \alpha_n$ omitting braces and abstracting from the order; also we use the notations $\bigwedge_{i=1}^n \alpha_i$ and $\bigwedge_{i \in I} \alpha_i$ for $\alpha_1 \wedge \cdots \wedge \alpha_n$ where $I = \{1, \ldots, n\}$; in particular $\bigwedge_{i \in \emptyset} \alpha_i = \omega_A$. Finally we write $\alpha =_A \beta$ if $\alpha \leq_A \beta \leq_A \alpha$; since $\alpha \wedge \alpha =_A \alpha$, we assume that in the type expression $\bigwedge_{i \in I} \alpha_i$ all the α_i are distinct.

Definition 13.2.2. *A non empty $F \subseteq \mathcal{L}_A$ is a* filter *of Th_A if it is closed under $\wedge$ and upward closed w.r.t. $\leq_A$; let $\mathcal{F}_A$ be the set of filters of Th_A. The* principal filter over *α is the set $\{\beta \in \mathcal{L}_A \mid \alpha \leq_A \beta\}$.*

For any $X \subseteq \mathcal{L}_A$ we write $\uparrow_A X$ for the least filter in $\mathcal{F}_A$ including X; this can be equivalently defined as $\bigcap\{F \in \mathcal{F}_A \mid X \subseteq F\}$ or as the set $\{\beta \in \mathcal{L}_A \mid$

$\exists n, \alpha_1, \dots, \alpha_n \in X.\ \bigwedge_{i=1}^{n} \alpha_i \leq_A \beta\}$; in particular the principal filter over α is $\uparrow_A \{\alpha\}$, also written $\uparrow_A \alpha$, coinciding with the upward closure of $\{\alpha\}$. The mapping $\uparrow\cdot$ is evidently a closure operator; in the following we shall write just $\uparrow X$ and $\uparrow\alpha$ whenever A is understood.

The Theorem 13.2.4 is the fundamental fact about intersection types and ω-algebraic lattices; before its proof, we recap some basic properties of the poset $(\mathcal{F}_A, \subseteq)$ in the following lemma.

Lemma 13.2.3. *Consider the poset $(\mathcal{F}_A, \subseteq)$ and let $\mathcal{X} \subseteq \mathcal{F}_A$ be a family of filters:*

1. *the sets $\bigcap \mathcal{X}$ and $\uparrow(\bigcup \mathcal{X})$ are the inf and the sup of $\mathcal{X}$ in $\mathcal{F}_A$, respectively;*
2. *if $\mathcal{X}$ is directed w.r.t. $\subseteq$, then $\uparrow(\bigcup \mathcal{X}) = \bigcup \mathcal{X}$.*

Proof. Part (1) is immediate; in particular, that $\bigcap \mathcal{X}$ is a filter follows by the fact that any $F \in \mathcal{X}$ is such, and the intersection of filters must satisfy the same closure properties of all the elements of $\mathcal{X}$. Note that the closure $\uparrow\cdot$ is necessary in $\uparrow(\bigcup \mathcal{X})$: take $\mathcal{X} = \{\uparrow\alpha, \uparrow\beta\}$, with types $\alpha, \beta \in \mathcal{L}_A$ unrelated w.r.t. $\leq_A$, then $\alpha \wedge \beta \not\in \uparrow\alpha \cup \uparrow\beta$.

Part (2): any directed $\mathcal{X}$ is non empty, hence there is an $F \in \mathcal{X}$ such that $\omega_A \in F \subseteq \bigcup \mathcal{X}$, hence $\bigcup \mathcal{X}$ is nonempty. If $\alpha \in \bigcup \mathcal{X}$ then there is an F such that $\alpha \in F \subseteq \bigcup \mathcal{X}$; hence if $\alpha \leq_A \beta$ then $\beta \in F \subseteq \bigcup \mathcal{X}$ as F is a filter.

Let $\alpha, \beta \in \bigcup \mathcal{X}$; then there are $F, F' \in \mathcal{X}$ such that $\alpha \in F$ and $\beta \in F'$. By hypothesis there exists an $F'' \in \mathcal{X}$ such that $F \subseteq F'' \supseteq F'$, hence $\alpha, \beta \in F''$ and therefore $\alpha \wedge \beta \in F'' \subseteq \bigcup \mathcal{X}$. □

In view of Lemma 13.2.3, we write $\bigsqcup \mathcal{Y} = \uparrow(\bigcup \mathcal{Y})$ for the sup of an arbitrary family of filters $\mathcal{Y}$, and $\bigsqcup^{\uparrow} \mathcal{X} = \bigcup \mathcal{X}$ for a directed family $\mathcal{X}$.

Theorem 13.2.4 (Representation theorem)**.** *The partial order $(\mathcal{F}_A, \subseteq)$ of the filters of an intersection type theory Th_A is an ω-algebraic lattice, whose compact elements are the principal filters. Vice versa, any ω-algebraic lattice A is isomorphic the poset $(\mathcal{F}_A, \subseteq)$ of filters of some intersection type theory Th_A.*

Proof. To prove the first part note that, by Lemma 13.2.3.1, we know that $(\mathcal{F}_A, \subseteq)$ is a complete lattice. Let $\mathcal{X} \subseteq \mathcal{F}_A$ be directed; then $\uparrow\alpha \subseteq \bigcup \mathcal{X} = \bigsqcup^{\uparrow} \mathcal{X}$ and, since $\alpha \in \uparrow\alpha$, there exists $F \in \mathcal{X}$ such that $\alpha \in F$; it follows that $\uparrow\alpha \subseteq F$, since F is upward closed w.r.t. $\leq_A$. This proves that $\uparrow\alpha \in K(\mathcal{F}_A)$; vice versa let $F \in K(\mathcal{F}_A)$ and let $\mathcal{X} = \{\uparrow\alpha \mid \alpha \in F\}$, then $\mathcal{X}$ is directed since if $\alpha, \beta \in F$ then $\alpha \wedge \beta \in F$ and $\uparrow\alpha \subseteq \uparrow\alpha \wedge \beta \supseteq \uparrow\beta$; moreover $F = \bigcup \mathcal{X} = \bigsqcup^{\uparrow} \mathcal{X}$. By assumption F is compact, hence for some $\alpha \in F$ we have $F \subseteq \uparrow\alpha$, but also $\uparrow\alpha \subseteq F$ as F is a filter, hence $F = \uparrow\alpha$.

By the above we conclude that $K(\mathcal{F}_A) = \{\uparrow\alpha \mid \alpha \in \mathcal{L}_A\}$ which is countable as $\mathcal{L}_A$ is such. Finally from the equality $F = \bigsqcup^{\uparrow}\{\uparrow\alpha \mid \alpha \in F\} = \bigcup\{\uparrow\alpha \mid \alpha \in F\}$ we conclude that $(\mathcal{F}_A, \subseteq)$ is ω-algebraic.

To the second part, let $\mathcal{L}_A = \{\alpha_d \mid d \in K(A)\}$ (where each α_d is a new type constant) which is countable by hypothesis, and take $Th_A = \{\alpha_d \leq_A \alpha_e \mid e \sqsubseteq d\}$. By observing that $\alpha_d \wedge_A \alpha_e =_A \alpha_{d \sqcup e}$, where $d \sqcup e \in K(A)$ if $d, e \in K(A)$, and that $\omega_A =_A \alpha_\perp$, we have that Th_A is an intersection type theory and that $\mathcal{L}_A/_{\leq_A}$ is isomorphic to $K^{op}(A)$ ordered by the opposite to the ordering of A. From this it follows that $\mathcal{F}_A$ is isomorphic to the ideal completion of $K(A)$, which in turn is isomorphic to A by algebraicity. □

The above theorem substantiates the claim that intersection type theories are the *logic* of the domains in ω-**ALG**. The theory Th_A is called the Lindenbaum algebra of A in [Abr91]; Theorem 13.2.4 is the object part of the construction establishing that intersection type theories, together with a suitable notion of morphisms among them, do form a category that is equivalent to ω-**ALG**; such equivalence is an instance of Stone duality as studied in [Abr91] w.r.t. the category of 2/3 SFP domains, when ω-**ALG** is viewed as a subcategory of topological spaces: see e.g. [ADCH04].

The second part in the proof of Theorem 13.2.4 is the most relevant to us: it provides a recipe to describe a domain via a formal system, deriving inequalities among type expressions that encode "finite approximations" of points in a domain, hence of the denotations of terms if the domain is a λ_{imp}-model. Therefore, we seek theories Th_D and Th_S such that:

$$\mathcal{F}_D \cong [\mathcal{F}_D \to [\mathcal{F}_S \to (\mathcal{F}_D \times \mathcal{F}_S)_\perp]] \tag{13.5}$$

The first step is to show how the functors involved in the equation above, namely the lifting, the product, and the (continuous) function space can be put in correspondence with the construction of new theories out of the theories which determine the domains combined by the functors. As this is known after [Abr91], we just recall their definitions, and fix the notation.

Definition 13.2.5. *Suppose that the theories Th_A and Th_B are given, then for $\alpha \in \mathcal{L}_A$ and $\beta \in \mathcal{L}_B$ define:*

$$\begin{array}{lll} \mathcal{L}_{A_\perp} & \alpha_\perp & ::= \ \alpha \mid \alpha_\perp \wedge \alpha'_\perp \mid \omega_{A_\perp} \\ \mathcal{L}_{A\times B} & \pi & ::= \ \alpha \times \beta \mid \pi \wedge \pi' \mid \omega_{A\times B} \\ \mathcal{L}_{A\to B} & \phi & ::= \ \alpha \to \beta \mid \phi \wedge \phi' \mid \omega_{A\to B} \\ \mathcal{L}_{A^{\mathbf{L}}} & \sigma & ::= \ \langle \ell : \alpha \rangle \mid \sigma \wedge \sigma' \mid \omega_{A^{\mathbf{L}}} \end{array}$$

Then, we define the following sets of axioms:

1. $\alpha \leq_A \alpha' \Rightarrow \alpha \leq_{A_\perp} \alpha'$.
2. $\omega_{A\times B} \leq_{A\times B} \omega_A \times \omega_B$ *and all instances of*
$$(\alpha \times \beta) \wedge (\alpha' \times \beta') \leq_{A\times B} (\alpha \wedge \alpha') \times (\beta \wedge \beta')$$
3. $\omega_{A\to B} \leq_{A\to B} \omega_A \to \omega_B$ *and all instances of*
$$(\alpha \to \beta) \wedge (\alpha \to \beta') \leq_{A\to B} \alpha \to (\beta \wedge \beta')$$
4. $\omega_{A^{\mathbf{L}}} \leq_{A^{\mathbf{L}}} \langle \ell : \omega_A \rangle$ *and all instances of*
$$\langle \ell : \alpha \rangle \wedge \langle \ell : \alpha' \rangle \leq_{A^{\mathbf{L}}} \langle \ell : \alpha \wedge \alpha' \rangle$$

Finally, the theories $Th_{A\times B}$, $Th_{A\to B}$ and $Th_{A^{\mathbf{L}}}$ are closed under the rules:

$$\frac{\alpha \leq_A \alpha' \quad \beta \leq_B \beta'}{\alpha \times \beta \leq_{A\times B} \alpha' \times \beta'} \qquad \frac{\alpha' \leq_A \alpha \quad \beta \leq_B \beta'}{\alpha \to \beta \leq_{A\to B} \alpha' \to \beta'} \qquad \frac{\alpha \leq_A \alpha'}{\langle \ell : \alpha\rangle \leq_{A^{\mathbf{L}}} \langle \ell : \alpha'\rangle}$$

In the semantics of λ_{imp} functional application and abstraction play a central role; below we define such operations in the case of domains of filters, toward establishing some properties of them.

Definition 13.2.6. *If $X \in \mathcal{F}_{A\to B}$ and $Y \in \mathcal{F}_A$, we define:*

$$X \cdot Y = \{\psi \in \mathcal{L}_B \mid \exists \varphi \in Y.\ \varphi \to \psi \in X\}$$

We write $\varphi =_A \psi$ if $\varphi \leq_A \psi \leq_A \varphi$. For a map $f : \mathcal{F}_A \to \mathcal{F}_B$, define

$$\Lambda(f) = \uparrow_{A\to B} \{\varphi \to \psi \in \mathcal{L}_{A\to B} \mid \psi \in f(\uparrow_A \varphi)\}$$

Lemma 13.2.7. *If $X \in \mathcal{F}_{A\to B}$ and $Y \in \mathcal{F}_A$ then $X \cdot Y \in \mathcal{F}_B$. Moreover, the map $\_ \cdot \_$ is continuous in both its arguments.*

Proof. We have $X \ni \omega_{A\to B} \leq \omega_A \to \omega_B$ and that $\omega_A \in Y$, hence $\omega_B \in X \cdot Y$. Since $\to$ is monotonic in its second argument, $X \cdot Y$ is upward closed. Finally, if $\psi_1, \psi_2 \in X \cdot Y$ then $\varphi_i \in Y$ and $\varphi_i \to \psi_i \in X$ for $i = 1, 2$ and some φ_1, φ_2; then, by antimonotonicity of $\to$ w.r.t. its first argument, $\varphi_i \to \psi_i \leq \varphi_1 \wedge \varphi_2 \to \psi_i \in X$ being X upward closed, and $\varphi_1 \wedge \varphi_2 \to \psi_1 \wedge \psi_2 = (\varphi_1 \wedge \varphi_2 \to \psi_1) \wedge (\varphi_1 \wedge \varphi_2 \to \psi_2) \in X$ since X is closed under intersections.

Concerning continuity, we have to show that $X \cdot Y = \bigsqcup \mathcal{Z}$ where $\mathcal{Z} = \{\uparrow \varphi \cdot \uparrow \psi \mid \varphi \in X \ \ \& \ \ \psi \in Y\}$. Inclusion from left to right follows by observing that if $\chi \in X \cdot Y$ then $\psi \to \chi \in X$ for some $\psi \in Y$, and of course $\chi \in \uparrow (\psi \to \chi) \cdot \uparrow \psi$. Viceversa if $\chi \in \bigsqcup \mathcal{Z}$ then for finitely many ψ_i, χ_i we have that $\bigwedge_i \chi_i \leq \chi$, $\psi_i \in Y$ and $\psi_i \to \chi_i \in X$. It follows that $\chi_i \in X \cdot Y$ for all i, hence the thesis since $X \cdot Y$ is a filter by the above. □

Lemma 13.2.8. *If $f : \mathcal{F}_A \to \mathcal{F}_B$ is continuous then $\Lambda(f) \in \mathcal{F}_{A\to B}$. $\Lambda(\cdot)$ is itself continuous and such that:*

$$\Lambda(f) \cdot X = f(X) \qquad \Lambda(\lambda\!\!\!\lambda Y.(X \cdot Y)) = X$$

Proof. Easy by unfolding definitions and by Lemma 13.2.7. □

Now we can establish:

Proposition 13.2.9. *The following are isomorphisms in ω-**ALG**:*

$$\begin{array}{cc} \mathcal{F}_{A_\perp} \cong (\mathcal{F}_A)_\perp, & \mathcal{F}_{A\times B} \cong \mathcal{F}_A \times \mathcal{F}_B, \\ \mathcal{F}_{A\to B} \cong [\mathcal{F}_A \to \mathcal{F}_B], & \mathcal{F}_{A^{\mathbf{L}}} \cong (\mathcal{F}_A)^{\mathbf{L}}. \end{array}$$

Proof. That $\mathcal{F}_{A_\perp} \cong (\mathcal{F}_A)_\perp$ is a consequence of the fact that $\omega_A <_{A_\perp} \omega_{A_\perp}$ is strict, hence $\uparrow \omega_{A_\perp}$ is the new bottom added to $\mathcal{F}_A$. $\mathcal{F}_{A\times B} \cong \mathcal{F}_A \times \mathcal{F}_B$ is induced by the continuous extension of the map $\uparrow (\alpha \times \beta) \mapsto (\uparrow \alpha, \uparrow \beta)$, that is clearly invertible. That $\mathcal{F}_{A\to B} \cong [\mathcal{F}_A \to \mathcal{F}_B]$ is immediate by Lemma 13.2.8.

Finally, to see that $\mathcal{F}_{A^{\mathbf{L}}} \cong (\mathcal{F}_A)^{\mathbf{L}}$ let us define the maps $F \mapsto \varsigma_F$ from $\mathcal{F}_{A^{\mathbf{L}}}$ to $(\mathcal{F}_A)^{\mathbf{L}}$ and $\varsigma \mapsto F_\varsigma$ from $(\mathcal{F}_A)^{\mathbf{L}}$ to $\mathcal{F}_{A^{\mathbf{L}}}$ by

$$\varsigma_F(\ell) = \bigsqcup\{\uparrow\alpha \mid \langle \ell : \alpha\rangle \in F\} = \{\alpha \mid \langle \ell : \alpha\rangle \in F\}$$

and

$$F_\varsigma = \bigsqcup\{\uparrow\langle \ell : \alpha\rangle \mid \alpha \in \varsigma(\ell)\} = \uparrow\{\langle \ell : \alpha\rangle \mid \alpha \in \varsigma(\ell)\}$$

Then it is routine to prove that these maps are morphisms of ω-**ALG** and inverse each other. □

The next step is to apply Proposition 13.2.9 to describe the compact elements of $D \cong \mathcal{F}_D$ and of $S \cong \mathcal{F}_S$ and the (inverse of) their orderings. Alas, this cannot be done directly because of the recursive nature of the Equation (13.5), but it can be obtained by mirroring the inverse limit construction, e.g. along the lines of [ADCH04, AS08]. Although possible in principle, such a construction requires lots of machinery from the theory of the solution of domain equations; instead we follow the shorter path to define the type theories below simply by mutual induction:

Definition 13.2.10. *Recall that $S = (D_\perp)^{\boldsymbol{L}}$ and let us abbreviate $C = (D \times S)_\perp$ and $\mathbb{S}D = S \to C$; then define the following type languages by mutual induction:*

$$\begin{array}{llll}
\mathcal{L}_D : & \delta & ::= & \delta \to \tau \mid \delta \wedge \delta' \mid \omega_D \\
\mathcal{L}_S : & \sigma & ::= & \langle \ell : \delta_\perp\rangle \mid \sigma \wedge \sigma' \mid \omega_S \qquad \delta_\perp \in \mathcal{L}_{D_\perp} \\
\mathcal{L}_C : & \kappa & ::= & \delta \times \sigma \mid \kappa \wedge \kappa' \mid \omega_C \\
\mathcal{L}_{\mathbb{S}D} : & \tau & ::= & \sigma \to \kappa \mid \tau \wedge \tau' \mid \omega_{\mathbb{S}D}
\end{array}$$

Then the respective itt's $Th_D = (\mathcal{L}_D, \leq_D)$, $Th_S = (\mathcal{L}_S, \leq_S)$, $Th_C = (\mathcal{L}_C, \leq_C)$ and $Th_{\mathbb{S}D} = (\mathcal{L}_{\mathbb{S}D}, \leq_{\mathbb{S}D})$ are defined according to Definition 13.2.5.

We assume that $\wedge$ and $\times$ take precedence over $\to$ and that $\to$ associates to the right so that $\delta \to \tau \wedge \tau'$ reads as $\delta \to (\tau \wedge \tau')$ and $\delta' \to \sigma' \to \delta'' \times \sigma''$ reads as $\delta' \to (\sigma' \to (\delta'' \times \sigma''))$.

Remark 13.2.11. We observe that, in the language $\mathcal{L}_S$ we have both types $\langle \ell : \omega_D\rangle$ and $\langle \ell : \omega_{D_\perp}\rangle$, where ω_D and $\omega_{D_\perp}$ are not equated in $Th_{D_\perp}$. Therefore, $\langle \ell : \omega_{D_\perp}\rangle = \omega_S$, but $\langle \ell : \omega_D\rangle$ is strictly smaller than ω_S.

In comparison to the theory Th_S in [dT21b, dT21a] the key difference is that in those papers there was no distinction among ω_D and $\omega_{D_\perp}$ (as the latter is not included in the type syntax) so that the equation $\omega_S = \langle \ell : \omega_D\rangle$ was part of the theory. The present change essentially amounts to drop such an equation.

The following definition will be essential in the technical development:

Definition 13.2.12. *For any $\sigma \in \mathcal{L}_S$ define $dom(\sigma) \subseteq \boldsymbol{L}$ by:*

$$\begin{array}{rcl}
dom(\langle \ell : \delta\rangle) & = & \begin{cases} \{\ell\} & \textit{if } \delta \neq_{D_\perp} \omega_{D_\perp} \\ \emptyset & \textit{otherwise} \end{cases} \\
dom(\sigma \wedge \sigma') & = & dom(\sigma) \cup dom(\sigma') \\
dom(\omega_S) & = & \emptyset
\end{array}$$

Remark 13.2.13. Let us observe some relevant properties of store types.

The set $dom(\sigma)$, which is always finite, is computable because it is decidable whether $\delta \neq_{D_\perp} \omega_{D_\perp}$. Indeed, $\mathcal{L}_D \subset \mathcal{L}_{D_\perp}$ and any type $\delta \in \mathcal{L}_{D_\perp} \setminus \mathcal{L}_D$ is either an intersection of $\omega_{D_\perp}$ or it can be equated to a type $\delta' \wedge \omega_{D_\perp}$, with $\delta' \in \mathcal{L}_D$, that is clearly equal to δ'.

If $\sigma \leq_S \sigma'$ then $dom(\sigma) \supseteq dom(\sigma')$, as a consequence

$$\sigma \leq_S \langle \ell : \omega_D \rangle \Leftrightarrow \ell \in dom(\sigma)$$

In fact, the $\Rightarrow$ part follows since then $dom(\sigma) \supseteq \{\ell\} = dom(\langle \ell : \omega_D \rangle)$. Vice versa, by definition of $dom(\sigma)$ and of $Th_{D_\perp}$, $\sigma = \langle \ell : \delta \rangle \wedge \sigma'$ for some $\sigma' \in \mathcal{L}_S$ and $\delta \in \mathcal{L}_D$. So that, $\langle \ell : \delta \rangle \wedge \sigma' \leq_S \langle \ell : \delta \rangle \leq_S \langle \ell : \omega_D \rangle$.

From the above observations we deduce that $\sigma \leq_S \langle \ell : \omega_D \rangle$ is decidable.

Remark 13.2.14. Observe that the only constants used in Definition 13.2.10 are the ω's; also we have plenty of equivalences $\varphi = \psi$, namely relations $\varphi \leq \psi \leq \varphi$, involving these constants, that are induced by the definition of the itt's above. For example $\delta \to \omega_{\mathbb{S}D} = \omega_D$ is derivable since $\omega_D \leq_D \delta \to \omega_{\mathbb{S}D} \leq_D \omega_D$ are axioms of Th_D; similarly, $\omega_S \to \omega_C = \omega_{\mathbb{S}D}$.

However, none of the theories above is trivial, because of the strict inequalities:

(i) $\langle \ell : \omega_D \rangle <_S \omega_S$, for any $\ell \in \mathbf{L}$;

(ii) $\omega_D \times \omega_S <_S \omega_C$;

(iii) $\omega_S \to \omega_D \times \omega_S <_{\mathbb{S}D} \omega_S \to \omega_C =_{\mathbb{S}D} \omega_{\mathbb{S}D}$.

Therefore, $\uparrow\omega_S \subset \uparrow\bigwedge_{i\in I}\langle \ell_i : \omega_D \rangle$, for any non empty I, and $\uparrow\omega_C \subset \uparrow(\omega_D \times \omega_S)$, that is $\uparrow\omega_C$ corresponds to the new bottom element added to $\mathcal{F}_{D\times S} \cong \mathcal{F}_D \times \mathcal{F}_S$ in $\mathcal{F}_C = \mathcal{F}_{(D\times S)_\perp} \cong (\mathcal{F}_D \times \mathcal{F}_S)_\perp$.

Theorem 13.2.15. *The theories Th_D and Th_S induce the filter domains $\mathcal{F}_D$ and $\mathcal{F}_S$ which satisfy Equation* (13.5).

Proof. By Proposition 13.2.9 we have

$$\mathcal{F}_{D\to\mathbb{S}D} \cong [\mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D}] \cong [\mathcal{F}_D \to [\mathcal{F}_S \to (\mathcal{F}_D \times \mathcal{F}_S)_\perp]]$$

where $\mathcal{F}_S = \mathcal{F}_{(D_\perp)^{\mathbf{L}}}$. Then the thesis follows since $\mathcal{F}_D = \mathcal{F}_{D\to\mathbb{S}D}$ by construction. $\square$

Remark 13.2.16. The choice of not having atomic types in $\mathcal{L}_D$ is minimalistic, and parallels the analogous definition 5.2.1 of [AO93], where the only constant in the domain logic of a lazy λ-model $D \cong [D \to D]_\perp$ is t (true), corresponding to our ω_D, and having $t \not\leq (t \to t)_\perp$ in the theory.

As a more detailed description of $\mathcal{F}_D$ would show, by relating its construction to the solution of Equation (13.2) in Theorem 13.1.4, the reason why the ω's suffice is that $\mathcal{F}_D$ is (isomorphic to) the non trivial initial solution to the domain equation.

Adding atomic types ξ to $\mathcal{L}_D$ also leads to a filter-model $\mathcal{F}_{D'}$ of λ_{imp}, which is however not isomorphic to $[\mathcal{F}_{D'} \to \mathbb{S}(\mathcal{F}_{D'})]$. To restore the desired isomorphism it suffices to add axioms $\xi =_{D'} \omega_{D'} \to \omega_S \to (\xi \times \omega_S)$ for all atomic ξ: these correspond to the axioms $\xi = \omega \to \xi$ in [BCD83], which are responsible of obtaining a "natural equated" solution to the equation $D = [D \to D]$ of Scott's model: see [AS08].

The λ_{imp} filter-model. According to Definition 13.1.5, to show that $\mathcal{F}_D$ is a λ_{imp}-model it remains to see that $\mathcal{F}_{\mathbb{S}D} \cong \mathbb{S}\mathcal{F}_D$ can be endowed with the structure of a monad, which amounts to say that the maps $unit^{\mathcal{F}} : \mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D}$ and $\star^{\mathcal{F}} : \mathcal{F}_{\mathbb{S}D} \times [\mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D}] \to \mathcal{F}_{\mathbb{S}D}$ are definable in such a way to satisfy the monadic laws. This follows by instantiating Definition 13.1.2 to filter domains:

$$unit^{\mathcal{F}} X \overset{def}{=} \Lambda(\lambda\!\!\lambda Y \in \mathcal{F}_S.(X, Y)) = \uparrow\{\sigma \to \delta \times \sigma \in \mathcal{L}_{\mathbb{S}D} \mid \delta \in X\} \tag{13.6}$$

On the other hand, observing that $\mathcal{F}_{\mathbb{S}D} \times [\mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D}] \to \mathcal{F}_{\mathbb{S}D} \cong \mathcal{F}_{\mathbb{S}D} \times \mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D}$, for all $X \in \mathcal{F}_{\mathbb{S}D}$, $Y \in \mathcal{F}_D$ and $Z \in \mathcal{F}_S$ we expect that:

$$\begin{aligned}(X \star^{\mathcal{F}} Y) \cdot Z &= \mathsf{let}\,(U, V) := X \cdot Z \,\mathsf{in}\, (Y \cdot U) \cdot V \\ &= \begin{cases} (Y \cdot U) \cdot V & \text{if } X \cdot Z = U \times V \neq \uparrow_C \omega_C \\ \uparrow_C \omega_C & \text{otherwise} \end{cases}\end{aligned}$$

Hence we define $X \star^{\mathcal{F}} Y$ by

$$\uparrow\{\sigma \to \delta'' \times \sigma'' \in \mathcal{L}_{\mathbb{S}D} \mid \exists \delta', \sigma'.\ \sigma \to \delta' \times \sigma' \in X \ \ \&\ \ \delta' \to \sigma' \to \delta'' \times \sigma'' \in Y\} \tag{13.7}$$

Lemma 13.2.17. *Both $unit^{\mathcal{F}} X$ and $X \star^{\mathcal{F}} Y$ are continuous in X and in X and Y, respectively.*

Proof. By definition, $unit^{\mathcal{F}}(\uparrow \delta) = \uparrow \{\sigma \to \delta' \times \sigma \mid \sigma \in \mathcal{L}_S \ \&\ \delta' \in \uparrow\delta\}$; on the other hand if $\delta_0 \leq_D \delta_1$ then $\uparrow\delta_1 \subseteq \uparrow\delta_0$, therefore

$$\begin{array}{rcl} unit^{\mathcal{F}}(\uparrow\delta_1) &=& \uparrow\{\sigma \to \delta' \times \sigma \mid \sigma \in \mathcal{L}_S \ \&\ \delta' \in \uparrow\delta_1\} \\ &\subseteq& \uparrow\{\sigma \to \delta'' \times \sigma \mid \sigma \in \mathcal{L}_S \ \&\ \delta'' \in \uparrow\delta_0\} \\ &=& unit^{\mathcal{F}}(\uparrow\delta_0) \end{array}$$

Hence $unit^{\mathcal{F}}(\uparrow\delta) \subseteq unit^{\mathcal{F}}(\uparrow(\delta \wedge \delta')) \supseteq unit^{\mathcal{F}}(\uparrow\delta')$ for all δ, δ'; this implies that the family $\{unit^{\mathcal{F}}(\uparrow\delta) \mid \delta \in X\}$ is directed for any filter X. Now

$$\begin{array}{rcll} unit^{\mathcal{F}} X &=& unit^{\mathcal{F}}(\bigcup_{\delta\in X} \uparrow\delta) & \text{as } X = \bigsqcup^{\uparrow}_{\delta\in X} \uparrow\delta = \bigcup_{\delta\in X} \uparrow\delta \\ &=& \uparrow\{\sigma \to \delta' \times \sigma \mid \sigma \in \mathcal{L}_S \ \&\ \delta' \in \bigcup_{\delta\in X} \uparrow\delta\} & \text{by def. of } unit^{\mathcal{F}} \\ &=& \bigcup_{\delta\in X} \uparrow\{\sigma \to \delta' \times \sigma \mid \sigma \in \mathcal{L}_S \ \&\ \delta' \in \uparrow\delta\} & \text{as } \{\uparrow\delta \mid \delta \in X\} \text{ is directed} \\ &=& \bigcup_{\delta\in X} unit^{\mathcal{F}}(\uparrow\delta) & \text{by def. of } unit^{\mathcal{F}} \\ &=& \bigsqcup^{\uparrow}_{\delta\in X} unit^{\mathcal{F}}(\uparrow\delta) & \text{by Lemma 13.2.3.2} \\ &&& \text{and the above remark} \end{array}$$

The proof of the continuity of $X \star^{\mathcal{F}} Y$ is similar. □

The final step is to define the interpretations of $get_\ell : (\mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D}) \to \mathcal{F}_{\mathbb{S}D} \cong \mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D}$ and $set_\ell : \mathcal{F}_D \times \mathcal{F}_{\mathbb{S}D} \to \mathcal{F}_{\mathbb{S}D}$, which also are derivable from their interpretation into a generic model $\mathcal{D}$.

Definition 13.2.18. *Let $X \in \mathcal{F}_S$, $\ell \in \boldsymbol{L}$:*

$$X \cdot \{\ell\} = \uparrow\{\delta \in \mathcal{L}_D \mid \langle \ell : \delta \rangle \in X\}$$

The above operation represents the application of the “store” X to the location ℓ. Observe that if $X = \uparrow\omega_S$ then there is no $\langle \ell : \delta \rangle \in X$, hence the closure $\uparrow\cdot$ is necessary.

According to Definition 13.1.5, for any $X \in \mathcal{F}_D$ and $Y \in \mathcal{F}_S$ we must have:

$$get_\ell^{\mathcal{F}}(X) \cdot Y = X \cdot (Y \cdot \{\ell\}) \cdot Y$$

where we assume that $\_ \cdot \_$ associates to the left. Now, let $Z = \{\tau \in \mathcal{L}_{\mathbb{S}D} \mid \exists \delta \in \langle \ell : \delta \rangle \in Y \;\;\&\;\; \delta \to \tau \in X\}$ then

$$X \cdot (Y \cdot \{\ell\}) \cdot Y = Z \cdot Y$$

If $\tau = \omega_{\mathbb{S}D}$ then $\delta \to \tau = \omega_D$, which trivially belongs to any filter; if instead $\tau \neq \omega_{\mathbb{S}D}$ then $\tau = \bigwedge_I \sigma_i \to \kappa_i$ and $\delta \to \tau = \bigwedge_{i \in I} \delta \to \sigma_i \to \kappa_i \in X$ if and only if $\delta \to \sigma_i \to \kappa_i \in X$ for all $i \in I$. From this we conclude that

$$\begin{array}{rcl} Z \cdot Y & = & \{\kappa \in \mathcal{L}_C \mid \exists \sigma \in Y \mid \sigma \to \kappa \in Z\} \\ & = & \{\kappa \in \mathcal{L}_C \mid \exists \langle \ell : \delta \rangle \wedge \sigma \in Y.\, \delta \to \sigma \to \kappa \in X\} \end{array}$$

and therefore the appropriate definition of $get_\ell^{\mathcal{F}}(X)$ is

$$get_\ell^{\mathcal{F}}(X) \overset{def}{=} \uparrow\{(\langle \ell : \delta \rangle \wedge \sigma) \to \kappa \in \mathcal{L}_{\mathbb{S}D} \mid \delta \to (\sigma \to \kappa) \in X\} \tag{13.8}$$

Similarly, again by Definition 13.1.5, for any $X \in \mathcal{F}_D$, $Y \in \mathcal{F}_{\mathbb{S}D}$ and $Z \in \mathcal{F}_S$ we expect:

$$set_\ell^{\mathcal{F}}(X, Y) \cdot Z = Y \cdot (Z[\ell \mapsto X])$$

where $Z[\ell \mapsto X]$ is supposed to represent the update of Z by associating X to ℓ, namely:

$$Z[\ell \mapsto X] = \uparrow\{\langle \ell : \delta \rangle \mid \delta \in X\} \cup \{\langle \ell' : \delta' \rangle \mid \langle \ell' : \delta' \rangle \in Z \;\;\&\;\; \ell' \neq \ell\}$$

Then

$$\begin{array}{rcl} Y \cdot (Z[\ell \mapsto X]) & = & \uparrow\{\kappa \mid \exists \sigma \to \kappa \in Z[\ell \mapsto X].\, \sigma \to \kappa \in Y\} \\ & = & \uparrow\{\kappa \mid \exists \sigma' \in Z, \delta \in X.\, \langle \ell : \delta \rangle \wedge \sigma' \to \kappa \in Y \;\;\&\;\; \ell \notin dom(\sigma')\} \end{array}$$

and therefore we define:

$$set_\ell^{\mathcal{F}}(X, Y) \overset{def}{=} \uparrow\{\sigma' \to \kappa \in \mathcal{L}_{\mathbb{S}D} \mid \exists \delta \in X.\, \langle \ell : \delta \rangle \wedge \sigma' \to \kappa \in Y \;\;\&\;\; \ell \notin dom(\sigma')\} \tag{13.9}$$

Lemma 13.2.19. *Both $get_\ell^{\mathcal{F}}(X)$ and $set_\ell^{\mathcal{F}}(X, Y)$ are continuous in X and in X and Y, respectively.*

Proof. We know that $get^{\mathcal{F}}_{\ell}(X) \cdot Y = X \cdot (Y \cdot \{\ell\}) \cdot Y$, hence by Lemma 13.2.7 to prove that it is continuous in X it suffices to show that $Y \cdot \{\ell\}$ is such in Y, as the composition of continuous functions is continuous. Now

$$\begin{array}{rcl} Y \cdot \{\ell\} & = & \uparrow\{\delta \mid \langle \ell : \delta \rangle \in Y\} \\ & = & \uparrow\{\delta \mid \langle \ell : \delta \rangle \in \bigcup_{\sigma \in Y} \uparrow\sigma\} \\ & = & \bigcup_{\sigma \in Y} \uparrow\{\delta \mid \langle \ell : \delta \rangle \in \uparrow\sigma\} \\ & = & \bigcup_{\sigma \in Y} (\uparrow\sigma \cdot \{\ell\}) \\ & = & \bigsqcup^{\uparrow}_{\sigma \in Y} (\uparrow\sigma \cdot \{\ell\}) \end{array}$$

by directness of $\{\uparrow\sigma \cdot \{\ell\} \mid \sigma \in Y\}$ and Lemma 13.2.3.

The proof of continuity of $set^{\mathcal{F}}_{\ell}(X, Y)$ is similar, using $set^{\mathcal{F}}_{\ell}(X, Y) \cdot Z = Y \cdot (Z[\ell \mapsto X])$, the continuity of application and the fact that $Z[\ell \mapsto X] = \bigsqcup_{\delta \in X} Z[\ell \mapsto \uparrow\delta]$ that is easily seen. □

Eventually we have:

Theorem 13.2.20. *The structure $\mathcal{F} = (\mathcal{F}_D, \mathbb{S}, [\![\cdot]\!]^{\mathcal{F}_D}, [\![\cdot]\!]^{\mathcal{F}_{\mathbb{S}D}})$ is a λ_{imp}-model where, for $e : Var \to \mathcal{F}_D$ the interpretations $[\![V]\!]^{\mathcal{F}_D} e \in \mathcal{F}_D$ and $[\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e \in \mathcal{F}_{\mathbb{S}D}$ are such that:*

$$\begin{array}{rcl} [\![x]\!]^{\mathcal{F}_D} e & = & e(x) \\ [\![\lambda x.M]\!]^{\mathcal{F}_D} e & = & \uparrow\{\delta \to \sigma \mid \sigma \in [\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e[x \mapsto \uparrow\delta]\} \\ [\![[V]]\!]^{\mathcal{F}_{\mathbb{S}D}} e & = & \uparrow\{\sigma \to \delta \times \sigma \in \mathcal{L}_{\mathbb{S}D} \mid \delta \in [\![V]\!]^{\mathcal{F}_D} e\} \\ [\![M \star V]\!]^{\mathcal{F}_{\mathbb{S}D}} e & = & \uparrow\{\sigma \to \delta'' \times \sigma'' \mid \exists \delta', \sigma'.\ \sigma \to \delta' \times \sigma' \in [\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e \\ & & \qquad \& \ \ \delta' \to \sigma' \to \delta'' \times \sigma'' \in [\![V]\!]^{\mathcal{F}_D} e\} \\ [\![get_{\ell}(\lambda x.M)]\!]^{\mathcal{F}_{\mathbb{S}D}} e & = & \uparrow\{(\langle \ell : \delta \rangle \wedge \sigma) \to \kappa \in \mathcal{L}_{\mathbb{S}D} \mid \delta \to (\sigma \to \kappa) \in [\![\lambda x.M]\!]^{\mathcal{F}_D} e\} \\ [\![set_{\ell}(V, M)]\!]^{\mathcal{F}_{\mathbb{S}D}} e & = & \uparrow\{\sigma' \to \kappa \mid \exists \delta \in [\![V]\!]^{\mathcal{F}_D} e.\ \langle \ell : \delta \rangle \wedge \sigma' \to \kappa \in [\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e \\ & & \qquad \& \ \ \ell \notin dom(\sigma')\} \end{array}$$

Proof. By an easy induction over M, we can show that

$$[\![M]\!]^{\mathcal{F}} e[x \mapsto X] = \bigsqcup^{\uparrow}_{\delta \in X} [\![M]\!]^{\mathcal{F}} e[x \mapsto \uparrow\delta]$$

It follows that the function $\lambda\!\!\lambda X \in \mathcal{F}_D.\ [\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e[x \mapsto X]$ is continuous. Therefore, applying the clauses of Definition 13.1.5 to the model $\mathcal{F}$, we get:

$$\begin{array}{rcl} [\![x]\!]^{\mathcal{F}_D} e & = & e(x) \\ [\![\lambda x.M]\!]^{\mathcal{F}_D} e & = & \Lambda(\lambda\!\!\lambda X \in \mathcal{F}_D.\ [\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e[x \mapsto X]) \\ [\![[V]]\!]^{\mathcal{F}_{\mathbb{S}D}} e & = & unit^{\mathcal{F}}([\![V]\!]^{\mathcal{F}_D} e) \\ [\![M \star V]\!]^{\mathcal{F}_{\mathbb{S}D}} e & = & ([\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e) \star^{\mathcal{F}} ([\![V]\!]^{\mathcal{F}_D} e) \\ [\![get_{\ell}(\lambda x.M)]\!]^{\mathcal{F}_{\mathbb{S}D}} e & = & get^{\mathcal{F}}_{\ell}([\![\lambda x.M]\!]^{\mathcal{F}_D} e) \\ [\![set_{\ell}(V, M)]\!]^{\mathcal{F}_{\mathbb{S}D}} e & = & set^{\mathcal{F}}_{\ell}([\![V]\!]^{\mathcal{F}_D} e, [\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e) \end{array}$$

Then the thesis follows by equations (13.6)-(13.9). □

13.3 Deriving the Type Assignment System

The definition of a type assignment system can be obtained out of the construction of the filter-model. The system is nothing else than the description of the denotation of terms in the particular λ_{imp}-model $\mathcal{F}$, which is possible because both $[\![V]\!]^{\mathcal{F}_D} e \in \mathcal{F}_D$ and $[\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e \in \mathcal{F}_{\mathbb{S}D}$, and filters are sets of types.

Types are naturally interpreted as subsets of the domains of term interpretation, namely $\mathcal{F}_D$ or $\mathcal{F}_{\mathbb{S}D}$ according to their kind; by applying to the present case the same construction as in [BCD83] and [Abr91], originating from Stone duality for boolean algebras, we set:

Definition 13.3.1. *For $A = D, S, C$, and $\mathbb{S}D$, and $\varphi \in \mathcal{L}_A$ define:*

$$[\![\varphi]\!]^{\mathcal{F}} = \{X \in \mathcal{F}_A \mid \varphi \in X\}$$

Such interpretation is a generalization of the natural interpretation of intersection types over an extended type structure [CDHL84], which in the present case yields:

Proposition 13.3.2. *For $A = D, S, C$, and $\mathbb{S}D$, and $\varphi, \psi \in \mathcal{L}_A$ we have:*

$$[\![\varphi \wedge \psi]\!]^{\mathcal{F}} = [\![\varphi]\!]^{\mathcal{F}} \cap [\![\psi]\!]^{\mathcal{F}}, \qquad [\![\omega_A]\!]^{\mathcal{F}} = \mathcal{F}_A \quad \textit{and} \quad \varphi \leq \psi \Rightarrow [\![\varphi]\!]^{\mathcal{F}} \subseteq [\![\psi]\!]^{\mathcal{F}}$$

Moreover:

1. $[\![\delta \to \tau]\!]^{\mathcal{F}} = \{X \in \mathcal{F}_D \mid \forall Y \in [\![\delta]\!]^{\mathcal{F}}.\ X \cdot Y \in [\![\tau]\!]^{\mathcal{F}}\}$
2. $[\![\langle \ell : \delta \rangle]\!]^{\mathcal{F}} = \{X \in \mathcal{F}_S \mid X \cdot \{\ell\} \in [\![\delta]\!]^{\mathcal{F}}\}$
3. $[\![\delta \times \sigma]\!]^{\mathcal{F}} = \{X \in \mathcal{F}_{D \times S} \mid \pi_1(X) \in [\![\delta]\!]^{\mathcal{F}}\ \&\ \pi_2(X) \in [\![\sigma]\!]^{\mathcal{F}}\}$
4. $[\![\sigma \to \kappa]\!]^{\mathcal{F}} = \{X \in \mathcal{F}_{\mathbb{S}D} \mid \forall Y \in [\![\sigma]\!]^{\mathcal{F}}.\ X \cdot Y \in [\![\kappa]\!]^{\mathcal{F}}\}$

where, for $X \in \mathcal{F}_{D \times S} \cong (\mathcal{F}_D \times \mathcal{F}_S)$, $\pi_1(X) = \{\delta \in \mathcal{L}_D \mid \exists \sigma \in \mathcal{L}_S.\ \delta \times \sigma \in X\}$ and similarly for $\pi_2(X)$.

Proof. The first part is immediate from the definition of type interpretation and of filters.

To see (1) observe that $\uparrow\varphi \in [\![\varphi]\!]^{\mathcal{F}}$. Hence, if $X \in [\![\delta \to \tau]\!]^{\mathcal{F}}$ then $\delta \to \tau \in X$, which implies that $X \cdot \uparrow\delta = \uparrow\tau \in [\![\tau]\!]^{\mathcal{F}}$. Viceversa if $X \cdot Y \in [\![\tau]\!]^{\mathcal{F}}$ for all $Y \in [\![\delta]\!]^{\mathcal{F}}$ then in particular $X \cdot \uparrow\delta = \uparrow\tau \in [\![\tau]\!]^{\mathcal{F}}$, which implies that for some $\delta' \in \uparrow\delta$, $\delta' \to \tau \in X$; but then $\delta \leq_D \delta'$ so that $\delta' \to \tau \leq_D \delta \to \tau$ and therefore $\delta \to \tau \in X$ as X is upward closed.

All the other cases are similar and easier. We just remark that in part (3) $\pi_1(X)$ is a filter: indeed it is nonempty and upward closed because X is such. If $\delta_1, \delta_2 \in \pi_1(X)$ then $\delta_1 \times \sigma_1, \delta_2 \times \sigma_2 \in X$ for some σ_1 and σ_2; then $X \ni (\delta_1 \times \sigma_1) \wedge (\delta_2 \times \sigma_2) = (\delta_1 \wedge \delta_2) \times (\sigma_1 \wedge \sigma_2)$, so that $\delta_1 \wedge \delta_2 \in \pi_1(X)$. □

Let us define the notion of *typing context* in our setting.

Definition 13.3.3. *A* typing context Γ *is a finite set $\{x_1 : \delta_1, \ldots, x_n : \delta_n\}$ where $n \geq 0$, the x_i's are pairwise distinct, and $\delta_i \in \mathcal{L}_D$ for all i.*

$$\frac{}{\Gamma, x:\delta \vdash x:\delta}\,(var) \qquad \frac{\Gamma, x:\delta \vdash M:\tau}{\Gamma \vdash \lambda x.M : \delta \to \tau}\,(\lambda)$$

$$\frac{\Gamma \vdash V:\delta}{\Gamma \vdash [V] : \sigma \to \delta \times \sigma}\,(unit) \qquad \frac{\Gamma \vdash M : \sigma \to \delta' \times \sigma' \quad \Gamma \vdash V : \delta' \to \sigma' \to \delta'' \times \sigma''}{\Gamma \vdash M \star V : \sigma \to \delta'' \times \sigma''}\,(\star)$$

$$\frac{\Gamma, x:\delta \vdash M : \sigma \to \kappa}{\Gamma \vdash get_\ell(\lambda x.M) : (\langle \ell : \delta\rangle \wedge \sigma) \to \kappa}\,(get) \qquad \frac{\Gamma \vdash V:\delta \quad \Gamma \vdash M : (\langle \ell : \delta\rangle \wedge \sigma) \to \kappa \quad \ell \notin dom(\sigma)}{\Gamma \vdash set_\ell(V, M) : \sigma \to \kappa}\,(set)$$

Figure 13.1: Intersection type assignment system for λ_{imp}

We can now build the type assignment system as the description, via type derivations, of the semantics of terms and types in the model $\mathcal{F}$. First, type contexts $\Gamma = \{x_1 : \delta_1, \dots, x_n : \delta_n\}$ are put into correspondence with environments $e_\Gamma : Var \to \mathcal{F}_D$ by setting $e_\Gamma(x_i) = \uparrow_D \delta_i$ and $e_\Gamma(y) = \uparrow \omega_D$ if $y \notin dom(\Gamma)$. Second, typing judgments translate the statements:

$$\Gamma \vdash V : \delta \iff ([\![V]\!]^{\mathcal{F}_D} e_\Gamma) \in [\![\delta]\!]^{\mathcal{F}} \iff \delta \in [\![V]\!]^{\mathcal{F}_D} e_\Gamma$$

and similarly, $\Gamma \vdash M : \tau \iff \tau \in [\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e_\Gamma$.

Third and final step, typing rules of the shape

$$\frac{\Gamma_1 \vdash P_1 : \varphi_1 \quad \cdots \quad \Gamma_n \vdash P_n : \varphi_n}{\Gamma \vdash Q : \psi}$$

are the translations of equations

$$[\![Q]\!]^{\mathcal{F}} e_\Gamma = \uparrow \{\psi \mid \varphi_1 \in [\![P_1]\!]^{\mathcal{F}} e_{\Gamma_1} \ \& \ \cdots \ \& \ \varphi_n \in [\![P_n]\!]^{\mathcal{F}} e_{\Gamma_n}\} \tag{13.10}$$

from Proposition 13.3.2 and Theorem 13.2.20. We take advantage of the factorization of the right hand sides of Equations (13.10) into a set of types U and its closure $\uparrow U$, by adding the rules:

$$\frac{}{\Gamma \vdash P : \omega}\,(\omega) \qquad \frac{\Gamma \vdash P : \varphi \quad \Gamma \vdash P : \psi}{\Gamma \vdash P : \varphi \wedge \psi}\,(\wedge) \qquad \frac{\Gamma \vdash P : \varphi \quad \varphi \le \psi}{\Gamma \vdash P : \psi}\,(\le) \tag{13.11}$$

Therefore, we are left to translate set inclusions

$$[\![Q]\!]^{\mathcal{F}} e_\Gamma \supseteq \{\psi \mid \varphi_1 \in [\![P_1]\!]^{\mathcal{F}} e_{\Gamma_1} \ \& \ \cdots \ \& \ \varphi_n \in [\![P_n]\!]^{\mathcal{F}} e_{\Gamma_n}\} \tag{13.12}$$

where ψ is neither an intersection nor an ω:

Definition 13.3.4. *(***Intersection type assignment system for** λ_{imp}*) The system is obtained by adding to the rules in Equation* (13.11) *the rules in Figure 13.1.*

The rules in Definition 13.3.4 are obtained by instantiating the inclusion (13.12) to Equations (13.6) - (13.9). By such a construction, we obtain for our type system the analogous of the "Type-semantics theorem" for intersection types and the ordinary λ-calculus (see [BDS13], Thm. 16.2.7). Toward such a theorem we define the following concept:

Definition 13.3.5. *If $e \in$ Term-Env$_{\mathcal{F}}$ then for any type context Γ:*

$$e \models^{\mathcal{F}} \Gamma \Leftrightarrow \forall x : \delta \in \Gamma.\ e(x) \subseteq [\![\delta]\!]^{\mathcal{F}}$$

It is immediate by Definition 13.3.1 that $e \models^{\mathcal{F}} \Gamma$ if and only if $\delta \in e(x)$ for all $x : \delta \in \Gamma$. On the other hand, from the very construction of the type system, we have:

Lemma 13.3.6. *Let $Q \in Term$ and Γ be any typing context, then*

$$[\![Q]\!]^{\mathcal{F}} e_\Gamma = \{\varphi \mid \Gamma \vdash Q : \varphi\}$$

Proof. The proof is based on Theorem 13.2.20 that characterizes $[\![Q]\!]^{\mathcal{F}}_{e_\Gamma}$ in terms of operations over filters: it will be used without explicit mention below.

The inclusion $[\![Q]\!]^{\mathcal{F}}_{e_\Gamma} \subseteq \{\varphi \mid \Gamma \vdash Q : \varphi\}$ is shown by induction over Q.

Case $Q \equiv x$: then $[\![x]\!]^{\mathcal{F}}_{e_\Gamma} = e_\Gamma(x)$. If $x : \delta \in \Gamma$ for some δ then $e_\Gamma(x) = \uparrow\delta$ and $\Gamma \vdash x : \delta'$ for all $\delta' \geq \delta$ by rules (*var*) and ($\leq$). Otherwise, $x \notin dom(\Gamma)$ implies that $e_\Gamma(x) = \uparrow\omega_D$ so that $\Gamma \vdash x : \delta$ for any $\delta = \omega_D$ by rules (ω) and ($\leq$).

Case $Q \equiv \lambda x.M$: then by Definition 13.2.6 we have

$$\begin{array}{rcl} [\![\lambda x.M]\!]^{\mathcal{F}} e_\Gamma & = & \Lambda(\lambda\!\!\!\lambda X \in \mathcal{F}_D.\ [\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e_\Gamma[x \mapsto X]) \\ & = & \uparrow\{\delta \to \tau \mid \tau \in [\![M]\!]^{\mathcal{F}} e_\Gamma[x \mapsto \uparrow\delta]\} \end{array}$$

By definition of the $\uparrow\_$ operator, the filter above consists of intersections $\bigwedge_{i\in I} \delta_i \to \tau_i$ such that $\tau_i \in [\![M]\!]^{\mathcal{F}} e_\Gamma[x \mapsto \uparrow\delta_i]$ for all $i \in I$. Observing that $e_\Gamma[x \mapsto \uparrow\delta_i] = e_{\Gamma,x:\delta_i}$ we have by induction that $\Gamma, x : \delta_i \vdash M : \tau_i$ and therefore $\Gamma \vdash \lambda x.M : \delta_i \to \tau_i$ by rule (λ) for all $i \in I$. Hence $\Gamma \vdash \lambda x.M : \bigwedge_{i\in I} \delta_i \to \tau_i$ by repeated use of rule ($\wedge$).

Case $Q \equiv [V]$: then $[\![[V]]\!]^{\mathcal{F}} e_\Gamma = \uparrow\{\sigma \to \delta \times \sigma \mid \delta \in [\![V]\!]^{\mathcal{F}} e_\Gamma\}$ by Equation (13.6). By induction we have $\Gamma \vdash V : \delta$ and the thesis follows by rule (*unit*).

Case $Q \equiv M \star V$: then, by Equation (13.7), $[\![M \star V]\!]^{\mathcal{F}} e_\Gamma$ is the filter

$$\uparrow\{\sigma \to \delta'' \times \sigma'' \mid \exists \delta', \sigma'.\ \sigma \to \delta' \times \sigma' \in [\![M]\!]^{\mathcal{F}} e_\Gamma\ \ \&\ \ \delta' \to \sigma' \to \delta'' \times \sigma'' \in [\![V]\!]^{\mathcal{F}} e_\Gamma\}$$

As in the case of λ-abstraction, the types in such a filter have the shape $\bigwedge_{i\in I} \sigma_i \to \delta''_i \times \sigma''_i$ and $\sigma_i \to \delta'_i \times \sigma'_i \in [\![M]\!]^{\mathcal{F}} e_\Gamma$ and $\delta'_i \to \sigma'_i \to \delta''_i \times \sigma''_i \in [\![V]\!]^{\mathcal{F}} e_\Gamma$ for all $i \in I$. Therefore by rule ($\star$) we have
$\Gamma \vdash M \star V : \sigma_i \to \delta''_i \times \sigma''_i$ for all $i \in I$ and we conclude by repeated use of rule ($\wedge$).

Case $Q \equiv get_\ell(\lambda x.M)$: then by Equation (13.8) we have that $[\![get_\ell(\lambda x.M)]\!]^{\mathcal{F}} e_\Gamma$ is the filter

$$\uparrow\{(\langle \ell : \delta\rangle \wedge \sigma) \to \kappa \mid \delta \to (\sigma \to \kappa) \in [\![\lambda x.M]\!]^{\mathcal{F}} e_\Gamma\}$$

Reasoning as above we know that $\delta \to (\sigma \to \kappa) \in [\![\lambda x.M]\!]^{\mathcal{F}} e_\Gamma$ implies that $\sigma \to \kappa \in [\![M]\!]^{\mathcal{F}} e_{\Gamma,x:\delta}$; hence by induction $\Gamma, x : \delta \vdash M : \sigma \to \kappa$, from which the thesis follows by rule (*get*).

Case $Q \equiv set_\ell(V, M)$: then by Equation (13.9) we know that $[\![set_\ell(V, M)]\!]^{\mathcal{F}} e_\Gamma$ is the filter

$$\uparrow \{\sigma' \to \kappa \mid \exists \delta \in [\![V]\!]^{\mathcal{F}} e_\Gamma.\ \langle \ell : \delta \rangle \wedge \sigma' \to \kappa \in [\![M]\!]^{\mathcal{F}} e_\Gamma \quad \& \quad \ell \notin dom(\sigma')\}$$

By induction $\Gamma \vdash V : \delta$ and $\Gamma \vdash M : \langle \ell : \delta \rangle \wedge \sigma' \to \kappa$; now condition $\ell \notin dom(\sigma')$ allows to apply rule (set), and we are done.

The inclusion $[\![Q]\!]^{\mathcal{F}}_{e_\Gamma} \supseteq \{\varphi \mid \Gamma \vdash Q : \varphi\}$ is proved by induction over the derivation of $\Gamma \vdash Q : \varphi$. The cases of rules (ω), $(\wedge)$ and $(\leq)$ are immediate by the induction hypothesis and the fact that $[\![Q]\!]^{\mathcal{F}}_{e_\Gamma}$ is a filter.

Case (var): then $Q \equiv x$, $\varphi \equiv \delta$ and $x : \delta \in \Gamma$; hence

$$[\![x]\!]^{\mathcal{F}} e_\Gamma = e_\Gamma(x) = \uparrow \delta \ni \delta$$

Case (λ): then $Q \equiv \lambda x.M$, $\varphi \equiv \delta \to \tau$ and the premise is $\Gamma, x : \delta \vdash M : \tau$. By induction $\tau \in [\![M]\!]^{\mathcal{F}} e_{\Gamma, x:\delta}$; on the other hand, as observed above, $[\![\lambda x.M]\!]^{\mathcal{F}} e_\Gamma = \uparrow \{\delta \to \tau \mid \tau \in [\![M]\!]^{\mathcal{F}} e_\Gamma[x \mapsto \uparrow \delta]\}$, and $e_\Gamma[x \mapsto \uparrow \delta] = e_{\Gamma, x:\delta}$, hence we conclude that $\delta \to \tau \in [\![\lambda x.M]\!]^{\mathcal{F}} e_\Gamma$.

Case $(unit)$: then $Q \equiv [V]$, $\varphi \equiv \sigma \to \delta \times \sigma$ and the premise is $\Gamma \vdash V : \delta$. By induction $\delta \in [\![V]\!]^{\mathcal{F}} e_\Gamma$ hence $\sigma \to \delta \times \sigma \in unit^{\mathcal{F}} [\![V]\!]^{\mathcal{F}} e_\Gamma = [\![[V]]\!]^{\mathcal{F}} e_\Gamma$ by Equation (13.6).

Case $(\star)$: then $Q \equiv M \star V$, $\varphi \equiv \sigma \to \delta'' \times \sigma''$ which is derived from the premises $\Gamma \vdash M : \sigma \to \delta' \times \sigma'$ and $\Gamma \vdash V : \delta' \to \sigma' \to \delta'' \times \sigma''$. By induction $\sigma \to \delta' \times \sigma' \in [\![M]\!]^{\mathcal{F}} e_\Gamma$ and $\delta' \to \sigma' \to \delta'' \times \sigma'' \in [\![V]\!]^{\mathcal{F}} e_\Gamma$ so that $\sigma \to \delta'' \times \sigma'' \in [\![M]\!]^{\mathcal{F}} e_\Gamma \star^{\mathcal{F}} [\![V]\!]^{\mathcal{F}} e_\Gamma = [\![M \star V]\!]^{\mathcal{F}} e_\Gamma$ by Equation (13.7).

Case (get): then $Q \equiv get_\ell(\lambda x.M)$, $\varphi \equiv (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa$ and the premise is $\Gamma, x : \delta \vdash M : \sigma \to \kappa$. By induction $\sigma \to \kappa \in [\![M]\!]^{\mathcal{F}} e_{\Gamma, x:\delta}$ so that, by reasoning as in case (λ), we have that $\delta \to \sigma \to \kappa \in [\![\lambda x.M]\!]^{\mathcal{F}} e_\Gamma$, from which we conclude $(\langle \ell : \delta \rangle \wedge \sigma) \to \kappa \in get^{\mathcal{F}}_\ell([\![\lambda x.M]\!]^{\mathcal{F}} e_\Gamma) = [\![get_\ell(\lambda x.M)]\!]^{\mathcal{F}} e_\Gamma$ by Equation (13.8).

Case (set): then $Q \equiv set_\ell(V, M)$, $\varphi \equiv \sigma \to \kappa$ and the premises are $\Gamma \vdash V : \delta$ and $\Gamma \vdash M : (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa$; also we know that $\ell \notin dom(\sigma)$. By induction $\delta \in [\![V]\!]^{\mathcal{F}} e_\Gamma$ and $(\langle \ell : \delta \rangle \wedge \sigma) \to \kappa \in [\![M]\!]^{\mathcal{F}} e_\Gamma$. Then we conclude that $\varphi \equiv \sigma \to \kappa \in set^{\mathcal{F}}_\ell([\![V]\!]^{\mathcal{F}_D} e_\Gamma, [\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e_\Gamma) = [\![set_\ell(V, M)]\!]^{\mathcal{F}_{\mathbb{S}D}} e_\Gamma$ by Equation (13.9).

□

Definition 13.3.7. *Over $TypeEnv^{\mathcal{F}}$ we define the ordering:*

$$e \sqsubseteq e' \Leftrightarrow \forall x \in Var.\ e(x) \subseteq e'(x)$$

The poset $(TypeEnv^{\mathcal{F}}, \sqsubseteq)$ is a cpo, with bottom $e_\bot$ such that $e_\bot(x) = \uparrow \omega_D$ for all $x \in Var$. We extend the definition of directed sups in $\mathcal{F}_D$ to directed families $\mathcal{E} \subseteq TypeEnv^{\mathcal{F}}$, putting:

$$\forall x \in Var.\ (\bigsqcup\nolimits^{\uparrow} \mathcal{E})(x) \overset{def}{=} \bigcup_{e \in \mathcal{E}} e(x)$$

Handy notations in the next lemmas are: $\Gamma(x) = \delta$ if $x : \delta \in \Gamma$, $\Gamma(x) = \omega_D$ otherwise; and for contexts Γ and Γ' we define the context

$$\begin{array}{rcl} \Gamma \wedge \Gamma' & \overset{def}{=} & \{x : \delta \in \Gamma \mid x \notin \mathit{dom}(\Gamma')\} \\ & \cup & \{x : \delta' \in \Gamma' \mid x \notin \mathit{dom}(\Gamma)\} \\ & \cup & \{x : \delta \wedge \delta' \mid x : \delta \in \Gamma \ \ \& \ \ x : \delta' \in \Gamma'\} \end{array}$$

As a consequence $(\Gamma \wedge \Gamma')(x) = \Gamma(x) \wedge \Gamma'(x)$.

Lemma 13.3.8.

1. *For all $e \in \mathit{TypeEnv}^{\mathcal{F}}$ the set $\mathcal{E} = \{e_\Gamma \mid e \models^{\mathcal{F}} \Gamma\}$ is directed and $e = \bigsqcup^{\uparrow} \mathcal{E}$*

2. *For all $Q \in \mathit{Term}$ and directed $\mathcal{E}$, the family of filters $\{[\![Q]\!]^{\mathcal{F}} e \mid e \in \mathcal{E}\}$ is directed and*
$$[\![Q]\!]^{\mathcal{F}}(\bigsqcup^{\uparrow}\mathcal{E}) = \bigsqcup^{\uparrow}_{e\in\mathcal{E}}[\![Q]\!]^{\mathcal{F}} e$$

Proof. Part (1): by Definition 13.3.5 and the definition of e_Γ we have that

$$e \models^{\mathcal{F}} \Gamma \Leftrightarrow \forall x \in \mathit{Var}.\, \Gamma(x) \in e(x) \iff \forall x \in \mathit{Var}.\, e_\Gamma(x) = \uparrow\Gamma(x) \subseteq e(x)$$

namely $e_\Gamma \sqsubseteq e$ if and only if $e \models^{\mathcal{F}} \Gamma$. In particular

$$\begin{array}{rcl} (e_\Gamma \sqcup e_{\Gamma'})(x) & = & \uparrow\Gamma(x) \sqcup \uparrow\Gamma'(x) \\ & = & \uparrow(\Gamma(x) \wedge \Gamma'(x)) \\ & = & \uparrow(\Gamma \wedge \Gamma')(x) \end{array}$$

hence $e_\Gamma \sqcup e_{\Gamma'} = e_{\Gamma\wedge\Gamma'}$ by the arbitrary choice of $x \in \mathit{Var}$. Since $\Gamma(x) \in e(x)$ and $\Gamma'(x) \in e(x)$ imply that $\Gamma(x) \wedge \Gamma'(x) = (\Gamma \wedge \Gamma')(x) \in e(x)$ as $e(x)$ is a filter, we have that $e \models^{\mathcal{F}} \Gamma$ and $e \models^{\mathcal{F}} \Gamma'$ imply that $e \models^{\mathcal{F}} \Gamma \wedge \Gamma'$, so that we conclude that $\mathcal{E}$ is directed.

From the above it follows that $\bigsqcup \mathcal{E} \sqsubseteq e$; to see the inverse it suffices to observe that if $\delta \in e(x)$, then $e_{\{x:\delta\}} \in \mathcal{E}$ so that

$$\delta \in e_{\{x:\delta\}}(x) \subseteq \bigcup_{e_\Gamma \in \mathcal{E}} e_\Gamma(x) = (\bigsqcup^{\uparrow}\mathcal{E})(x)$$

for any $x \in \mathit{Var}$, therefore $e \sqsubseteq \bigsqcup \mathcal{E}$.

Part (2): we reason by induction over $Q \in \mathit{Term}$.

Case $Q \equiv x$: then $\{[\![Q]\!]^{\mathcal{F}} e \mid e \in \mathcal{E}\} = \{e(x) \mid e \in \mathcal{E}\}$, which is directed by the hypothesis that $\mathcal{E}$ is such. Then

$$[\![Q]\!]^{\mathcal{F}}(\bigsqcup \mathcal{E}) = (\bigsqcup^{\uparrow}\mathcal{E})(x) = \bigcup_{e\in\mathcal{E}} e(x) = \bigsqcup^{\uparrow}_{e\in\mathcal{E}}([\![Q]\!]^{\mathcal{F}} e)$$

Case $Q \equiv \lambda x.M$: let $\mathcal{X} = \{[\![\lambda x.M]\!]^{\mathcal{F}} e \mid e \in \mathcal{E}\}$, and recall that $[\![\lambda x.M]\!]^{\mathcal{F}} e = {\uparrow}\{\delta \to \tau \mid \tau \in [\![M]\!]^{\mathcal{F}} e[x \mapsto {\uparrow}\delta]\}$. Let $F_1, F_2 \in \mathcal{X}$, namely $F_i = [\![\lambda x.M]\!]^{\mathcal{F}} e_i$ for some $e_i \in \mathcal{E}$ and $i = 1, 2$. By directness of $\mathcal{E}$, there exists $e_3 \in \mathcal{E}$ such that $e_1, e_2 \sqsubseteq e_3$; then take $F_3 = [\![\lambda x.M]\!]^{\mathcal{F}} e_3 \in \mathcal{X}$.

W.l.o.g. a non trivial element F_i for $i = 1, 2$ has the shape $\delta_i = \bigwedge_{j \in J_i} \delta_j^{(i)} \to \tau_j^{(i)}$ where $\tau_j^{(i)} \in [\![M]\!]^{\mathcal{F}} e_i[x \mapsto {\uparrow}\, \delta_j^{(i)}]$ for all $j \in J_i$. Now the set $\{e_1, e_2, e_3\}$ is directed with sup e_3, so that by induction the set of filters $\{[\![M]\!]^{\mathcal{F}} e_i[x \mapsto {\uparrow}\, \delta_j^{(i)}] \mid i = 1, 2, 3\}$ is directed with sup $[\![M]\!]^{\mathcal{F}} e_3[x \mapsto {\uparrow}\, \delta_j^{(i)}]$, for all $j \in J_i$. This implies that $\delta_j^{(i)} \to \tau_j^{(i)} \in [\![\lambda x.M]\!]^{\mathcal{F}} e_3$ for all $j \in J_i$, and hence $\delta_i \in [\![\lambda x.M]\!]^{\mathcal{F}} e_3$ as the latter is a filter. We conclude that $F_1, F_2 \subseteq F_3$, so $\mathcal{X}$ is directed. Now

$$\begin{array}{rcl}
[\![Q]\!]^{\mathcal{F}}(\bigsqcup^{\uparrow} \mathcal{E}) & = & {\uparrow}\{\delta \to \tau \mid \tau \in [\![M]\!]^{\mathcal{F}}(\bigsqcup^{\uparrow} \mathcal{E})[x \mapsto {\uparrow}\delta]\} \\
& = & {\uparrow}\{\delta \to \tau \mid \tau \in \bigcup_{e \in \mathcal{E}} [\![M]\!]^{\mathcal{F}} e[x \mapsto {\uparrow}\delta]\} \\
& & \text{by induction and } (\bigsqcup^{\uparrow} \mathcal{E})[x \mapsto {\uparrow}\delta] = \bigsqcup^{\uparrow}_{e \in \mathcal{E}} e[x \mapsto {\uparrow}\delta] \\
& = & \bigcup_{e \in \mathcal{E}} {\uparrow}\{\delta \to \tau \mid \tau \in [\![M]\!]^{\mathcal{F}} e[x \mapsto {\uparrow}\delta]\} \\
& & \text{since } \mathcal{E} \text{ is directed} \\
& = & \bigcup_{e \in \mathcal{E}} [\![\lambda x.M]\!]^{\mathcal{F}} e \\
& = & \bigsqcup^{\uparrow}_{e \in \mathcal{E}} [\![Q]\!]^{\mathcal{F}} e
\end{array}$$

Case $Q \equiv M \star V$: then $\{[\![M \star V]\!]^{\mathcal{F}} e \mid e \in \mathcal{E}\}$ is directed since $\{[\![M]\!]^{\mathcal{F}} e \mid e \in \mathcal{E}\}$ and $\{[\![V]\!]^{\mathcal{F}} e \mid e \in \mathcal{E}\}$ are such by induction, $[\![M \star V]\!]^{\mathcal{F}} e = ([\![M]\!]^{\mathcal{F}} e) \star^{\mathcal{F}} ([\![V]\!]^{\mathcal{F}} e)$, and the operator $\star^{\mathcal{F}}$ is continuous by Lemma 13.2.17, and hence monotonic. Now

$$\begin{array}{rcll}
[\![Q]\!]^{\mathcal{F}}(\bigsqcup \mathcal{E}) & = & [\![M]\!]^{\mathcal{F}}(\bigsqcup \mathcal{E}) \star^{\mathcal{F}} [\![V]\!]^{\mathcal{F}}(\bigsqcup \mathcal{E}) & \\
& = & (\bigsqcup_{e \in \mathcal{E}} [\![M]\!]^{\mathcal{F}} e) \star^{\mathcal{F}} (\bigsqcup_{e' \in \mathcal{E}} [\![V]\!]^{\mathcal{F}} e') & \text{by induction} \\
& = & \bigsqcup_{e, e' \in \mathcal{E}} ([\![M]\!]^{\mathcal{F}} e) \star^{\mathcal{F}} [\![V]\!]^{\mathcal{F}} e') & \text{by continuity of } \_ \star^{\mathcal{F}} \_ \\
& = & \bigsqcup_{e'' \in \mathcal{E}} ([\![M]\!]^{\mathcal{F}} e'') \star^{\mathcal{F}} [\![V]\!]^{\mathcal{F}} e'') & \text{by directness of } \mathcal{E} \\
& = & \bigsqcup_{e'' \in \mathcal{E}} [\![Q]\!]^{\mathcal{F}} e'' &
\end{array}$$

The remaining cases of $[V]$, $get_\ell(\lambda x.M)$ and $set_\ell(V, M)$ are similar, using the continuity of $unit^{\mathcal{F}}(\_)$, $get^{\mathcal{F}}_\ell(\_)$ and $set^{\mathcal{F}}_\ell(\_, \_)$, respectively.

□

Theorem 13.3.9 (Type semantics). *For all $V \in Val$ and $M \in Com$:*

1. $[\![V]\!]^{\mathcal{F}_D} e = \{\delta \in \mathcal{L}_D \mid \exists \Gamma.\ e \models^{\mathcal{F}} \Gamma \ \& \ \Gamma \vdash V : \delta\}$
2. $[\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e = \{\tau \in \mathcal{L}_{\mathbb{S}D} \mid \exists \Gamma.\ e \models^{\mathcal{F}} \Gamma \ \& \ \Gamma \vdash M : \tau\}$

Proof. Recall that $e \models^{\mathcal{F}} \Gamma$ if and only if $e_\Gamma \sqsubseteq e$:

$$\begin{array}{rcll}
[\![V]\!]^{\mathcal{F}_D} e & = & [\![V]\!]^{\mathcal{F}_D}(\bigsqcup\{e_\Gamma \mid e \models^{\mathcal{F}} \Gamma\}) & \text{by Lemma 13.3.8.1} \\
& = & \bigsqcup_{e_\Gamma \sqsubseteq e} [\![V]\!]^{\mathcal{F}_D} e_\Gamma & \text{by Lemma 13.3.8.2} \\
& = & \bigcup_{e_\Gamma \sqsubseteq e} [\![V]\!]^{\mathcal{F}_D} e_\Gamma & \text{as } \{[\![V]\!]^{\mathcal{F}_D} e_\Gamma \mid e_\Gamma \sqsubseteq e\} \text{ is directed} \\
& & & \text{by Lemma 13.3.8.2} \\
& = & \bigcup_{e_\Gamma \sqsubseteq e} \{\delta \mid \Gamma \vdash V : \delta\} & \text{by Lemma 13.3.6} \\
& = & \{\delta \mid \exists \Gamma.\, \Gamma \vdash V : \delta \ \& \ e \models^{\mathcal{F}} \Gamma\} &
\end{array}$$

The proof in case of $[\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e$ is similar. □

13.4 Typing Configurations

After having obtained a type system for λ_{imp} out of its denotational semantics, and more precisely from the filter-model $\mathcal{F}$, we now look at the relation among the type system and the operational semantics, as presented in Sections 12.2 and 12.3.

However, there is still a mismatch among the type system and the reduction semantics, or its equivalent formulation in terms of convergence, which is due to the fact that the former deals with terms only, whereas the reduction is a relation among configurations (M, s), with $M \in \mathit{Com}$ and $s \in \mathit{Store}$. Therefore we fill the gap by defining the interpretations of store terms and configurations.

Definition 13.4.1. *Let $\mathcal{D} = (D, \mathbb{S}, [\![\cdot]\!]^D, [\![\cdot]\!]^{\mathbb{S}D})$ be a λ_{imp}-model, where $S = (D_\perp)^{\boldsymbol{L}}$. Then we interpret store terms by the maps $[\![\cdot]\!]^S : \mathit{Store} \to \mathit{Term\text{-}Env} \to S$ and $[\![\cdot]\!]^{\boldsymbol{L}} : \mathit{Lkp} \to \mathit{Term\text{-}Env} \to D$ where:*

$$\begin{array}{rcl} [\![\mathit{emp}]\!]^S e & = & \perp_S \\ [\![\mathit{upd}_\ell(u,s)]\!]^S e & = & ([\![s]\!]^S e)[\ell \mapsto [\![u]\!]^{\boldsymbol{L}} e] \\ [\![V]\!]^{\boldsymbol{L}} e & = & [\![V]\!]^D e \\ [\![\mathit{lkp}_\ell(s)]\!]^{\boldsymbol{L}} e & = & [\![s]\!]^S e\, \ell \end{array}$$

where, if $\varsigma \in S$, $\ell \in \boldsymbol{L}$ and $d \in D$, then the map $\varsigma[\ell \mapsto d]$ is the updating of ς that associates d to ℓ.

By instantiating Definition 13.4.1 to the filter-model $\mathcal{F}$ we obtain:

$$\begin{array}{rcl} [\![\mathit{emp}]\!]^{\mathcal{F}_S} e & = & \uparrow\omega_S \\ [\![\mathit{upd}_\ell(u,s)]\!]^{\mathcal{F}_S} e & = & ([\![s]\!]^{\mathcal{F}_S} e)[\ell \mapsto [\![u]\!]^{\mathbf{L}} e] \\ & = & \uparrow(\{\langle \ell : \delta\rangle \mid \delta \in [\![s]\!]^{\mathcal{F}_S} e\} \;\cup\; \{\sigma \in [\![s]\!]^{\mathcal{F}_S} e \mid \ell \notin \mathit{dom}(\sigma)\}) \\ [\![V]\!]^{\mathcal{F}_{\mathbf{L}}} e & = & [\![V]\!]^{\mathcal{F}_D} e \\ [\![\mathit{lkp}_\ell(s)]\!]^{\mathcal{F}_{\mathbf{L}}} e & = & ([\![s]\!]^{\mathcal{F}_S} e) \cdot \{\ell\} = \uparrow\{\delta \mid \langle \ell : \delta\rangle \in [\![s]\!]^{\mathcal{F}_S} e\} \end{array}$$

From the interpretations of store and lookup terms in $\mathcal{F}$ and using the same method as in the previous section, we obtain the following typing rules:

$$\frac{\Gamma \vdash V : \delta}{\Gamma \vdash \mathit{upd}_\ell(V,s) : \langle \ell : \delta\rangle}\,(\mathit{upd}_1) \qquad \frac{\Gamma \vdash s : \langle \ell' : \delta\rangle \quad \ell \neq \ell'}{\Gamma \vdash \mathit{upd}_\ell(V,s) : \langle \ell' : \delta\rangle}\,(\mathit{upd}_2) \tag{13.13}$$

and

$$\frac{\Gamma \vdash s : \langle \ell : \delta\rangle}{\Gamma \vdash \mathit{lkp}_\ell(s) : \delta}\,(\mathit{lkp}) \tag{13.14}$$

Remark 13.4.2. Interpreting store and lookup terms into S and D, respectively, provides a model of the theory axiomatized in Definition 12.2.2. Indeed, by Theorem 12.2.10 the equation $s = t$ is derivable if and only if $s \simeq t$, namely s

and t are extensionally equivalent: $dom(s) = dom(t)$ and for all ℓ in their domain $lkp_\ell(s) \equiv lkp_\ell(t)$, where we stress that this is syntactic identity.

Since the interpretation of store terms is functional and $[\![lkp_\ell(s)]\!]^{\mathbf{L}} e = [\![s]\!]^S e\, \ell$, we immediately conclude that if $\vdash s = t$ then $[\![s]\!]^S e = [\![t]\!]^S e$ for all $e \in$ *Term-Env*. The vice versa does not hold: that $[\![s]\!]^S e\, \ell = [\![V]\!]^D e = [\![W]\!]^D e = [\![t]\!]^S e\, \ell$ does not imply that $V \equiv W$: take e.g. $V \equiv \lambda x.[x]$ and $W \equiv \lambda y.\,([y] \star V)$.

Finally, notice that $[\![emp]\!]^S e = \bot_S \neq [\![upd_\ell(V, s)]\!]^S e$ for any V, s. In particular, if $\Gamma \vdash emp : \sigma$ then $\sigma = \omega_S$, while $\Gamma \vdash upd_\ell(V, s) : \langle \ell : \omega_D \rangle \neq \omega_S$.

Remark 13.4.3. By the same technique used in the proof of Theorem 13.3.9, we can show that $[\![s]\!]^{\mathcal{F}_S} e = \{\sigma \in \mathcal{L}_S \mid \exists \Gamma.\ e \models^{\mathcal{F}} \Gamma \ \ \&\ \ \Gamma \vdash s : \sigma\}$, and similarly for $[\![lkp_\ell(s)]\!]^{\mathcal{F}_{\mathbf{L}}} e$; hence semantically equal store and lookup terms have exactly the same types. As argued in Remark 13.4.2, the interpretation maps $[\![\cdot]\!]^S$ and $[\![\cdot]\!]^{\mathbf{L}}$ provide a model of the equational theory of such terms in any λ_{imp}-model $\mathcal{D}$, hence in $\mathcal{F}$. Therefore if $\vdash s = t$ then s and t have the same types.

On the other hand, by Corollary 12.2.11 any $s \in$ *Store* with $s \not\equiv emp$ can be equated to a term of the form:

$$upd_{\ell_1}(V_1, \cdots upd_{\ell_n}(V_n, emp) \cdots)$$

with distinct locations $\ell_1, \ldots, \ell_n$, so that we may focus on the typing of these terms.

If $s \not\equiv emp$ then $n > 0$ and $\Gamma \vdash V_i : \delta_i$ for some δ_i and each $1 \leq i \leq n$, so that we can derive $\Gamma \vdash s : \langle \ell_i : \delta_i \rangle$ for all i by either using rule (upd_1) or (upd_2). Hence, by repeated use of $(\wedge)$, the judgment $\Gamma \vdash s : \bigwedge_{1 \leq i \leq n} \langle \ell_i : \delta_i \rangle$ can be derived, expressing that s associates to each ℓ_i a value of type δ_i. In summary, the following rule is derivable:

$$\frac{\Gamma \vdash V_1 : \delta_1 \quad \cdots \quad \Gamma \vdash V_n : \delta_n}{\Gamma \vdash upd_{\ell_1}(V_1, \cdots upd_{\ell_n}(V_n, emp) \cdots) : \bigwedge_{1 \leq i \leq n} \langle \ell_i : \delta_i \rangle} \tag{13.15}$$

Vice versa, if s has type $\bigwedge_{1 \leq i \leq n} \langle \ell_i : \delta_i \rangle$ then $lkp_{\ell_i}(s) = V_i$ has type δ_i, by rules $(\leq)$ and (lkp).

Recall that $[\![M]\!]^{\mathbb{S}D} e \in [S \to (D \times S)_\bot]$ and that $[\![s]\!]^S e \in S$; therefore, the following definition of the semantics of a configuration (M, s) is quite naturally the functional application of the meaning of M to that of s:

Definition 13.4.4. *Given a λ_{imp}-model $\mathcal{D}$ and the interpretation maps $[\![\cdot]\!]^S$ and $[\![\cdot]\!]^{\mathbf{L}}$ from Definition 13.4.1, we interpret configurations (M, s) into $C = (D \times S)_\bot$ by*

$$[\![(M, s)]\!]^C e = [\![M]\!]^{\mathbb{S}D} e\, ([\![s]\!]^S e)$$

Once again we look at the instance $\mathcal{F}$ of $\mathcal{D}$, where:

$$[\![(M, s)]\!]^{\mathcal{F}_C} e = ([\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e) \cdot ([\![s]\!]^{\mathcal{F}_S} e) = \uparrow\{\kappa \mid \exists \sigma \in [\![s]\!]^{\mathcal{F}_S} e.\ \sigma \to \kappa \in [\![M]\!]^{\mathcal{F}_{\mathbb{S}D}} e\}$$

from which we derive the typing rule for configurations:

$$\frac{\Gamma \vdash M : \sigma \to \kappa \quad \Gamma \vdash s : \sigma}{\Gamma \vdash (M, s) : \kappa}\ (conf) \tag{13.16}$$

For the reader convenience the rules of the full system are reported in Figure 13.2.

Illustrating the type system w.r.t. the operational semantics. So far the type assignment system has been obtained by means of abstract concepts; here we turn to a more concrete understanding of such rules, looking to the operational semantics. The discussion is informal and includes some examples; the technical analysis of the typing system and operational semantics is deferred to the next sections.

Definition 13.4.4 perfectly matches the operational interpretation of (M, s) as the application $M(s)$ in Section 12.3 (see eq. (12.1) and the comments thereafter), where it returns a pair (V, t) if $(M, s) \Downarrow (V, t)$ and it is undefined, otherwise. Such a correspondence, together with the interpretation of typings, provides the basis for illustrating the typing rules in Figure 13.2.

As usual the interpretation of types are sets or, equivalently, properties of terms. In the present setting this is the content of Proposition 13.3.2, hence the intended meaning of a typing judgment $\Gamma \vdash Q : \varphi$ is that Q satisfies the property φ under the hypothesis that x satisfies $\Gamma(x)$ for all $x \in \mathit{dom}(\Gamma)$.

Since term variables range over values and are values themselves, in a typing context Γ we only have typing judgments of the form $x : \delta$ with $\delta \in \mathcal{L}_D$.

Rules for ω, intersection, and subtyping

$$\frac{}{\Gamma \vdash Q : \omega}\,(\omega) \qquad \frac{\Gamma \vdash Q : \varphi \quad \Gamma \vdash Q : \varphi'}{\Gamma \vdash Q : \varphi \wedge \varphi'}\,(\wedge) \qquad \frac{\Gamma \vdash Q : \varphi \quad \varphi \leq \varphi'}{\Gamma \vdash Q : \varphi'}\,(\leq)$$

Rules for *Val* and *Com* terms

$$\frac{}{\Gamma, x : \delta \vdash x : \delta}\,(\mathit{var}) \qquad \frac{\Gamma,\, x : \delta \vdash M : \tau}{\Gamma \vdash \lambda x.M : \delta \to \tau}\,(\lambda)$$

$$\frac{\Gamma \vdash V : \delta}{\Gamma \vdash [V] : \sigma \to \delta \times \sigma}\,(\mathit{unit}) \qquad \frac{\Gamma \vdash M : \sigma \to \delta' \times \sigma' \quad \Gamma \vdash V : \delta' \to \sigma' \to \delta'' \times \sigma''}{\Gamma \vdash M \star V : \sigma \to \delta'' \times \sigma''}\,(\star)$$

$$\frac{\Gamma,\, x : \delta \vdash M : \sigma \to \kappa}{\Gamma \vdash \mathit{get}_\ell(\lambda x.M) : (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa}\,(\mathit{get}) \qquad \frac{\Gamma \vdash V : \delta \quad \Gamma \vdash M : (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa \quad \ell \notin \mathit{dom}(\sigma)}{\Gamma \vdash \mathit{set}_\ell(V, M) : \sigma \to \kappa}\,(\mathit{set})$$

Rules for *Store*, *Lkp* terms and configurations

$$\frac{\Gamma \vdash V : \delta}{\Gamma \vdash \mathit{upd}_\ell(V, s) : \langle \ell : \delta \rangle}\,(\mathit{upd}_1) \qquad \frac{\Gamma \vdash s : \langle \ell' : \delta \rangle \quad \ell \neq \ell'}{\Gamma \vdash \mathit{upd}_\ell(V, s) : \langle \ell' : \delta \rangle}\,(\mathit{upd}_2)$$

$$\frac{\Gamma \vdash s : \langle \ell : \delta \rangle}{\Gamma \vdash \mathit{lkp}_\ell(s) : \delta}\,(\mathit{lkp}) \qquad \frac{\Gamma \vdash M : \sigma \to \kappa \quad \Gamma \vdash s : \sigma}{\Gamma \vdash (M, s) : \kappa}\,(\mathit{conf})$$

Figure 13.2: Type assignment system for λ_{imp}

Rules for ω, intersection and subtyping Rules for ω_A, intersection and subtyping are the same as in [BCD83], instantiated to the four kinds of judgment of the present system. The *universal types* ω_A are interpreted as the whole set of terms that can be typed by a type of sort $\mathcal{L}_A$ (see Definition 14.2.2); they are usually called *trivial types* since do not carry any information about their subjects, but for the sort in the language. Intersection and subtyping are understood set theoretically, which justifies rules $(\wedge)$ and $(\leq)$. In particular if $\varphi = \varphi'$ then $\Gamma \vdash Q : \varphi$ implies $\Gamma \vdash Q : \varphi'$.

Rules for *Val* and *Com* terms Coming to the rules for values and computations, rule (*var*) is obvious. About rule (λ) we observe that in our syntax abstractions have the shape $\lambda x.M$ where $M \in \mathit{Com}$, so that they represent functions from values (the variable x can be substituted only by terms in *Val*) to computations. Also, abstractions are values and by rule (λ) are assigned functional types $\delta \to \tau \in \mathcal{L}_D$ if δ is the type of x and $\tau \in \mathcal{L}_{\mathbb{S}D}$ is a type of M, as usual.

We observe that $\delta \leq_D \omega_D \leq_{D_\perp} \omega_{D_\perp}$, therefore any value V has $\omega_{D_\perp}$ as its type by rule $(\leq)$.

Rules (*unit*) and $(\star)$ are instances of the corresponding rules in pure computational λ-calculus in [dT20], specialized to the case of the state monad. The term $[V]$ represents the trivial computation returning V; as a function of stores it is such that $[V](s) = (V, s)$ behaving as the constant function w.r.t. the value V, and as the identity w.r.t. the store s; therefore if δ is a type of V any type $\sigma \to \delta \times \sigma$ can be assigned to $[V]$ by rule (*unit*).

In rule $(\star)$ the first premise says that applying M to a store s of type σ yields a result $M(s) = (W, t)$ such that $W : \delta'$ and $t : \sigma'$. We abbreviate this by saying that $(W, t) = [W](t)$ has type $\delta' \times \sigma'$. In the second premise the value V of type $\delta' \to \sigma' \to \delta'' \times \sigma''$ is the currified version of a function of type $\delta' \times \sigma' \to \delta'' \times \sigma''$ (which is not a type in our formalism) sending (W, t) to some new result of type $\delta'' \times \sigma''$. By the operational semantics of the $\star$ operator, we then conclude that for arbitrary s of type σ, $(M \star V)(s)$ yields a result of type $\delta'' \times \sigma''$, hence the typing $M \star V : \sigma \to \delta'' \times \sigma''$ in the conclusion of the rule.

Rules for *Store*, *Lkp* terms, and configurations Since there is no rule explicitly typing *emp*, $\Gamma \vdash \mathit{emp} : \omega_S$ is the only typing of *emp* in the any context Γ, up to $=_S$.

According to rule (*get*), the result of $(\mathit{get}_\ell(\lambda x.M))(s)$ has type κ if s has type $\langle \ell : \delta \rangle \wedge \sigma$, hence it has both type $\langle \ell : \delta \rangle$ and σ. As we have seen above, if s has type $\langle \ell : \delta \rangle$ and $\delta \neq \omega_{D_\perp}$ then $\mathit{lkp}_\ell(s) = V$ for some value V of type δ. Now, from the operational semantics we may argue that $(\mathit{get}_\ell(\lambda x.M))(s) = (M\{V/x\})(s)$; hence if s has type σ then a sufficient condition to produce a result $M(s)$ of type κ is to assume that x in M has type δ, as required in the premise of the rule.

Rule (*set*) is the central and subtlest rule of the system. Reasoning as before, we argue that

$$(\mathit{set}_\ell(V, M))(s) = M(\mathit{upd}_\ell(V, s))$$

and we look for sufficient conditions for the result having type κ whenever s has type σ. Now if V has type δ as in the first premise, then $\mathit{upd}_\ell(V, s)$ has type

$\langle \ell : \delta \rangle$; moreover, as discussed with reference to the derived rule (13.15), it has also type σ, because $\ell \notin \mathit{dom}(\sigma)$, namely the side condition of the rule, so that it suffices that M has type $\langle \ell : \delta \rangle \wedge \sigma \to \kappa$ to conclude, which is the second premise.

The rules (*set*) and (*get*) are dual, in a sense. Consider the derivation:

$$\dfrac{\Gamma \vdash V : \delta \qquad \dfrac{\Gamma, x : \delta \vdash M : \sigma \to \kappa}{\Gamma \vdash \mathit{get}_\ell(\lambda x.M) : (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa}\,(\mathit{get}) \qquad \ell \notin \mathit{dom}(\sigma)}{\Gamma \vdash \mathit{set}_\ell(V, \mathit{get}_\ell(\lambda x.M)) : \sigma \to \kappa}\,(\mathit{set})$$

Since $\langle \ell : \delta \rangle$ is strictly smaller than ω_S for any δ, it follows that $\omega_S \to \kappa < \langle \ell : \delta \rangle \wedge \sigma \to \kappa$ for any δ and σ. Indeed, as a consequence of Theorem 14.2.6, we know that $\Gamma \not\vdash \mathit{get}_\ell(\lambda x.M) : \omega_S \to \omega_D \times \omega_S$.

In fact, $\mathit{get}_\ell(\lambda x.M) \not\Downarrow$ because $(\mathit{get}_\ell(\lambda x.M), \mathit{emp})$ is a blocked configuration. But if the term $\mathit{get}_\ell(\lambda x.M)$ is placed in a suitable context, as in the conclusion of the derivation above, then the type $\omega_S \to \omega_D \times \omega_S$ is derivable for $\mathit{set}_\ell(V, \mathit{get}_\ell(\lambda x.M))$ depending on the typing of M. E.g. consider $M \equiv [x]$, one can take $\sigma = \omega_S$ and $\kappa = \omega_D \times \omega_S$; from the derivation above, we see that $M' = \mathit{set}_\ell(V, \mathit{get}_\ell(\lambda x.[x]))$ has type $\omega_S \to \omega_D \times \omega_S$. On the other hand, (M', emp) converges:

$$(M', \mathit{emp}) \to (\mathit{get}_\ell(\lambda x.[x]), \mathit{upd}_\ell(V, \mathit{emp})) \to ([V], \mathit{upd}_\ell(V, \mathit{emp}))$$

It remains to explain why the side condition $\ell \notin \mathit{dom}(\sigma)$ in rule (*set*) is not only sufficient, but also necessary. Recall that $\ell \notin \mathit{dom}(\sigma)$ implies that there exists no $\delta \in \mathcal{L}_D$ such that $\sigma \leq_S \langle \ell : \delta \rangle$, see Remark 13.2.13. Therefore, if $\sigma = \bigwedge_i \langle \ell_i : \delta_i \rangle$ where $\delta \in \mathcal{L}_D$, then for all i we have $\ell_i \neq \ell$. This means that from the side condition no information about the value associated to ℓ by stores of type σ is recorded in the type.

Therefore the typing $M : (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa$ in the second premise does not depend on any information about the value of the store at ℓ, but for the δ in $\langle \ell : \delta \rangle$, which is the type of V in the first premise.

Dropping the side condition we could have, say, $M : \langle \ell : \delta \rangle \wedge \langle \ell : \delta' \rangle \to \kappa$ where $\langle \ell : \delta \rangle \wedge \langle \ell : \delta' \rangle = \langle \ell : \delta \wedge \delta' \rangle$, with both $\delta, \delta' \in \mathcal{L}_D$, and hence we could derive the type $\langle \ell : \delta' \rangle \to \kappa$ for $\mathit{set}_\ell(V, M)$ in the conclusion. Now this says that if s has type $\langle \ell : \delta' \rangle$ then $(\mathit{set}_\ell(V, M))(s) = M(\mathit{upd}_\ell(V, s))$ will have type κ, notwithstanding that the value of $\mathit{upd}_\ell(V, s)$ at ℓ is V, of which we only know that it has type δ, not δ'.

We further illustrate the point in the next example.

Example 13.4.5. A key issue in defining the system in Figure 13.2 is how to type terms like $M \equiv \mathit{set}_\ell(W, \mathit{set}_\ell(V, \mathit{get}_\ell(\lambda x.[x])))$ from Example 12.2.12, exhibiting the strong update property of the calculus. Let us abbreviate $N \equiv \mathit{set}_\ell(V, \mathit{get}_\ell(\lambda x.[x]))$, and suppose that $\vdash V : \delta$ and $\vdash W : \delta'$; we want to derive $\vdash M : \sigma \to \kappa$ for suitable σ and κ. Suppose that the derivation ends by (*set*), then

$$\dfrac{\vdash W : \delta' \quad \vdash N : \langle \ell : \delta' \rangle \wedge \sigma \to \kappa \quad \ell \notin \mathit{dom}(\sigma)}{\vdash M \equiv \mathit{set}_\ell(W, N) : \sigma \to \kappa}\,(\mathit{set})$$

where the derivation of the second premise is

$$\dfrac{\vdash V : \delta \quad \vdash get_\ell(\lambda x.[x]) : \langle \ell : \delta \rangle \wedge \langle \ell : \delta' \rangle \wedge \sigma \to \kappa \quad (*)}{\vdash N \equiv set_\ell(V, get_\ell(\lambda x.[x])) : \langle \ell : \delta' \rangle \wedge \sigma \to \kappa}\ (set)$$

and $(*)$ is $\ell \notin dom(\langle \ell : \delta' \rangle \wedge \sigma)$. Since $\ell \notin dom(\sigma)$, $(*)$ implies that $\delta' = \omega_{D_\bot}$, and therefore, $\sigma \to \kappa = \omega_S \wedge \sigma \to \kappa = \langle \ell : \omega_{D_\bot} \rangle \wedge \sigma \to \kappa$.

On the other hand, for all σ:

$$\dfrac{x : \delta \vdash [x] : \sigma \to \kappa}{\vdash get_\ell(\lambda x.[x]) : \langle \ell : \delta \rangle \wedge \sigma \to \delta \times \sigma}\ (get)$$

Hence, M has type $\sigma \to \delta \times \sigma$.

Now, suppose that $\vdash s : \sigma$: hence $\vdash (M, s) : \delta \times \sigma$. From Example 12.2.12, we know that $(M, s) \to^* ([V], upd_\ell(V, upd_\ell(W, s)))$; since we expect that the types are preserved by reduction (see Theorem 14.1.4), we must be able to type $\vdash [V] : \sigma \to \delta \times \sigma$, which is plain by (*unit*) and the hypothesis that $\vdash V : \delta$, and $\vdash upd_\ell(V, upd_\ell(W, s)) : \sigma$.

Let us assume that $\sigma \neq \omega_S$ (otherwise the thesis is trivial). Then $\sigma = \bigwedge_i \langle \ell_i : \delta_i \rangle$ with $\delta_i \neq \omega_{D_\bot}$ for all i, so that $\ell \notin dom(\sigma)$ implies that $\ell \neq \ell_i$ for all i. Therefore $\vdash s : \langle \ell_i : \delta_i \rangle$ for all i by $(\leq)$ and

$$\dfrac{\dfrac{\vdash s : \langle \ell_i : \delta_i \rangle}{\vdash upd_\ell(W, s) : \langle \ell_i : \delta_i \rangle}\ (upd_2)}{\vdash upd_\ell(V, upd_\ell(W, s)) : \langle \ell_i : \delta_i \rangle}\ (upd_2)$$

Now by repeated applications of $(\wedge)$ we conclude $\vdash upd_\ell(V, upd_\ell(W, s)) : \sigma$, as desired.

On passing we observe that $upd_\ell(V, upd_\ell(W, s)) = upd_\ell(V, s)$ in the algebra of store terms, and that the latter has the same types as the former.

Remark 13.4.6. From the example above, we realize that the following rule is derivable

$$\dfrac{\Gamma \vdash M : \sigma \to \kappa \qquad \ell \notin dom(\sigma)}{\Gamma \vdash set_\ell(V, M) : \sigma \to \kappa}\ (set')$$

By adding this rule to the assignment system, we could restrict type for values just to $\delta \in \mathcal{L}_D$. We prefer the present system to save the symmetry among the rules (*set*) and (*get*).

CHAPTER 14

CHARACTERIZATION OF CONVERGENCE

14.1 Type Invariance

A characteristic property of intersection types for ordinary λ-calculus is invariance both under reduction and expansion of the subject. The analogous results are proved for the pure computational λ-calculus in [dT20]. Here we extend such results to the present calculus w.r.t. the typing of algebraic operations.

A preliminary lemma states a fundamental property of subtyping of arrow types, both in Th_D and in $Th_{\mathbb{S}D}$:

Lemma 14.1.1. *Let $\tau \neq \omega_{\mathbb{S}D}$ and $\kappa \neq \omega_C$, then:*

1. $\bigwedge_{i\in I}(\delta_i \to \tau_i) \leq_D \delta \to \tau \iff$
   $\exists J \subseteq I.\ J \neq \emptyset \ \ \&\ \ \delta \leq_D \bigwedge_{j\in J} \delta_j \ \ \&\ \ \bigwedge_{j\in J} \tau_j \leq_{\mathbb{S}D} \tau$
2. $\bigwedge_{i\in I}(\sigma_i \to \kappa_i) \leq_{\mathbb{S}D} \sigma \to \kappa \iff$
   $\exists J \subseteq I.\ J \neq \emptyset \ \ \&\ \ \sigma \leq_S \bigwedge_{j\in J} \sigma_j \ \ \&\ \ \bigwedge_{j\in J} \kappa_j \leq_C \kappa$

Proof. By induction over the definition of $\leq$. □

The next lemma is an extension of the analogous property of the system in [BCD83], also called Inversion Lemma in [BDS13] 14.1 because it is the backward reading of the typing rules.

Lemma 14.1.2 (Generation lemma). *Assume that $\delta \neq \omega_D$, $\sigma \neq \omega_S$, and $\tau \neq \omega_{\mathbb{S}D}$, then:*

1. $\Gamma \vdash x : \delta \iff \Gamma(x) \leq_D \delta$
2. $\Gamma \vdash \lambda x.M : \delta \iff \exists I, \delta_i, \tau_i.\ \forall i \in I.\ \Gamma, x : \delta_i \vdash M : \tau_i \ \ \&\ \ \bigwedge_{i\in I} \delta_i \to \tau_i \leq_D \delta$
3. $\Gamma \vdash [V] : \tau \iff$
   $\exists I, \delta_i, \sigma_i.\ \forall i \in I.\ \Gamma \vdash V : \delta_i \ \ \&$
   $\bigwedge_{i\in I} \sigma_i \to \delta_i \times \sigma_i \leq_{\mathbb{S}D} \tau$

4. $\Gamma \vdash M \star V : \tau \iff$
$\exists I, \tau_i = \sigma_i \to \delta_i \times \sigma'_i, \delta''_i, \sigma''_i.\ \forall i \in I.\ \Gamma \vdash M : \sigma_i \to \delta''_i \times \sigma''_i$ &
$\Gamma \vdash V : \delta''_i \to \sigma''_i \to \delta_i \times \sigma'_i$ & $\bigwedge_{i\in I} \tau_i \leq_{\mathbb{S}D} \tau$

5. $\Gamma \vdash \mathit{get}_\ell(\lambda x.M) : \tau \iff$
$\exists I, \delta_i, \sigma_i, \kappa_i.\ \forall i \in I.\ \Gamma, x : \delta_i \vdash M : \sigma_i \to \kappa_i$ &
$\bigwedge_{i\in I}(\langle \ell : \delta_i\rangle \wedge \sigma_i \to \kappa_i) \leq_{\mathbb{S}D} \tau$

6. $\Gamma \vdash \mathit{set}_\ell(V, M) : \tau \iff$
$\exists I, \delta_i, \sigma_i, \kappa_i.\ \forall i \in I.\ \Gamma \vdash M : \langle \ell : \delta_i\rangle \wedge \sigma_i \to \kappa_i$ &
$\Gamma \vdash V : \delta_i$ & $\bigwedge_{i\in I}(\sigma_i \to \kappa_i) \leq_{\mathbb{S}D} \tau$ &
$\ell \notin \mathit{dom}(\bigwedge_{i\in I} \sigma_i)$

7. $\Gamma \vdash s : \sigma \iff$
$\exists I, l_i, \delta_i.\ \forall i \in I.\ \exists V_i.\ \mathit{lkp}_{\ell_i}(s) = V_i$ &
$\Gamma \vdash V_i : \delta_i$ & $\sigma \leq_S \bigwedge_{i\in I}\langle \ell_i : \delta_i\rangle$

8. $\Gamma \vdash \mathit{lkp}_\ell(s) : \delta \iff \exists I, \delta_i.\ \forall i \in I.\ \Gamma \vdash s : \langle \ell : \delta_i\rangle$ & $\bigwedge_{i\in I} \delta_i \leq_D \delta$

9. $\Gamma \vdash \mathit{upd}_\ell(V, s) : \sigma \iff$
$\exists I, \delta_i, \ell_i, \delta.\ \forall i \in I.\ s = \bigwedge_{i\in I}\langle \ell_i : \delta_i\rangle$ *where* $\ell_i \neq \ell$ &
$\Gamma \vdash V : \delta$ & $\langle \ell : \delta\rangle \wedge \bigwedge_{i\in I}\langle \ell_i : \delta_i\rangle \leq \sigma$

10. $\Gamma \vdash (M, s) : \kappa \iff$
$\exists I, \sigma_i, \kappa_i.\ \forall i \in I.\ \Gamma \vdash M : \sigma_i \to \kappa_i$ &
$\Gamma \vdash s : \sigma_i$ & $\bigwedge_{i\in I} \kappa_i \leq \kappa$

Proof. The implications $\Leftarrow$ are immediate. To see the implications $\Rightarrow$ we reason by induction over the derivations, by distinguishing some cases based on the last rule. Parts i) and 2 are the same as for ordinary intersection types and λ-calculus; part 3 is immediate by the induction hypothesis, hence we treat part 4, 5, and 6, only.

4. If the last rule in the derivation of $\Gamma \vdash M \star V : \tau$ is $(\star)$ just take I as a singleton set. If it is $(\leq)$ then the thesis follows immediately by induction and the transitivity of $\leq_{\mathbb{S}D}$. Finally, suppose that the derivation ends by

$$\frac{\Gamma \vdash M \star V : \tau_1 \quad \Gamma \vdash M \star V : \tau_2}{\Gamma \vdash M \star V : \tau_1 \wedge \tau_2}\ (\wedge)$$

and $\tau \equiv \tau_1 \wedge \tau_2$. Then by induction we have $\exists I, \tau_i = \sigma_i \to \delta_i \times \sigma'_i, \delta''_i, \sigma''_i.\ \forall i \in I.\ \Gamma \vdash M : \sigma_i \to \delta''_i \times \sigma''_i$ & $\Gamma \vdash V : \delta''_i \to \sigma''_i \to \delta_i \times \sigma'_i$ & $\bigwedge_{i\in I} \tau_i \leq_{\mathbb{S}D} \tau_1$ and $\exists J, \tau_j = \sigma_j \to \delta_j \times \sigma'_j, \delta''_j, \sigma''_j.\ \forall j \in J.\ \Gamma \vdash M : \sigma_j \to \delta''_j \times \sigma''_j$ & $\Gamma \vdash V : \delta''_j \to \sigma''_j \to \delta_j \times \sigma'_j$ & $\bigwedge_{i\in I} \tau_j \leq_{\mathbb{S}D} \tau_2$. From this, the thesis immediately follows by observing that

$$\bigwedge_{i\in I} \tau_i \leq_{\mathbb{S}D} \tau_1 \ \&\ \bigwedge_{j\in J} \tau_j \leq_{\mathbb{S}D} \tau_2 \Rightarrow \bigwedge_{i\in I} \tau_i \wedge \bigwedge_{j\in J} \tau_j \leq_{\mathbb{S}D} \tau_1 \wedge \tau_2.$$

5. If the last rule in the derivation of $\Gamma \vdash \mathit{get}_\ell(\lambda x.M) : \tau$ is (get) just take I as a singleton set, as before. Also case subsumption case is a routine proof by i.

h. Let's suppose the derivation ends by

$$\frac{\Gamma \vdash get_\ell(\lambda x.M) : \tau_1 \quad \Gamma \vdash get_\ell(\lambda x.M) : \tau_2}{\Gamma \vdash get_\ell(\lambda x.M) : \tau_1 \wedge \tau_2}\,(\wedge)$$

Then by induction we have:
$\exists I, \delta_i, \sigma_i, \kappa_i.\ \forall i \in I.\ \Gamma, x : \delta_i \vdash M : \sigma_i \to \kappa_i\ \ \&\ \ \bigwedge_{i \in I}(\langle \ell : \delta_i \rangle \wedge \sigma_i \to \kappa_i) \leq_{\mathbb{S}D} \tau_1$ and $\exists J, \delta_j, \sigma_j, \kappa_j.\ \forall j \in J.\ \Gamma, x : \delta_j \vdash M : \sigma_j \to \kappa_j\ \ \&\ \ \bigwedge_{j \in J}(\langle \ell : \delta_j \rangle \wedge \sigma_j \to \kappa_j) \leq_{\mathbb{S}D} \tau_2$ Since $\tau \neq \omega_{\mathbb{S}D}$ then also $\tau_1, \tau_2 \neq \omega_{\mathbb{S}D}$, hence w.l.o.g. we can suppose that for $i = 1, 2$ there exist $\bar{\sigma_i}$ and $\bar{\kappa_i}$ such that τ_i has the shape $\bar{\sigma_i} \to \bar{\kappa_i}$.

By Part (2) of Lemma 14.1.1 we know that $\exists \bar{I} \subseteq I.\ \bar{I} \neq \emptyset\ \ \&\ \ \bar{\sigma_1} \leq_S \bigwedge_{i \in \bar{I}} \sigma_i \wedge \langle \ell : \delta_i \rangle\ \ \&\ \ \bigwedge_{i \in \bar{I}} \kappa_i \leq_C \bar{\kappa_1}$ and $\exists \bar{J} \subseteq J.\ \bar{J} \neq \emptyset\ \ \&\ \ \bar{\sigma_2} \leq_S \bigwedge_{j \in \bar{J}} \sigma_j \wedge \langle \ell : \delta_j \rangle\ \ \&\ \ \bigwedge_{j \in \bar{J}} \kappa_j \leq_C \bar{\kappa_2}$ By this, the thesis follows by observing that $\langle \ell : \bigwedge_{i \in \bar{I}} \delta_i \wedge \bigwedge_{j \in \bar{J}} \delta_j \rangle \wedge \bigwedge_{i \in \bar{I}} \sigma_i \wedge \bigwedge_{j \in \bar{J}} \sigma_j \to \bigwedge_{i \in \bar{I}} \kappa_i \wedge \bigwedge_{j \in \bar{J}} \kappa_j \leq_{\mathbb{S}D} \tau_1 \wedge \tau_2 = \tau$

6. As in the previous parts, let's discuss just about the case in which the last rule in the derivation is $(\wedge)$:

$$\frac{\Gamma \vdash set_\ell(V, M) : \tau_1 \quad \Gamma \vdash set_\ell(V, M) : \tau_2}{\Gamma \vdash set_\ell(V, M) : \tau_1 \wedge \tau_2}\,(\wedge)$$

then, by induction we have:

$\exists I, \delta_i, \sigma_i, \kappa_i.\ \forall i \in I.\ \Gamma \vdash M : \langle \ell : \delta_i \rangle \wedge \sigma_i \to \kappa_i\ \ \&\ \ \Gamma \vdash V : \delta_i\ \ \&\ \ \bigwedge_{i \in I}(\sigma_i \to \kappa_i) \leq_{\mathbb{S}D} \tau_1\ \ \&\ \ \ell \notin dom(\bigwedge_{i \in I} \sigma_i)$

$\exists J, \delta_j, \sigma_j, \kappa_j.\ \forall j \in J.\ \Gamma \vdash M : \langle \ell : \delta_j \rangle \wedge \sigma_j \to \kappa_j\ \ \&\ \ \Gamma \vdash V : \delta_j\ \ \&\ \ \bigwedge_{j \in J}(\sigma_j \to \kappa_j) \leq_{\mathbb{S}D} \tau_2\ \ \&\ \ \ell \notin dom(\bigwedge_{j \in J} \sigma_j)$.

The thesis by similar use of Part (2) of Lemma 14.1.1 as in the previous point and by the fact that by Definition 13.2.12 we have that $\ell \notin dom(\bigwedge_{i \in I} \sigma_i) \cup dom(\bigwedge_{j \in J} \sigma_j)$, hence $\ell \notin dom(\bigwedge_{i \in I} \sigma_i \wedge \bigwedge_{j \in J} \sigma_j)$. □

Lemma 14.1.3 (Substitution and expansion)**.**

1. *If* $\Gamma, x : \delta \vdash M : \tau$ *and* $\Gamma \vdash V : \delta$ *then* $\Gamma \vdash M\{V/x\} : \tau$.
2. *If* $\Gamma \vdash M\{V/x\} : \tau$ *then there exists* $\delta \in \mathcal{L}_D$ *such that:*

$$\Gamma \vdash V : \delta \quad \textit{and} \quad \Gamma, x : \delta \vdash M : \tau$$

Proof. Both parts are proved by induction over derivations, using Lemma 14.1.1 and Lemma 14.1.2. □

We are now ready to establish the type invariance property w.r.t. reduction and expansion:

Theorem 14.1.4 (Subject reduction)**.**

$$\Gamma \vdash (M, s) : \kappa\ \ \&\ \ (M, s) \to (N, t)\ \ \Rightarrow\ \ \Gamma \vdash (N, t) : \kappa$$

Proof. Let us assume that $\kappa \neq \omega_C$ since the thesis is trivial, otherwise. The proof is by induction over the definition of $(M, s) \to (N, t)$, using Lemma 14.1.2. We treat the interesting cases. From the hypothesis $\Gamma \vdash (M, s) : \kappa$ and by Part 10 of Lemma 14.1.2 we have that there exist a finite set I and types σ_i, κ_i such that for all $i \in I$:

(a) $\Gamma \vdash M : \sigma_i \to \kappa_i$;

(b) $\Gamma \vdash s : \sigma_i$

(c) $\bigwedge_{i\in I} \kappa_i \leq_C \kappa$.

Case $([V] \star (\lambda x.M'), s) \to (M'\{V/x\}, s)$:

By ((a)), and using Parts 3 and 4 of Generation Lemma 14.1.2, for all $i \in I$ there is $J_i, \delta_{ij}, \delta'_{ij}, \delta''_{ij}, \sigma''_{ij}$ such that for all $j \in J_i$:

(d) $\Gamma \vdash V : \delta'_{ij}$ with $\bigwedge_{j\in J_i} \delta'_{ij} \leq_D \delta_{ij}$, where $\Gamma \vdash [V] : \sigma_{ij} \to \delta'_{ij} \times \sigma_{ij}$

(e) $\Gamma \vdash \lambda x.M' : \delta_{ij} \to \tau_{ij}$ with $\bigwedge_{j\in J_i} \tau_{ij} \leq_C \sigma_i \to \kappa_i$, where $\tau_{ij} = \sigma \to \delta''_{ij} \times \sigma''_{ij}$.

By applying Part 3 of Generation Lemma 14.1.2 to ((e)), for all $i \in I$, $j \in J_i$, there is a finite set K_{ij}, $\delta_{ijk}, \delta''_{ijk}, \sigma_{ijk}$ such that for all $k \in K_{ij}$:

(f) $\Gamma, x : \delta_{ijk} \vdash M' : \sigma_{ijk} \to \delta''_{ijk} \times \sigma''_{ijk}$ with $\bigwedge_{k\in K}(\delta_{ijk} \to \sigma_{ijk} \to \delta''_{ijk} \times \sigma''_{ijk}) \leq_{\mathbb{S}D} \delta_{ij} \to \tau_{ij}$.

Set $\sigma_{ijk} \to \delta''_{ijk} \times \sigma''_{ijk} =: \tau_{ijk}$. In virtue of Lemma 14.1.1, we may assume w.l.o.g. that there exists a not empty set $\overline{K} \subseteq K_{ij}$ such that $\delta_{ij} \leq_D \bigwedge_{k\in\overline{K}} \delta_{ijk}$ and $\bigwedge_{k\in\overline{K}} \tau_{ijk} \leq_{\mathbb{S}D} \tau_{ij}$.

By ((e)) we have: $\delta'_{ij} \leq_D \delta_{ij} \leq_D \delta_{ijk} \Rightarrow \Gamma \vdash V : \delta_{ijk}$.
By ((f)) we have: $\bigwedge_{i\in I} \bigwedge_{j\in J_i} \bigwedge_{k\in\overline{K}} \leq_{\mathbb{S}D} \tau_{ij} \leq_{\mathbb{S}D} \sigma_i \to \kappa_i$.

In conclusion, by Substitution Lemma 14.1.3, $\Gamma \vdash M'\{V/x\} : \sigma_i \to \kappa_i$, hence by (*conf*) $\Gamma \vdash (M'\{V/x\}, s) : \kappa_i$; now by repeated applications of rule $(\wedge)$ and $(\leq)$, $\Gamma \vdash (M'\{V/x\}, s) : \kappa$.

Case $(set_\ell(V, M), s) \to (M, upd_\ell(V, s))$:

By applying Part 6 of Generation Lemma to ((a)):

$\forall i \in I.\ \exists J_i, \delta_{ij}, \sigma_{ij}, \kappa_{ij}$ such that for all $j \in J_i$:

1. $\Gamma \vdash M' : \langle \ell : \delta_{ij} \rangle \wedge \sigma_{ij} \to \kappa_{ij}$
2. $\Gamma \vdash V : \delta_{ij}$
3. $\bigwedge_{j\in J_i}(\sigma_{ij} \to \kappa_{ij}) \leq_{\mathbb{S}D} \sigma_i \to \kappa_i$
4. $\ell \notin dom(\bigwedge_{j\in J_i} \sigma_{ij})$

By Parts 3 and 4 of Generation Lemma 14.1.2 and Lemma 14.1.1: there exists $\overline{J} \in J_i$

$$\sigma_i \leq_S \bigwedge_{j\in\overline{J}} \sigma_{ij} \ \ \&\ \ \bigwedge_{j\in\overline{J}} \kappa_{ij} \leq_C \kappa_i$$

Moreover, by (upd_2) and $(\wedge)$: $\Gamma \vdash upd_\ell(V, s) : \sigma_i \wedge \langle \ell : \delta_{ij} \rangle$. We conclude by (*conf*) and $(\leq)$.

Case $(get_\ell(\lambda x.M), s) \to (M\{V/x\}, s)$ where $lkp_\ell(s) = V$:

By applying Part 5 of Generation Lemma to (a), for all $i \in I$ there exist $J_i, \delta_{ij}, \sigma_{ij}, \kappa_{ij}$ such that for all $j \in J_i$:

$$\Gamma, x : \delta_{ij} \vdash M' : \sigma_{ij} \to \kappa_{ij} \;\;\&\;\; \bigwedge_{j \in J_i} (\langle \ell : \delta_{ij} \rangle \wedge \sigma_{ij} \to \kappa_{ij}) \leq_{\mathbb{S}D} \sigma_i \to \kappa_i$$

By Lemma 14.1.1 there exists $\overline{J} \in J_i$

$$\sigma_i \leq_S \bigwedge_{j \in \overline{J}} \langle \ell : \delta_{ij} \rangle \wedge \sigma_{ij} \;\;\&\;\; \bigwedge_{j \in \overline{J}} \kappa_{ij} \leq_C \kappa_i$$

By Part 7 of Generation Lemma 14.1.2 we know that there exist at least one V such that $lkp_\ell(s) = V$ and $\Gamma \vdash V : \delta_{ij}$. We conclude by applying Substitution Lemma 14.1.3 and then routine arguments.

All other cases are immediate by Lemma 14.1.2. □

In order to prove Subject Expansion, we have to to establish some properties of stores, relating store terms with their types.

Lemma 14.1.5.

(a) $\ell \notin dom(\sigma) \Rightarrow [\Gamma \vdash s : \sigma \iff \Gamma \vdash upd_\ell(V, s) : \sigma]$

(b) $\Gamma \vdash upd_\ell(V, s) : \sigma \;\;\&\;\; \sigma \leq_S \langle \ell : \delta \rangle \neq \omega_S \Rightarrow \Gamma \vdash V : \delta$

Proof. ((a)) The if part is immediate by induction over the derivation of $\Gamma \vdash upd_\ell(V, s) : \sigma$. For the only if part, when $\sigma = \omega_S$ the thesis follows by (ω). Otherwise, let $\sigma = \bigwedge_{i \in I} \langle \ell_i : \delta_i \rangle$. Then $I \neq \emptyset$ and for all $i \in I$ we have $\delta_i \neq \omega_D$ and then $\ell_i \neq \ell$ because $\ell \notin dom(\sigma)$. Therefore, for all $i \in I$, $\Gamma \vdash s : \langle \ell_i : \delta_i \rangle$ by $(\leq)$ and $\Gamma \vdash upd_\ell(V, s) : \langle \ell_i : \delta_i \rangle$ by (upd_2), and we conclude by $(\wedge)$.

((b)) By hypothesis $\sigma = \bigwedge_{i \in I} \langle \ell_i : \delta_i \rangle \leq_S \langle \ell : \delta \rangle$, where w.l.o.g. we assume that in $\bigwedge_{i \in I} \langle \ell_i : \delta_i \rangle$ the ℓ_i are pairwise distinct. Then there exists exactly one $i' \in I$ such that $\langle \ell_{i'} : \delta_{i'} \rangle \leq_S \langle \ell : \delta \rangle$, so that $\ell_{i'} = \ell$ and $\delta_{i'} \leq_D \delta$. Now $\Gamma \vdash upd_\ell(V, s) : \sigma$ implies that $\Gamma \vdash upd_\ell(V, s) : \langle \ell_{i'} : \delta_{i'} \rangle \equiv \langle \ell : \delta_{i'} \rangle$ which is derivable only if $\Gamma \vdash V : \delta_{i'}$. Then from $\delta_{i'} \leq_D \delta$ we conclude by $(\leq)$. □

Proposition 14.1.6. $\Gamma \vdash s : \sigma \;\;\&\;\; \vdash s = t \Rightarrow \Gamma \vdash t : \sigma$

Proof. By checking axioms in Definition 12.2.2. The only interesting case is when $s \equiv upd_\ell(V, upd_\ell(W, s'))$ and $t \equiv upd_\ell(V, s')$. If $\ell \notin dom(\sigma)$ then $\Gamma \vdash s : \sigma$ iff $\Gamma \vdash s' : \sigma$ iff $\Gamma \vdash t : \sigma$ by Part ((a)) of Lemma 14.1.5.

If $\ell \in dom(\sigma)$ then there exist $\delta \neq \omega_D$ and σ' such that $\sigma = \langle \ell : \delta \rangle \wedge \sigma'$ and $\ell \notin dom(\sigma')$. By the above and that $\Gamma \vdash \sigma \leq \sigma'$, we have that $\Gamma \vdash s' : \sigma'$; by Part ((b)) of Lemma 14.1.5, $\Gamma \vdash V : \delta$, hence $\Gamma \vdash t \equiv upd_\ell(V, s') : \langle \ell : \delta \rangle$ by (upd_1). By Part ((a)) of Lemma 14.1.5 $\Gamma \vdash t : \sigma'$, hence we conclude by $(\wedge)$. □

We are now in place to prove the Subject expansion property of the typing system.

Theorem 14.1.7 (Subject expansion)**.**

$$\Gamma \vdash (N,t) : \kappa \ \ \& \ \ (M,s) \to (N,t) \ \ \Rightarrow \ \ \Gamma \vdash (M,s) : \kappa$$

Proof. The proof is by induction over $(M,s) \to (N,t)$, assuming that $\kappa \neq \omega_C$. The only interesting cases are the following.

Case: $M \equiv [V] \star (\lambda x.M')$ and $N \equiv M'\{V/x\}$ and $s = t$.
By the last part of Lemma 14.1.2, $\exists I, \sigma_i, \kappa_i.\ \forall i \in I.\ \Gamma \vdash N : \sigma_i \to \kappa_i$ & $\Gamma \vdash t : \sigma_i$ & $\bigwedge_{i \in I} \kappa_i \leq \kappa$. By Lemma 14.1.3, for all $i \in I$ there exist δ_i such that $\Gamma \vdash V : \delta_i$ and $\Gamma, x : \delta_i \vdash M' : \sigma_i \to \kappa_i$. Then $\Gamma \vdash [V] : \sigma_i \to \delta_i \times \sigma_i$ by rule (*unit*) and $\Gamma \vdash \lambda x.M' : \delta_i \to \sigma_i \to \kappa_i$ by rule (λ). We conclude that $\Gamma \vdash [V] \star (\lambda x.M') : \sigma \to \kappa$ by rule ($\star$) and ($\wedge$).

Case $(\mathit{get}_\ell(\lambda x.M'), s) \to (M'\{V/x\}, s)$ where $\mathit{lkp}_\ell(s) = V$.
As before, by the last part of Generation Lemma 14.1.2, and by Lemma 14.1.3, $\exists I, \sigma_i, \kappa_i.\ \forall i \in I.\ \Gamma \vdash N : \sigma_i \to \kappa_i$ & $\Gamma \vdash s : \sigma_i$ & $\bigwedge_{i \in I} \kappa_i \leq \kappa$, and for all $i \in I$ there exist δ_i such that $\Gamma \vdash V : \delta_i$ and $\Gamma, x : \delta_i \vdash M' : \sigma_i \to \kappa_i$. Since $\Gamma \vdash V : \delta_i$ we derive by rule (upd_1) that $\Gamma \vdash s = \mathit{upd}_\ell(V, s') : \langle \ell : \delta_i \rangle$, where we can assume w.l.o.g. that $s = \mathit{upd}_\ell(V, s')$ where $l \notin \mathit{dom}(s')$ by Definition 12.2.2. By $\Gamma, x : \delta_i \vdash M' : \sigma_i \to \kappa_i$ and by (*get*) we obtain: $\Gamma \vdash \mathit{get}_\ell(\lambda x.M') : (\langle \ell : \delta_i \rangle \wedge \sigma_i) \to \kappa_i$. By this and $\Gamma \vdash s : \langle \ell : \delta_i \rangle \wedge \sigma_i$, we conclude that $\Gamma \vdash \mathit{get}_\ell(\lambda x.M') : \kappa$ by (*conf*) and ($\leq$).

Case $(\mathit{set}_\ell(V, M'), s) \to (M', \mathit{upd}_\ell(V, s))$.
By the last part of Lemma 14.1.2, $\exists I, \sigma_i, \kappa_i.\ \forall i \in I.\ \Gamma \vdash N : \sigma_i \to \kappa_i$ & $\Gamma \vdash \mathit{upd}_\ell(V, s) : \sigma_i$ & $\bigwedge_{i \in I} \kappa_i \leq \kappa$. We distinguish two cases. Suppose $\ell \notin \mathit{dom}(\sigma_i)$, then

$$\frac{\Gamma \vdash V : \omega_D \quad \Gamma \vdash M' : (\langle \ell : \omega_D \rangle \wedge \sigma_i) \to \kappa_i}{\Gamma \vdash \mathit{set}_\ell(V, M') : \sigma_i \to \kappa_i}\ (\mathit{set})$$

By lemma 14.1.5.(i) we know that $\Gamma \vdash s : \sigma_i$. Hence, the thesis follows by (*conf*) and ($\wedge$).
Otherwise, suppose $\ell \in \mathit{dom}(\sigma_i)$. By lemma 14.1.5.(ii), we have that there exist δ_i such that $\sigma_i \leq \langle \ell : \delta_i \rangle$ and $\Gamma \vdash V : \delta_i$. We can assume wlog $\sigma_i = \langle \ell : \delta_i \rangle \wedge \sigma'_i$ with $\ell \notin \mathit{dom}(\sigma'_i)$:

$$\begin{array}{lll} \Gamma \vdash \mathit{lkp}_\ell(V, s) : \sigma_i & \Rightarrow & \Gamma \vdash \mathit{lkp}_\ell(V, s) : \sigma'_i \\ & \Rightarrow & \Gamma \vdash s : \sigma'_i \qquad\qquad \text{since } \ell \notin \mathit{dom}(\sigma'_i) \text{ and by 14.1.5.(b)} \\ & \Rightarrow & \Gamma \vdash (\mathit{set}_\ell(V, M'), s) : \kappa_i \end{array}$$

The thesis follows by application of (*conf*) and ($\wedge$).

□

$$\dfrac{\dfrac{\Gamma \vdash V : \delta \quad \dfrac{\Gamma \vdash W : \delta'}{\Gamma \vdash [W] : (\langle \ell : \delta \rangle \wedge \sigma) \to \delta' \times (\langle \ell : \delta \rangle \wedge \sigma)}\,(\mathit{unit}) \quad \ell \not\in \mathit{dom}(\sigma)}{\Gamma \vdash \mathit{set}_\ell(V, [W]) : \sigma \to \delta' \times (\langle \ell : \delta \rangle \wedge \sigma)}\,(\mathit{set}) \quad \dfrac{\Gamma, x : \delta \vdash N : \sigma \to \kappa}{\Gamma \vdash \mathit{get}_\ell(\lambda x.N) : (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa}\,(\mathit{get})}{\Gamma \vdash \mathit{set}_\ell(V, [W])\,;\, \mathit{get}_\ell(\lambda x.N) : \sigma \to \kappa}\,(\,;\,)$$

Figure 14.1: Type derivation in Example 14.1.8

Example 14.1.8. In Example 12.2.13 we have seen that

$$(\mathit{set}_\ell(V, [W])\,;\, \mathit{get}_\ell(\lambda x.N), s) \to^* (N\{V/x\}, \mathit{upd}_\ell(V, s))$$

where $M; N \equiv M \star \lambda\_.N$. To illustrate subject reduction, let's first specialize the typing rule $(\star)$ to the case of $M; N$ as follows:

$$\dfrac{\Gamma \vdash M : \sigma \to \delta' \times \sigma' \quad \Gamma \vdash N : \sigma' \to \delta'' \times \sigma''}{\Gamma \vdash M; N : \sigma \to \delta'' \times \sigma''}\,(\,;\,)$$

Now, consider the derivation of $\Gamma \vdash \mathit{set}_\ell(V, [W])\,;\, \mathit{get}_\ell(\lambda x.N) : \sigma \to \kappa$ as in Figure 14.1; therefore, assuming $\Gamma \vdash s : \sigma$ we conclude

$$\Gamma \vdash (\mathit{set}_\ell(V, [W])\,;\, \mathit{get}_\ell(\lambda x.N), s) : \kappa$$

by rule (conf).

The obvious substitution lemma implies that the following rule is admissible:

$$\dfrac{\Gamma \vdash V : \delta \quad \Gamma, x : \delta \vdash N : \sigma \to \kappa}{\Gamma \vdash N\{V/x\} : \sigma \to \kappa}$$

On the other hand, we have that $\ell \not\in \mathit{dom}(\sigma)$ implies that $\sigma = \bigwedge_{i \in I} \langle \ell_i : \delta_i \rangle$ and $\ell \neq \ell_i$ for all $i \in I$. Therefore, by (upd_2) and $(\wedge)$, from the assumption that $\Gamma \vdash s$ we have $\Gamma \vdash \mathit{upd}_\ell(V, s) : \sigma$, hence $\Gamma \vdash (N\{V/x\}, \mathit{upd}_\ell(V, s)) : \kappa$ by (conf).

Notice that the typing of W does not take part to the derivation of $\Gamma \vdash (N\{V/x\}, \mathit{upd}_\ell(V, s)) : \kappa$, which corresponds to the fact that W gets discarded in the reduction from $(\mathit{set}_\ell(V, [W])\,;\, \mathit{get}_\ell(\lambda x.N), s)$ to $(N\{V/x\}, \mathit{upd}_\ell(V, s))$.

14.2 The Characterization Theorem

The main result of the chapter is Theorem 14.2.6 below, where convergent terms are characterized by typeability with a single type in the intersection type system from Section 13.4:

$$\forall M \in \mathit{Com}^0.\ M \Downarrow \quad \Longleftrightarrow \quad \vdash M : \omega_S \to \omega_D \times \omega_S$$

To prove the only-if part it suffices the equivalence of the big-step and the small-step operational semantics, namely reduction, established in Proposition 12.3.4, and Theorems 14.1.4 and 14.1.7; more precisely, the needed part of this theorem is subject expansion, and in particular that if $\Gamma \vdash ([V], t) : \kappa$ and $(M, s) \to^* ([V], t)$ then $\Gamma \vdash (M, s) : \kappa$.

Lemma 14.2.1.

$$\forall M \in \mathit{Com}^0.\ M \Downarrow \quad \Rightarrow \quad \vdash M : \omega_S \to \omega_D \times \omega_S$$

Proof. First note that

$$\begin{array}{lll} M \Downarrow & \Rightarrow & (M, \mathit{emp}) \Downarrow \\ & \Rightarrow & \exists V, t.\ (M, \mathit{emp}) \Downarrow (V, t) \\ & \Rightarrow & \exists V, t.\ (M, \mathit{emp}) \to^* ([V], t) \quad (*) \end{array}$$

by Proposition 12.3.4. Now consider the type derivation:

$$\dfrac{\dfrac{\dfrac{}{\vdash V : \omega_D}\,(\omega)}{\vdash [V] : \omega_S \to \omega_D \times \omega_S}\,(\mathit{unit}) \qquad \dfrac{}{\vdash t : \omega_S}\,(\omega)}{\vdash ([V], t) : \omega_D \times \omega_S}\,(\mathit{conf})$$

By (*) and Theorem 14.1.7, we have that $\vdash (M, \mathit{emp}) : \omega_D \times \omega_S$ so that $\vdash M : \omega_S \to \omega_D \times \omega_S$ since $\omega_D \times \omega_S \neq_C \omega_C$ and, as observed in Section 13.4, $\vdash \mathit{emp} : \omega_S$ is the only typing of *emp* in the empty context up to $=_S$. □

The proof of the if part of Theorem 14.2.6 is more difficult. We adapt to the present calculus the technique of *saturated sets* used in [Kri93] to denote certain sets of terms of ordinary λ-calculus, that are closed by β-expansion. Saturated sets correspond to Tait's computable predicates (see [vB95] Definition 3.2.2) and are called "stable" in [BDS13], §17.2.

Definition 14.2.2 (Saturated sets)**.** *Let $\spadesuit, \clubsuit$ be new symbols and let $M(s) = \spadesuit$ if $(M, s) \Uparrow$. Then define $|\cdot|$ as a map associating to each type a subset of closed terms, stores, or their combinations, depending on the its kind, plus the symbol $\spadesuit, \clubsuit$:*

1. $|\omega_{D_\perp}| = \mathit{Val}^0 \cup \{\clubsuit\}$
2. $|\delta| \subseteq \mathit{Val}^0$ *by* $|\omega_D| = \mathit{Val}^0$, *and*
   $|\delta \to \tau| = \{V \mid \forall W \in |\delta|.\ [W] \star V \in |\tau|\}$
3. $|\sigma| \subseteq \mathit{Store}^0$ *by*
   $|\langle \ell : \omega_{D_\perp} \rangle| = |\omega_S| = \mathit{Store}^0$ *and*
   $|\langle \ell : \delta \rangle| = \{s \mid \ell \in \mathit{dom}(s)\ \&\ \exists V \in |\delta|.\ \mathit{lkp}_\ell(s) = V\}$
   where δ is not an intersection and $\delta \not\equiv \omega_{D_\perp}$,
   $|\langle \ell : \delta \wedge \delta' \rangle| = |\langle \ell : \delta \rangle| \cap |\langle \ell : \delta' \rangle|$
4. $|\kappa| \subseteq (\mathit{Val}^0 \times \mathit{Store}^0) \cup \{\spadesuit\}$ *by*
   $|\omega_C| = (\mathit{Val}^0 \times \mathit{Store}^0) \cup \{\spadesuit\}$ *and* $|\delta \times \sigma| = |\delta| \times |\sigma|$
5. $|\tau| \subseteq \mathit{Com}^0$ *by* $|\omega_{\mathbb{S}D}| = \mathit{Com}^0$ *and*
   $|\sigma \to \kappa| = \{M \mid \forall s \in |\sigma|.\ M(s) \in |\kappa|\}$
6. $|\varphi \wedge \varphi'| = |\varphi| \cap |\varphi'|$ *for φ of any sort.*

Remark 14.2.3. Definition 14.2.2 is of interest on itself, because it provides an interpretation of types in $\mathcal{L}_D$ and $\mathcal{L}_{\mathbb{S}D}$ as sets of terms and of subtyping as the subset relation.

Differently from [Kri93], in the above definition the interpretation of a type does not depend on a mapping I for type variables, since in this setting the only atoms are the ω's. Nonetheless, the interpretation of types δ and τ are, in general, non trivial subsets of Val^0 and Com^0, respectively. First, observe that $|\omega_C| \neq |\delta \times \sigma|$ for all δ and σ, since $\spadesuit \notin |\delta \times \sigma|$. As a consequence, we have that

$$M \in |\sigma \to \delta \times \sigma'| \Leftrightarrow \forall s \in |\sigma|\, \exists (V,t) \in |\delta \times \sigma'|.\, (M,s) \Downarrow (V,t)$$

For example, the interpretation of the type $\omega_D \to (\omega_S \to \omega_D \times \omega_S)$ is

$$\{V \mid \forall M, s.\, ([W] \star V)(s) \in |\omega_D \times \omega_S|\}$$

therefore, $\lambda x.[x] \in |\omega_D \to (\omega_S \to \omega_D \times \omega_S)|$, but $\lambda x.\Omega_c$ does not belong to such set because $([W] \star \lambda x.\Omega_c)(s) = \Omega_c(s) = \spadesuit$ for any W.

Lemma 14.2.4. *For any φ, φ' of any kind, if $\varphi \leq \varphi'$ then $|\varphi| \subseteq |\varphi'|$.*

Proof. By induction over the definition of type pre-orders. □

The saturated sets yield a sound interpretation of type judgments as stated in the next lemma:

Lemma 14.2.5. *Let $\Gamma \vdash M : \tau$ with $\Gamma = \{x_1 : \delta_1, \ldots, x_n : \delta_n\}$ and $M \in \mathit{Com}$. For any $V_1, \ldots, V_n \in \mathit{Val}^0$, if $V_i \in |\delta_i|$ for $i = 1, \ldots, n$ then $M\{V_1/x_1\} \cdots \{V_n/x_n\} \in |\tau|$.*

Proof. By induction over the derivation of $\Gamma \vdash M : \tau$. We abbreviate $\overline{M} \equiv M\{V_1/x_1\} \cdots \{V_n/x_n\}$. The thesis is immediate if the derivation consists just of (*var*), and it follows immediately by induction hypothesis if the derivation ends by either (λ), (*unit*) or $(\wedge)$. In case it ends by $(\leq)$ we use Lemma 14.2.4. The following are the remaining cases.

Case $(\star)$: then $M \equiv M' \star V$, $\tau \equiv \sigma \to \delta'' \times \sigma''$ and the derivation ends by:

$$\frac{\Gamma \vdash M' : \sigma \to \delta' \times \sigma' \quad \Gamma \vdash V : \delta' \to \sigma' \to \delta'' \times \sigma''}{\Gamma \vdash M' \star V : \sigma \to \delta'' \times \sigma''}$$

By induction $\overline{M'} \in |\sigma \to \delta' \times \sigma'|$, hence for all $s \in |\sigma|$ there exists a result $(W, s') \in |\delta' \times \sigma'|$ such that $(\overline{M'}, s) \Downarrow (W, s')$. Now

$$\begin{array}{lll} & (\overline{M'}, s) \Downarrow (W, s') & \\ \Rightarrow & (\overline{M'}, s) \to^* ([W], s') & \text{by Prop. 12.3.4} \\ \Rightarrow & (\overline{M'} \star \overline{V}, s) \to^* ([W] \star \overline{V}, s') & (*) \end{array}$$

Also by induction, we know that $\overline{V} \in |\delta' \to \sigma' \to \delta'' \times \sigma''|$, therefore there exists $(U,t) \in |\delta'' \times \sigma''|$ such that $([W] \star \overline{V}, s') \Downarrow (U,t)$; on the other hand:

$$\begin{array}{lll} & ([W] \star \overline{V}, s') \Downarrow (U,t) & \\ \Rightarrow & ([W] \star \overline{V}, s') \to^* ([U], t) & \text{by Prop. 12.3.4} \\ \Rightarrow & (\overline{M'} \star \overline{V}, s) \to^* ([U], t) & \text{by } (*) \\ \Rightarrow & (\overline{M'} \star \overline{V}, s) \Downarrow (U,t) & \text{by Prop. 12.3.4} \end{array}$$

Then we conclude that $\overline{M' \star V} \equiv \overline{M'} \star \overline{V} \in |\sigma \to \delta'' \times \sigma''|$.

Case (*get*): then $M \equiv get_\ell(\lambda x.M')$, $\tau \equiv (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa$ and the derivation ends by:

$$\frac{\Gamma,\, x : \delta \vdash M' : \sigma \to \kappa}{\Gamma \vdash get_\ell(\lambda x.M') : (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa}$$

Assume $s \in |\langle \ell : \delta \rangle \wedge \sigma| = |\langle \ell : \delta \rangle| \cap |\sigma|$. Since $x : \delta$ is in the premise context, we have $\delta \in \mathcal{L}_D$ by context definition. By induction hypothesis for all $V \in |\delta|$ we have $\overline{M'}\{V/x\} \in |\sigma \to \kappa|$.

We have to prove that $(\overline{get_\ell(\lambda x.M')})(s) \equiv (get_\ell(\lambda x.\overline{M'}))(s) \in |\kappa|$ if $s \in |\langle \ell : \delta \rangle \wedge \sigma| = |\langle \ell : \delta \rangle| \cap |\sigma|$

So we have to handle just two cases: δ is an intersection or not. The most interesting case is when δ is not an intersection. By hypothesis we have:

(a) $s \in |\langle \ell : \delta \rangle| \Rightarrow \ell \in dom(s)\ \ \&\ \ \exists V \in |\delta|.\ lkp_\ell(s) = V$

(b) $s \in |\sigma|\ \ \&\ \ V \in |\delta| \Rightarrow (\overline{M'}\{V/x\})(s) \in |\kappa|$

Since $lkp_\ell(s) = V$, $(get_\ell(\lambda x.\overline{M'}), s) \to (\overline{M'}\{V/x\}, s)$, and by Proposition 12.3.4 we have $(get_\ell(\lambda x.\overline{M'}))(s) = (\overline{M'}\{V/x\})(s) \in |\kappa|$

Now consider the remaining case when δ is an intersection, that is $\delta = \bigwedge_{i \in I} \delta_i$, where every δ_i is not an intersection. By definition we know that $V \in |\delta| \Leftrightarrow \forall i \in I.\ V \in |\delta_i|$. Reasoning as in the previous case, we conclude that $get_\ell(\lambda x.\overline{M'}) \in |\langle \ell_i : \delta_i \rangle \wedge \sigma \to \kappa|$. But $\langle \ell : \delta_i \rangle \wedge \sigma \to \kappa \leq \langle \ell : \bigwedge_{i \in I} \delta_i \rangle \wedge \sigma \to \kappa \equiv \langle \ell : \delta \rangle \wedge \sigma \to \kappa$. We conclude by Lemma 14.2.4.

Case (*set*): then $M \equiv set_\ell(V, M')$, $\tau \equiv \sigma \to \kappa$ and the derivation ends by:

$$\frac{\Gamma \vdash V : \delta \quad \Gamma \vdash M' : (\langle \ell : \delta \rangle \wedge \sigma) \to \kappa \quad \ell \notin dom(\sigma)}{\Gamma \vdash set_\ell(V, M') : \sigma \to \kappa}$$

In this case $\overline{M} \equiv set_\ell(\overline{V}, \overline{M'})$. Now, let $s \in |\sigma|$ be arbitrary: then

$$(set_\ell(\overline{V}, \overline{M'}), s) \to (\overline{M'}, upd_\ell(\overline{V}, s))$$

so that $set_\ell(\overline{V}, \overline{M'})(s) = \overline{M'}(upd_\ell(\overline{V}, s))$. From the side condition $\ell \notin dom(\sigma)$ and the hypothesis $s \in |\sigma|$ we deduce that for some set of indexes J, labels in $dom(\sigma) = \{\ell_j \mid j \in J\}$ and types $\delta_j \in \mathcal{L}_D$:

$$|\sigma| = |\bigwedge_{j \in J} \langle \ell_j : \delta_j \rangle| = \bigcap_{j \in J} |\langle \ell_j : \delta_j \rangle|$$

where $\ell \neq \ell_j$ for all $j \in J$. By the Equation 2 in Definition 12.2.2 we have, for all $j \in J$:

$$lkp_{\ell_j}(upd_\ell(\overline{V}, s)) = lkp_{\ell_j}(s) \in |\delta_j| \text{ and } \ell_j \in dom(s)$$

hence $upd_\ell(\overline{V}, s) \in |\sigma|$ as well. On the other hand

$$lkp_\ell(upd_\ell(\overline{V}, s)) = \overline{V} \in |\delta|$$

by induction, so that $upd_\ell(V, s) \in |\langle \ell : \delta \rangle|$ and hence

$$upd_\ell(V, s) \in |\langle \ell : \delta \rangle| \cap |\sigma| = |\langle \ell : \delta \rangle \wedge \sigma|$$

It follows that

$$set_\ell(\overline{V}, \overline{M'})(s) = \overline{M'}(upd_\ell(\overline{V}, s)) \in |\kappa|$$

since $\overline{M'} \in |(\langle \ell : \delta \rangle \wedge \sigma) \to \kappa|$ by induction.

□

Theorem 14.2.6 (Characterization of convergence)**.**

$$\forall M \in Com^0.\ M \Downarrow \quad \Longleftrightarrow \quad \vdash M : \omega_S \to \omega_D \times \omega_S$$

Proof. The only-if part is Lemma 14.2.1. To show the if part, by Lemma 14.2.5 $\vdash M : \omega_S \to \omega_D \times \omega_S$ implies $M \in |\omega_S \to \omega_D \times \omega_S|$, where the typing context $\Gamma = \emptyset$ and no substitution are considered since M is closed.

Recall that $|\omega_S| = Store^0$ and $|\omega_D \times \omega_S| = Val^0 \times Store^0$. Therefore, for any $s \in Store^0$ we have $M(s) \in |\omega_D \times \omega_S|$, namely there exist $V \in Val^0$ and $t \in Store^0$ such that $(M, s) \Downarrow (V, t)$, hence $M \Downarrow$. □

CHAPTER 15

CONCLUSIONS AND RELATED WORK

15.1 Discussion and Related Work

The calculus and its monadic semantics But for the operators get_ℓ and set_ℓ, the calculus syntax is the same as in Part I, where we considered a pure untyped computational λ-calculus, namely without operations nor constants. Therefore, the monad T and the respective unit and bind were generic, so that types cannot convey any specific information about the domain TD, nor about effects represented by the monad. However, the reduction relation in Section 12.2 is strictly included in that one considered there, which is the compatible closure of the monadic laws from [Wad95], oriented from left to right. In contrast, here we just retain rule (β_c) and take the closure under rule ($\star$-*red*). On the other hand, we have shown in Part I that the reduction rules corresponding to the (right) identity and associativity of the bind operator are not essential when considering convergent terms.

The algebraic operators get_ℓ and set_ℓ come from Plotkin and Power [PP02, PP03, Pow06]. The algebra of store terms is inspired to [PP02], where the store monad in [Mog91] is generated by the update and lookup operations. Such a construction, however, does not perfectly match to ours, because here the set $\mathbf{L}$ of locations is infinite.

We have borrowed the notation for get_ℓ and set_ℓ from the "imperative λ-calculus" in chapter 3 of [Gav19], where also a definition of the convergence predicate of a configuration to a result is considered. Such a definition is a particular case of the analogous notion in [DGL17, LG19] for generic algebraic effects. It is stated in semantic terms, while we preferred the syntactical treatment in the algebra of store terms in Section 12.2.

Intersection types and computational effects Intersection types are an extension of Curry simple types, whose intended meaning is that of predicates of untyped terms. In particular, intersection types embody a form of ad hoc-polymorphism, where one may consider the conjunction of semantically unrelated types. As an instance of the Leibnitz's law of identity of indiscernibles, terms can be identified with the set of their properties, namely types. As a consequence, updating the value associated to a location may radically change the types of the store itself.

On the other hand, intersection types are a description of the value of a term, which in our setting is the result of evaluating a configuration. This is the proper reading of the typing rule (*set*): in the type of M we do not keep track of all the intermediate changes of the store s when evaluating (M, s). Rather we foresee the property of the final store in the result of (M, s), if any, under the assumption that s satisfies the hypothesis expressed in the antecedent of the arrow type of M.

Our store types are not reference types, and indeed we do not consider the locations among the values, nor we have a construct for dynamic allocation. This makes difficult the comparison to [DP00] and to the subsequent [DCRDR07]. In Pfenning's and others work, intersection types are added to the Hindley-Milner type system for ML to enhance type expressivity and to strengthen polymorphism. However, the resulting system is unsound, which forces the restriction of intersection types to values and the loss of subtyping properties, that are essentially those in Lemma 14.1.1. The issue is due to reference types in ML, where the type of a location is its "intrinsic" type in the sense of Reynolds [Rey00]. In contrast, our store types are predicates over the stores, namely "extrinsic" types, telling what are the meanings of the values associated to the locations in the store.

Type and effect systems A further line of research is to use our type system to investigate a semantic understanding of type and effect systems, introduced in [GL86] and pursued in [TJ94]. In the insightful paper [WT03] a type and effect judgement $\Gamma \vdash e : A, \varepsilon$ is translated into and ordinary typing $\Gamma \vdash e : T^{\varepsilon}A$, where T is a monad. This has fostered the application of type systems with monadic types to static analysis for code transformation in [BKHB06, BKBH09], but also raised the question of the semantics of the types $T^{\varepsilon}A$.

In the papers by Benton and others, the semantics of monadic types with effect decorations is given in terms of PERs that are preserved by read and write operations. Such a semantics validates equations that do hold under assumptions about the effects of the equated programs; e.g. only pure terms, neither depending on the store nor causing any mutation, can be evaluated in arbitrary order, or repeated occurrences of the same pure code can be replaced by a reference where the value of the code is stored after computing it just once. Such properties are nicely reflected in our types: if $\lambda x.M$ has type $\delta \to \sigma \to \delta' \times \sigma$, and $dom(\sigma)$ includes all the ℓ occurring in M, then we know that the function represented by $\lambda x.M$ is pure; similarly if $M : \sigma \to \delta \times \sigma$ and $N : \sigma \to \delta' \times \sigma$ then for any $P : \sigma \to \kappa$ both $M; N; P$ and $N; M; P$ will have the same types, and hence the same behaviour. In general this suggests how to encode regions with store types in our system.

Filter-model construction Since [BCD83] we know that a λ-model can be constructed by taking the filters of types in an intersection type system with subtyping. The relation among the filter-model and Scott's D_∞ construction has been subject to extensive studies, starting with [CDHL84] and continuing with [DHA03, ADCH04, ABD06, AS08]. In the meantime Abramsky's theory of domain logic in [Abr91] provided a generalization of the same ideas to algebraic domains in the category of 2/3 SFP, of which ω-**ALG** is a (full) subcategory, based on Stone duality.

A further research direction is to move from ω-**ALG** to other categories such as the category of relations. The latter is deeply related to non-idempotent intersection types [dC18, BEM07] that have been shown to catch intensional aspects of λ-calculi involving quantitative reasoning about computational resources. If the present approach can be rephrased in the category of relations, then our method could produce non-idempotent intersection type systems for effectful λ-calculi.

In the perspective of considering categories other than ω-**ALG**, it is natural to ask whether the synthesis of an "estrinisic" type system in the sense of Reynolds out of the denotational semantics of a calculus, either typed or not, can be described in categorical terms.

Refinement, essential, and non idempotent intersection types Another view of intersection types is as refined types of ordinary types (see [Abr91]), so that conjunction makes sense only among subtypes of the same type. This seems in contrast with the main example in the literature which is the type $(\sigma \wedge (\sigma \to \tau)) \to \tau$ of the term $\lambda x.xx$. However, both σ and $\sigma \to \tau$ are subtypes of $\omega = \omega \to \omega$ in [BCD83], which is the counterpart of Scott's domain equation $D = D \to D$. Here we have the equations $\omega_D = \omega_D \to \omega_{\S D}$ and $\omega_{\S D} = \omega_S \to \omega_C$, representing the solution of the domain equation $D = D \to TD$, where T is (a variant of) the state monad in [Mog91]. The study of type interpretation in the category of models of computational λ-calculi deserves further investigation.

As already mentioned, our type system is inspired by [BCD83], where the intersection types are related by the subtyping relation. It is known that subtyping can be avoided e.g. in the "essential" system in [vB95], which has syntax directed rules. Compared to the system in [BCD83], the essential intersection type system is equally powerful w.r.t. the characterization of strongly normalizable λ-terms and other similar properties, but it is unrelated to the Scott D_∞ model of the λ-calculus, which is instead isomorphic to the filter-model in [BCD83]. In general, the correspondence exploited in [Abr91] among type theories and domains gets lost in case of essential types, describing webbed-models like Engeler's. On the other hand the essential type system is undecidable right because its expressive power, as it is the case of ours because of Theorem 14.2.6.

Another family of intersection type systems have been introduced in [dC18]. Moving from Engeler's model, De Carvalho obtains a system where intersection is not idempotent, that is σ is not a subtype of $\sigma \wedge \sigma$. This provides an intensional type system that is sensible to the temporal complexity of the reduction of terms to normal form. It is a natural development of our work to design a non idempotent system for the side-effects with higher-order stores.

Towards a categorical perspective A development of the present work would be a method for synthesizing intersection type systems for a computational λ-calculus with algebraic operators for generic effects in the sense of [PP03]. Such a process should be described in categorical terms; indeed, while it is well known how to present the denotational semantics of the calculus as a functor into a suitable category of meanings, it has been shown in [MZ15] that a system like ours is itself a particular functor. A natural question is whether the latter functor can be uniformly obtained from the categorical semantics of the calculus.

15.2 Conclusion

In this part we presented a type assignment system as a way to study the semantics of an imperative computational λ-calculus equipped with global store and algebraic operators. The system defines the semantics of the untyped calculus, and we obtained a type theoretic characterization of convergence.

Deriving the type system from semantics as in Section 13.3 is the first contribution of this part. In the present work, we exploited denotational semantics of type systems, in general, but reversing the process from semantics to types. Usually, one starts with a calculus and a type system, looking for a semantics and studying properties of the model. On the contrary, we move from a domain equation and the definition of the denotational semantics of the calculus of interest and synthesize a filter-model and an intersection type system. This synthetic use of domain logic is, in our view, prototypical w.r.t. the construction of logics catching properties of any computational λ-calculus with operators. We expect that the study of such type systems will be of help in understanding the operational semantics of such calculi, a topic that has been addressed in [DGL17, LG19], but with different mathematical tools. Indeed, addressing the investigation in such a way, we smoothly obtain the soundness and, possibly, the completeness.

The second result is that (closed) terms that are meant to be convergent w.r.t. their natural operational semantics can be characterized via their typings in the system. More precisely there exists exactly one type that can be assigned to all convergent terms, which is akin to the lazy λ-calculus [AO93], where convergent terms are characterized by the type $\omega \to \omega$ in the endogenous logic of the calculus, in the sense of [Abr91].

OVERALL CONCLUSIONS

Since Moggi's seminal papers [Mog89, Mog91], a substantial body of research has been carried out about the computational λ-calculus and usage of the concept of monad, both in theory and in practice of functional programming languages.

There is still a long way to go before we have a full understanding of what has been presented in this thesis. The need for generalisation is evident from both a syntactic and a semantic point of view.

Syntactic knowledge. In the beginning of our investigation, a strong reduction was considered (i.e. rewriting steps could occur anywhere in the term) and then, via surface reduction, a weak one, which is deterministic. The results achieved in Part I motivate the weak reduction considered in the case of the state monad. This turning of interest is justified by the fact that, when interested in returning a value, weak β_c steps suffice. In addition, the theoretical result mirrors the actual evaluation of programs, for example, in Haskell. But what is a general pattern that one has to follow when it comes with a particular instance of the monad?

On a syntactic level, we should understand what it means to add generic operations to monadic constructors and, consequently, which is the shape of the reduction rules when modelling generic effects. After that the first question that arises is what kind of contextual closure would be taken into account.

In general, there is the need of reaching a top-down view of these reductions: when to be more permissive and, instead, when to force effects to be produced just at root level. A strictly connected issue is the generalisation of the factorization results for generic effects.

From general to particular and back. We have discussed the monadic interpretation in Part II. This interpretation has the defect of not addressing generic effects, and therefore generic algebraic operators. This is difficult to do, since one has to comprehend a multiplicity of behaviours via a purely equational axiomatization.

In the thesis the plot has consisted in considering a generic monad and then instantiate the calculus for a particular, interesting case. Now the time to turn back has come: using the insights gained through the study of state monad and applying them in the case of the generic one.

Comonadic treatment. Sometimes, monadic aspects are combined with comonadic ones. It is therefore natural to consider a λ-calculus modelling co-computations over a generic comonad $\mathbb{C}$. This involves defining and solving a new domain equation $D = \mathbb{C}D \to D$ from which to extract a calculus and a type assignment system, following a *modus operandi* similar to the one adopted in this thesis in Part III. The final goal is to combine the computational core with the obtained comonadic calculus.

Switching categories. A natural question is how to move from ω-$\mathbf{ALG}$, and from Scott's domains, to other categories of domains; surely the case of the category of relations is interesting. In fact, *REL* is deeply related to non-idempotent intersection types that have been shown to catch intensional aspects of λ-calculi involving quantitative reasoning about computational resources. We expect that the process presented in Part III, if worked out in this category, would bring to an non-idempotent intersection type discipline, that in the case of probabilistic monad, for example, is surely more proper. In the end, we aim to produce non-idempotent intersection type systems for generic effectful λ-calculi.

Refinement types. Intersection types were born as an extension of the Curry system for the untyped λ-calculus. Reynolds and Pfenning viewed intersection types as *extrinsic* types, namely as refinement of ordinary, *intrinsic* types. In Abramsky's domain logic, refinement types denote compact points of the interpretations of the ordinary types that they refine. If we interpret the D_∞ construction as the denotation of a recursive type, intersection types in BCD-like systems are refinement types of that type. This is the key to distil a type system from a domain equation. Can such a process be framed in categorical terms, where refinement type system is a functor, as proposed in [MZ15]?

BIBLIOGRAPHY

[ABD06] F. Alessi, F. Barbanera, and M. Dezani-Ciancaglini. Intersection types and lambda models. *Theor. Comput. Sci.*, 355(2):108–126, 2006.

[Abr90] S. Abramsky. The lazy lambda calculus. In *Research topics in functional programming*, pages 65–116. Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA, 1990.

[Abr91] S. Abramsky. Domain theory in logical form. *Ann. Pure Appl. Log.*, 51(1-2):1–77, 1991.

[AC98] R. Amadio and P.-L Curien. *Domains and lambda-calculi.* Cambridge University Press, 1998.

[Acc12] B. Accattoli. Proof nets and the call-by-value lambda-calculus. 113:11–26, 2012.

[ADCH04] F. Alessi, M. Dezani-Ciancaglini, and F. Honsell. Inverse Limit Models as Filter Models. In Delia Kesner, Femke van Raamsdonk, and Joe Wells, editors, *HOR'04*, pages 3–25, Aachen, 2004. RWTH Aachen.

[AFG21] B. Accattoli, C. Faggian, and G. Guerrieri. Factorize factorization. In *29th EACSL Annual Conference on Computer Science Logic, CSL 2021*, volume 183 of *LIPIcs*, pages 6:1–6:25. Schloss-Dagstuhl, 2021.

[AFM$^+$95] Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. The call-by-need lambda calculus. In *Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1995*, pages 233–246. ACM Press, 1995.

[AG16] B. Accattoli and G. Guerrieri. Open call-by-value. In *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, volume 10017 of *Lecture Notes in Computer Science*, pages 206–226. Springer, 2016.

[AJ94] S. Abramsky and A. Jung. Domain theory. In Samson Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science (Vol. 3): Semantic Structures*, pages 1–168. Oxford University Press, Inc., 1994.

[AO93] S. Abramsky and C.-H.L. Ong. Full abstraction in the lazy lambda calculus. *Inf. Comput.*, 105(2):159–267, 1993.

[AP12] B. Accattoli and L. Paolini. Call-by-value solvability, revisited. In *Functional and Logic Programming - 11th International Symposium, FLOPS 2012*, volume 7294 of *Lecture Notes in Computer Science*, pages 4–16. Springer, 2012.

[AS08] F. Alessi and P. Severi. Recursive Domain Equations of Filter Models. In V. Geffert, J. Karhumäki, A. Bertoni, B. Preneel, Pavol Návrat, and M. Bieliková, editors, *SOFSEM 2008: Theory and Practice of Computer Science, 34th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 19-25, 2008, Proceedings*, volume 4910 of *Lecture Notes in Computer Science*, pages 124–135. Springer, 2008.

[Bar85] H. Barendregt. *The Lambda Calculus: its Syntax and Semantics*, volume 103 of *Studies in logic and the foundations of mathematics.* North-Holland, revised edition, 1985.

[BCD83] H. P. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *J. Symb. Log.*, 48(4):931–940, 1983.

[BDS13] H. P. Barendregt, W. Dekkers, and R. Statman. *Lambda Calculus with Types.* Perspectives in logic. Cambridge University Press, 2013.

[BEM07] A. Bucciarelli, T. Ehrhard, and G. Manzonetto. Not enough points is enough. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 298–312. Springer, 2007.

[BHM02] N. Benton, J. Hughes, and E. Moggi. Monads and effects. In *Applied Semantics, International Summer School, APPSEM 2000*, volume 2395 of *Lecture Notes in Computer Science*, pages 42–122. Springer, 2002.

[BKBH09] N. Benton, A. Kennedy, L. Beringer, and M. Hofmann. Relational semantics for effect-based program transformations: higher-order store. In António Porto and Francisco Javier López-Fraguas, editors, *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 7-9, 2009, Coimbra, Portugal*, pages 301–312. ACM, 2009.

[BKHB06] N. Benton, A. Kennedy, M. Hofmann, and L. Beringer. Reading, writing and relations. In *Programming Languages and Systems, 4th Asian Symposium, APLAS 2006, Sydney, Australia, November 8-10, 2006, Proceedings*, volume 4279 of *Lecture Notes in Computer Science*, pages 114–130. Springer, 2006.

[BN98] F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

[CDC80] M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame J. Formal Log.*, 21:685–693, 1980.

[CDHL84] M. Coppo, M. Dezani-Ciancaglini, F. Honsell, and G. Longo. Extended type structures and filter lambda models. In G. Lolli, G. Longo, and A. Marcja, editors, *Logic Colloquium 82*, pages 241–262, Amsterdam, the Netherlands, 1984. North-Holland.

[CF93] M. Coppo and A. Ferrari. Type inference, abstract interpretation and strictness analysis. *Theor. Comput. Sci.*, 121(1&2):113–143, 1993.

[CG14] A. Carraro and G. Guerrieri. A semantical and operational account of call-by-value solvability. In *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Proceedings*, volume 8412 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 2014.

[Cio13] S. Ciobaca. From small-step semantics to big-step semantics, automatically. In Einar Broch Johnsen and Luigia Petre, editors, *Integrated Formal Methods, 10th International Conference, IFM 2013, Turku, Finland, June 10-14, 2013. Proceedings*, volume 7940 of *Lecture Notes in Computer Science*, pages 347–361. Springer, 2013.

[dC18] D. de Carvalho. Execution time of λ-terms via denotational semantics and intersection types. *Math. Struct. Comput. Sci.*, 28(7):1169–1203, 2018.

[DCRDR07] M. Dezani-Ciancaglini and S. Ronchi Della Rocca. Intersection and Reference Types. In *Reflections on Type Theory, Lambda Calculus, and the Mind*, pages 77–86. Radboud University Nijmegen, 2007.

[DGL17] U. Dal Lago, F. Gavazzo, and P. B. Levy. Effectful Applicative Bisimilarity: Monads, Relators, and Howe's Method. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12. IEEE Computer Society, 2017.

[DHA03] M. Dezani-Ciancaglini, F. Honsell, and F. Alessi. A complete characterization of complete intersection-type preorders. *ACM Trans. Comput. Log.*, 4(1):120–147, 2003.

[DMRU12] B. Düdder, M. Martens, J. Rehof, and P. Urzyczyn. Bounded combinatory logic. In *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012*, volume 16 of *LIPIcs*, pages 243–258. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[DP00] R. Davies and F. Pfenning. Intersection types and computational effects. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00)*, pages 198–208. ACM, 2000.

[DR17a] A. Dudenhefner and J. Rehof. Intersection type calculi of bounded dimension. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017*, pages 653–665. ACM, 2017.

[DR17b] A. Dudenhefner and J. Rehof. Typability in bounded dimension. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12. IEEE Computer Society, 2017.

[dT19] U. de'Liguoro and R. Treglia. Intersection types for the computational lambda-calculus. *CoRR*, abs/1907.05706, 2019.

[dT20] U. de'Liguoro and R. Treglia. The untyped computational λ-calculus and its intersection type discipline. *Theor. Comput. Sci.*, 846:141–159, 2020.

[dT21a] U. de'Liguoro and R. Treglia. From semantics to types: the case of the imperative lambda-calculus. In Ana Sokolova, editor, Proceedings 37th Conference on *Mathematical Foundations of Programming Semantics,* Hybrid: Salzburg, Austria and Online, 30th August - 2nd September, 2021, volume 351 of *Electronic Proceedings in Theoretical Computer Science*, pages 168–183. Open Publishing Association, 2021.

[dT21b] U. de'Liguoro and R. Treglia. Intersection types for a λ-calculus with global store. In Niccolò Veltri, Nick Benton, and Silvia Ghilezan, editors, *PPDP 2021: 23rd International Symposium on Principles and Practice of Declarative Programming, Tallinn, Estonia, September 6-8, 2021*, pages 5:1–5:11. ACM, 2021.

[EG16] T. Ehrhard and G. Guerrieri. The bang calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. In *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming (PPDP 2016)*, pages 174–187. ACM, 2016.

[Ehr12] T. Ehrhard. Collapsing non-idempotent intersection types. In *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012*, volume 16 of *LIPIcs*, pages 259–273. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[EMS09] J. Egger, R. E. Møgelberg, and A. Simpson. Enriching an effect calculus with linear types. In Erich Grädel and Reinhard Kahle, editors, *Computer Science Logic, 23rd international Workshop, CSL 2009, 18th Annual Conference of the EACSL. Proceedings*, volume 5771 of *Lecture Notes in Computer Science*, pages 240–254. Springer, 2009.

[Fel88] M. Felleisen. The theory and practice of first-class prompts. In *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '88, page 180–190, New York, NY, USA, 1988. Association for Computing Machinery.

[FF89] M. Felleisen and D. P. Friedman. A syntactic theory of sequential state. *Theor. Comput. Sci.*, 69(3):243–287, 1989.

[FG21] C. Faggian and G. Guerrieri. Factorization in call-by-name and call-by-value calculi via linear logic. In *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021,* *Proceedings*, volume 12650 of *Lecture Notes in Computer Science*, pages 205–225. Springer, 2021.

[Fil94] A. Filinski. Representing monads. In *Conference Record of POPL'94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 446–457. ACM Press, 1994.

[Gav19] F. Gavazzo. *Coinductive Equivalences and Metrics for Higher-order Languages with Algebraic Effects.* PhD thesis, University of Bologna, Italy, Aprile 2019.

[GL86] D. K. Gifford and J. M. Lucassen. Integrating functional and imperative programming. In *Proceedings of the 1986 ACM Conference on LISP and Functional Programming, LFP 1986, August 4-6, 1986, Cambridge, Massachusetts, USA*, pages 28–38. ACM, 1986.

[GM19] G. Guerrieri and G. Manzonetto. The bang calculus and the two Girard's translations. In *Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications (Linearity-TLLA 2018)*, volume 292 of *EPTCS*, pages 15–30, 2019.

[God58] R. Godement. *Topologie algébrique et théorie des faisceaux.* Number v. 1 in Actualités scientifiques et industrielles. Hermann, 1958.

[GPR17] G. Guerrieri, L. Paolini, and S. Ronchi Della Rocca. Standardization and conservativity of a refined call-by-value lambda-calculus. *Log. Methods Comput. Sci.*, 13(4), 2017.

[Gue15] G. Guerrieri. Head reduction and normalization in a call-by-value lambda-calculus. In *2nd International Workshop on Rewriting Techniques for Program Transformations and Evaluation, WPTE 2015*, volume 46 of *OASICS*, pages 3–17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[Gue19] G. Guerrieri. Towards a semantic measure of the execution time in call-by-value lambda-calculus. In *Proceedings Twelfth Workshop on Developments in Computational Models and Ninth Workshop on Intersection Types and Related Systems, DCM/ITRS 2018*, volume 293 of *EPTCS*, pages 57–72, 2019.

[Ham18] M. Hamana. Polymorphic rewrite rules: Confluence, type inference, and instance validation. In *Functional and Logic Programming - 14th International Symposium, FLOPS 2018, Proceedings*, volume 10818 of *Lecture Notes in Computer Science*, pages 99–115. Springer, 2018.

[Hin64] J.R. Hindley. *The Church-Rosser Property and a Result in Combinatory Logic.* PhD thesis, University of Newcastle-upon-Tyne, 1964.

[HP06] M. Hyland and J. Power. Discrete lawvere theories and computational effects. *Theor. Comput. Sci.*, 366(1-2):144–162, 2006.

[HP07] M. Hyland and J. Power. The category theoretic understanding of universal algebra: Lawvere theories and monads. *Electron. Notes Theor. Comput. Sci.*, 172:437–458, 2007.

[HZ09] H. Herbelin and S. Zimmermann. An operational account of call-by-value minimal and classical lambda-calculus in ”natural deduction” form. In Pierre-Louis Curien, editor, *Typed Lambda Calculi and Applications, 9th International Conference, TLCA 2009. Proceedings*, volume 5608 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2009.

[Jen95] T.P. Jensen. Conjunctive Type Systems and Abstract Interpretation of Higher-order Functional Programs. *Journal of Logic and Computation*, 5(4):397–421, 1995.

[Kri93] J.-L Krivine. *Lambda calculus, types and models.* Ellis Horwood, 1993.

[Ler90] X. Leroy. The ZINC experiment: an economical implementation of the ML language. Technical report 117, INRIA, 1990.

[Lev99] P. B. Levy. Call-by-push-value: A subsuming paradigm. In *Typed Lambda Calculi and Applications, 4th International Conference (TLCA'99)*, volume 1581 of *Lecture Notes in Computer Science*, pages 228–242, 1999.

[LG88] J. M. Lucassen and D. K. Gifford. Polymorphic effect systems. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, San Diego, California, USA, January 10-13, 1988*, pages 47–57. ACM Press, 1988.

[LG19] U. Dal Lago and F. Gavazzo. Effectful normal form bisimulation. In Luís Caires, editor, *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11423 of *Lecture Notes in Computer Science*, pages 263–292. Springer, 2019.

[LM08] U. Dal Lago and S. Martini. The weak lambda calculus as a reasonable machine. *Theor. Comput. Sci.*, 398(1-3):32–50, 2008.

[Mac97] S. MacLane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer, 2 edition, 1997.

[Mey82] A. R. Meyer. What is a model of the lambda calculus? *Inf. Control.*, 52(1):87–122, 1982.

[Mit96] J.C. Mitchell. *Foundations for Programming Languages.* MIT Press, Cambridge, MA, 1996.

[Mog88] E. Moggi. Computational Lambda-calculus and Monads. Report ECS-LFCS-88-66, University of Edinburgh, Edinburgh, Scotland, 1988.

[Mog89] E. Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89)*, pages 14–23. IEEE Computer Society, 1989.

[Mog91] E. Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.

[MOTW99] J. Maraist, M. Odersky, D. N. Turner, and P. Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Theor. Comput. Sci.*, 228(1-2):175–210, 1999.

[MZ15] P.-A. Melliès and N. Zeilberger. Functors are type refinement systems. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015*, pages 3–16. ACM, 2015.

[New42] M.H.A. Newman. On theories with a combinatorial definition of equivalence. *Annals of Mathematics*, 43(2), 1942.

[NNH99] F. Nielson, H. Riis Nielson, and C. Hankin. *Principles of program analysis.* Springer, 1999.

[Plo75] G. D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975.

[Plo04] G. D. Plotkin. A structural approach to operational semantics. *J. Log. Algebraic Methods Program.*, 60-61:17–139, 2004.

[Pow00] J. Power. Models for the computational lambda-calculus. *Electron. Notes Theor. Comput. Sci.*, 40:288–301, 2000.

[Pow06] J. Power. Generic models for computational effects. *Theor. Comput. Sci.*, 364(2):254–269, 2006.

[PP02] G. D. Plotkin and J. Power. Notions of computation determine monads. In *FOSSACS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 342–356. Springer, 2002.

[PP03] G. D. Plotkin and J. Power. Algebraic operations and generic effects. *Appl. Categorical Struct.*, 11(1):69–94, 2003.

[PS98] A. Pitts and I. Stark. Operational reasoning for functions with local state. In *In Higher Order Operational Techinques in Semantics*, pages 227–273. Cambridge University Press, 1998.

[Rey00] J.C. Reynolds. An Intrinsic Semantics of Intersection Types. In *Electronic Proceedings of 3rd International Workshop* Intersection Types and Related Systems *(ITRS'04), Turku, Finland*, pages 269–270, 2000.

[Sco80] D. Scott. Relating theories of the λ-calculus. In R. J. Hindley and J. P. Seldin, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 403–450. Academic Press, 1980.

[SF93] A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. *LISP Symb. Comput.*, 6(3-4):289–360, 1993.

[Sim05] A. Simpson. Reduction in a linear lambda-calculus with applications to operational semantics. In Jürgen Giesl, editor, *Term Rewriting and Applications*, pages 219–234, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[SRI91] V. Swarup, U. S. Reddy, and E. Ireland. Assignments for applicative languages. In John Hughes, editor, *Functional Programming Languages and Computer Architecture, 5th ACM Conference, Cambridge, MA, USA, August 26-30, 1991, Proceedings*, volume 523 of *Lecture Notes in Computer Science*, pages 192–214. Springer, 1991.

[SW97] A. Sabry and P. Wadler. A reflection on call-by-value. *ACM Trans. Program. Lang. Syst.*, 19(6):916–941, 1997.

[Tak95] M. Takahashi. Parallel reductions in lambda-calculus. *Inf. Comput.*, 118(1):120–127, 1995.

[Ter03] Terese. *Term rewriting systems*, volume 55 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, 2003.

[TJ94] J.-P. Talpin and P. Jouvelot. The type and effect discipline. *Inf. Comput.*, 111(2):245–296, 1994.

[Tof90] M. Tofte. Type inference for polymorphic references. *Inf. Comput.*, 89(1):1–34, 1990.

[vB95] S. van Bakel. Intersection Type Assignment Systems. *Theoretical Computer Science*, 151(2):385–435, 1995.

[vO94] V. van Oostrom. Confluence by decreasing diagrams. *Theor. Comput. Sci.*, 126(2):259–280, 1994.

[vO20a] V. van Oostrom. Private communication via electronic mail, 2020.

[vO20b] V. van Oostrom. Some symmetries of commutation diamonds. In Mauricio Ayala-Rincon and Samuel Mimram, editors, *Proceedings of the 9th International Workshop on Confluence*, pages 1–5, 2020.

[Wad92] P. Wadler. The essence of functional programming. In *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1992*, pages 1–14. ACM Press, 1992.

[Wad95] P. Wadler. Monads for functional programming. In *Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques*, volume 925 of *Lecture Notes in Computer Science*, pages 24–52. Springer, 1995.

[Wad98] P. Wadler. The marriage of effects and monads. In *Proceedings of the third ACM SIGPLAN International Conference on Functional Programming (ICFP '98*, pages 63–74. ACM, 1998.

[WF94] A. K. Wright and M. Felleisen. A syntactic approach to type soundness. *Inf. Comput.*, 115(1):38–94, 1994.

[WT03] P. Wadler and P. Thiemann. The marriage of effects and monads. *ACM Trans. Comput. Log.*, 4(1):1–32, 2003.

APPENDIX A

GENERAL PROPERTIES OF THE CONTEXTUAL CLOSURE

Shape Preservation. We start by recalling a basic but key property of contextual closure. If a step $\rightarrow_\gamma$ is obtained by closure under *non-empty context* of a rule $\mapsto_\gamma$, then it preserves the shape of the term. We say that T and T' have *the same shape* if both terms are an application (resp. an abstraction, an variable, a term of shape $!P$).

Fact 4.3.2 (Shape preservation)**.** *Let $\mapsto_\rho$ be a rule and $\rightarrow_\rho$ be its contextual closure. Assume $T = \mathsf{C}\langle R\rangle \rightarrow_\rho \mathsf{C}\langle R'\rangle = T'$ and that the context C is* non-empty*. Then T and T' have the same shape.*

Note that a root steps $\mapsto$ is both a *weak* and a *surface* step.

The implication in the previous lemma cannot be reversed as the following example shows:

$$M = V(\mathbf{I}P) \rightarrow_\iota VP = N$$

M is a σ-redex, but N is not.

Substitutivity. A relation $\hookrightarrow$ on terms is *substitutive* if

$$R \hookrightarrow R' \text{ implies } R\{Q/x\} \hookrightarrow R'\{Q/x\}. \qquad (\textbf{substitutive})$$

An obvious induction on the shape of terms shows the following ([Bar85] p. 54).

Fact A.0.1 (Substitutive)**.** *Let $\rightarrow_\gamma$ be the contextual closure of $\mapsto_\gamma$.*

1. *If $\mapsto_\gamma$ is substitutive then $\rightarrow_\gamma$ is substitutive: $T \rightarrow_\gamma T'$ implies $T\{Q/x\} \rightarrow_\gamma T'\{Q/x\}$.*

2. *If $Q \rightarrow_\gamma Q'$ then $T\{Q/x\} \rightarrow_\gamma^* T\{Q'/x\}$.*

A.1 Properties of the Syntax $\Lambda^!$

In this section, we consider the set of terms $\Lambda^!$ (the same syntax as the *full* bang calculus, as defined in Section 3.2.1), endowed with a generic reduction $\to_\rho$ (from a generic rule $\mapsto_\rho$). We study some properties that hold in general in $(\Lambda^!, \to_\rho)$.

Terms are generated by the grammar:

$$T, S, R ::= x \mid ST \mid \lambda x.T \mid !T \qquad\qquad (\textbf{terms } \Lambda^!)$$

Contexts (C), *surface contexts* (S) and *weak contexts* (W) are generated by the grammars:

$$\begin{aligned} \mathsf{C} &::= \langle\rangle \mid T\mathsf{C} \mid \mathsf{C}T \mid \lambda x.\mathsf{C} \mid !\mathsf{C} && (\textbf{contexts}) \\ \mathsf{S} &::= \langle\rangle \mid T\mathsf{S} \mid \mathsf{S}T \mid \lambda x.\mathsf{S} && (\textbf{surface contexts}) \\ \mathsf{W} &::= \langle\rangle \mid T\mathsf{W} \mid \mathsf{W}T \mid !\mathsf{W} && (\textbf{weak contexts}) \end{aligned}$$

If $\mapsto_\rho$ is a rule, the reduction $\to_\rho$ its the closure under context C. *Surface reduction* $\underset{\mathsf{s}}{\to}_\rho$ (resp. *weak reduction* $\underset{\mathsf{w}}{\to}_\rho$) is the closure of $\mapsto_\rho$ under surface contexts S (resp. weak contexts W). *Non-surface reduction* $\underset{\neg\mathsf{s}}{\to}_\rho$ (resp. *non-weak reduction* $\underset{\neg\mathsf{w}}{\to}_\rho$) is the closure of $\mapsto_\rho$ under contexts C that are not surface (resp. not weak).

A.1.1 Shape Preservation for Internal Steps in $\Lambda^!$

Fact 4.3.2 (p. 39) implies that $\underset{\neg\mathsf{s}}{\to}_\rho$ and $\underset{\neg\mathsf{w}}{\to}_\rho$ steps always preserve the shape of terms. We recall that we write $\mapsto_\rho$ to indicate the step $\to_\rho$ obtained by *empty contextual closure*. The following property immediately follows from Fact 4.3.2.

Fact A.1.1 (Internal Steps)**.** *Let $\mapsto_\rho$ be a rule and $\to_\rho$ be its contextual closure. $\underset{\neg\mathsf{w}}{\to}_\rho$ and $\underset{\neg\mathsf{s}}{\to}_\rho$ preserve the shapes of terms. Moreover, the following hold for $\underset{\mathsf{i}}{\to}_\rho \in \{\underset{\neg\mathsf{s}}{\to}_\rho, \underset{\neg\mathsf{w}}{\to}_\rho\}$:*

1. *There is no T such that $T \underset{\mathsf{i}}{\to}_\rho x$, for any variable x.*
2. *$T \underset{\mathsf{i}}{\to}_\rho !U_1$ implies $T = !T_1$ and $T_1 \to_\rho U_1$.*
3. *$T \underset{\mathsf{i}}{\to}_\rho \lambda x.U_1$ implies $T = \lambda x.T_1$ and $T_1 \to_\rho U_1$.*
4. *$T \underset{\mathsf{i}}{\to}_\rho U_1U_2$ implies $T = T_1T_2$, with either (i) $T_1 \underset{\mathsf{i}}{\to}_\rho U_1$ (and $T_2 = U_2$), or (ii) $T_2 \underset{\mathsf{i}}{\to}_\rho U_2$ (and $T_1 = U_1$). Moreover, T_1 and U_1 have the same shape, and so T_2 and U_2.*

Corollary A.1.2. *Let $\mapsto_\rho$ be a rule and $\to_\rho$ be its contextual closure. Assume $T \underset{\neg\mathsf{s}}{\to}_\rho S$ or $T \underset{\neg\mathsf{w}}{\to}_\rho S$.*

- *T is a $\beta_!$-redex if and only if S is.*
- *T is a σ-redex if and only if S is.*

Proof. By repetitively applying Fact A.1.1. □

A.1.2 Surface Factorization, Modularly

In an abstract setting, let us consider a rewrite system $(A, \to)$ where $\to = \to_\xi \cup \to_\gamma$. Under which condition $\to$ admits factorization, assuming that both $\to_\xi$ and $\to_\gamma$ do? To deal with this question, a technique for proving factorization for *compound systems* in a *modular* way has been introduced in [AFG21]. The approach can be seen as an analogous for factorization of the classical technique for confluence based on Hindley-Rosen lemma: if $\to_\xi, \to_\gamma$ are e-factorizing reductions, their union $\to_\xi \cup \to_\gamma$ also is, provided that two *local* conditions of commutation hold.

Theorem A.1.3 ([AFG21] Modular factorization). *Let $\to_\xi = (\underset{\mathsf{e}}{\to}_\xi \cup \underset{\mathsf{i}}{\to}_\xi)$ and $\to_\gamma = (\underset{\mathsf{e}}{\to}_\gamma \cup \underset{\mathsf{i}}{\to}_\gamma)$ be e-factorizing relations. Let $\underset{\mathsf{e}}{\to} := \underset{\mathsf{e}}{\to}_\xi \cup \underset{\mathsf{e}}{\to}_\gamma$, and $\underset{\mathsf{i}}{\to} := \underset{\mathsf{i}}{\to}_\xi \cup \underset{\mathsf{i}}{\to}_\gamma$. The union $\to_\xi \cup \to_\gamma$ satisfies factorization $\mathtt{Fact}(\underset{\mathsf{e}}{\to}, \underset{\mathsf{i}}{\to})$ if the following swaps hold*

$$\underset{\mathsf{i}}{\to}_\xi \cdot \underset{\mathsf{e}}{\to}_\gamma \subseteq \underset{\mathsf{e}}{\to}_\gamma \cdot \to_\xi^* \qquad \textit{and} \qquad \underset{\mathsf{i}}{\to}_\gamma \cdot \underset{\mathsf{e}}{\to}_\xi \subseteq \underset{\mathsf{e}}{\to}_\xi \cdot \to_\gamma^* \qquad (\textbf{Linear Swaps})$$

Extensions of the bang calculus. Following [FG21], we now consider a calculus $(\Lambda^!, \to)$, where $\to = \to_{\beta_!} \cup \to_\gamma$ and $\to_\gamma$ is the contextual closure of a new rule $\mapsto_\gamma$. Theorem A.1.3 states that the compound system $\to_{\beta_!} \cup \to_\gamma$ satisfies surface factorization if $\mathtt{Fact}(\underset{\mathsf{s}}{\to}_{\beta_!}, \underset{\neg\mathsf{s}}{\to}_{\beta_!})$, $\mathtt{Fact}(\underset{\mathsf{s}}{\to}_\gamma, \underset{\neg\mathsf{s}}{\to}_\gamma)$, and the two linear swaps hold. We know that $\mathtt{Fact}(\underset{\mathsf{s}}{\to}_{\beta_!}, \underset{\neg\mathsf{s}}{\to}_{\beta_!})$ always hold. We now show that to verify the linear swaps reduces to a single simple test, leading to Proposition 4.3.5.

First, we observe that each linear swap condition can be tested by considering for the surface step only $\mapsto$, that is, only the closure of $\mapsto$ under *empty* context. This is expressed in the following lemma, where we include also a useful variant.

Lemma A.1.4 (Root linear swaps). *In $\Lambda^!$, let $\to_\xi, \to_\gamma$ be the contextual closure of rules $\mapsto_\xi, \mapsto_\gamma$.*

1. *$\underset{\neg\mathsf{s}}{\to}_\xi \cdot \mapsto_\gamma \subseteq \underset{\mathsf{s}}{\to}_\gamma \cdot \to_\xi^*$ implies $\underset{\neg\mathsf{s}}{\to}_\xi \cdot \underset{\mathsf{s}}{\to}_\gamma \subseteq \underset{\mathsf{s}}{\to}_\gamma \cdot \to_\xi^*$.*
2. *Similarly, $\underset{\neg\mathsf{s}}{\to}_\xi \cdot \mapsto_\gamma \subseteq \underset{\mathsf{s}}{\to}_\gamma \cdot \to_\xi^=$ implies $\underset{\neg\mathsf{s}}{\to}_\xi \cdot \underset{\mathsf{s}}{\to}_\gamma \subseteq \underset{\mathsf{s}}{\to}_\gamma \cdot \to_\xi^=$.*

Proof. Assume $M \underset{\neg\mathsf{s}}{\to}_\xi U \underset{\mathsf{s}}{\to}_\gamma N$. If U is the redex, the claim holds by assumption. Otherwise, we prove $M \underset{\mathsf{s}}{\to}_\gamma \cdot \to_\xi^* N$, by induction on the structure of U. Observe that both M and N have the same shape as U (by Property 4.3.2).

- $U = U_1 U_2$ (hence $M = M_1 M_2$ and $N = N_1 N_2$). We have two cases.
    1. Case $U_1 \underset{\mathsf{s}}{\to}_\gamma N_1$. By Fact A.1.1, either $M_1 \to_\xi U_1$ or $M_2 \to_\xi U_2$.
        (a) Assume $M := M_1 M_2 \underset{\neg\mathsf{s}}{\to}_\xi U_1 M_2 \underset{\mathsf{s}}{\to}_\gamma N_1 M_2 =: N$.
        We have $M_1 \underset{\neg\mathsf{s}}{\to}_\xi U_1 \underset{\mathsf{s}}{\to}_\gamma N_1$, and we conclude by *i.h.*.
        (b) Assume $M := U_1 M_2 \underset{\neg\mathsf{s}}{\to}_\xi U_1 U_2 \underset{\mathsf{s}}{\to}_\gamma N_1 U_2 =: N$.
        Then $U_1 M_2 \underset{\mathsf{s}}{\to}_\gamma N_1 M_2 \to_\xi N_1 U_2$.
    2. Case $U_2 \underset{\mathsf{s}}{\to}_\gamma N_2$. Similar to the above.
- $U = \lambda x.U_0$ (hence $M = \lambda x.M_0$ and $N = \lambda x.N_0$). We conclude by *i.h.*.

Cases $U = !U_0$ or $U = x$ do not apply. □

As we study $\rightarrow_{\beta_!} \cup \rightarrow_{\gamma}$, one of the linear swap is $\underset{\neg\mathsf{s}}{\rightarrow}_{\gamma} \cdot \underset{\mathsf{s}}{\rightarrow}_{\beta_!} \subseteq \underset{\mathsf{s}}{\rightarrow}_{\beta_!} \cdot \rightarrow_{\gamma}^{*}$. We show that *any* $\rightarrow_{\gamma}$ linearly swaps after $\underset{\mathsf{s}}{\rightarrow}_{\beta_!}$ as soon as $\mapsto_{\gamma}$ is *substitutive.*

Lemma A.1.5 (Swap with $\underset{\mathsf{s}}{\rightarrow}_{\beta_!}$)**.** *If $\mapsto_{\gamma}$ is substitutive, then $\underset{\neg\mathsf{s}}{\rightarrow}_{\gamma} \cdot \underset{\mathsf{s}}{\rightarrow}_{\beta_!} \subseteq \underset{\mathsf{s}}{\rightarrow}_{\beta_!} \cdot \rightarrow_{\gamma}^{*}$ always holds.*

Proof. We prove $\underset{\neg\mathsf{s}}{\rightarrow}_{\gamma} \cdot \mapsto_{\beta_!} \subseteq \underset{\mathsf{s}}{\rightarrow}_{\beta_!} \cdot \rightarrow_{\gamma}^{*}$, and conclude by Lemma A.1.4.

Assume $M \underset{\neg\mathsf{s}}{\rightarrow}_{\gamma} (\lambda x.P)!Q \mapsto_{\beta_!} P\{Q/x\}$. We want to prove $M \underset{\mathsf{s}}{\rightarrow}_{\beta_!} \cdot \rightarrow_{\gamma}^{*} P\{Q/x\}$. By Fact A.1.1, $M = M_1 M_2$ and either $M_1 = \lambda x.P_0 \rightarrow_{\gamma} \lambda x.P$ or $M_2 = !Q_0 \rightarrow_{\gamma} !Q$.

- In the first case, $M = (\lambda x.P_0)!Q$, with $P_0 \rightarrow_{\gamma} P$. So, $M = (\lambda x.P_0)!Q \mapsto_{\beta_!} P_0\{Q/x\}$ and we conclude by substitutivity of $\rightarrow_{\gamma}$ (Fact A.0.1.1).
- In the second case, $M = (\lambda x.P)!Q_0$ with $Q_0 \rightarrow_{\gamma} Q$. Therefore $M = (\lambda x.P)!Q_0 \mapsto_{\beta_!} P\{Q_0/x\}$, and we conclude by Fact A.0.1.2. □

Summing up, since surface factorization for $\beta_!$ is known, we obtain the following compact test for surface factorization in extensions of $\rightarrow_{\beta_!}$.

Proposition A.1.6 (A modular test for surface factorization)**.** *Let $\rightarrow_{\beta_!}$ be $\beta_!$-reduction and $\rightarrow_{\gamma}$ be the contextual closure of a rule $\mapsto_{\gamma}$. The reduction $\rightarrow_{\beta_!} \cup \rightarrow_{\gamma}$ satisfies surface factorization if:*

1. Surface factorization of $\rightarrow_{\gamma}$*:* $\quad \rightarrow_{\gamma}^{*} \subseteq \underset{\mathsf{s}}{\rightarrow}_{\gamma}^{*} \cdot \underset{\neg\mathsf{s}}{\rightarrow}_{\gamma}^{*}$
2. $\mapsto_{\gamma}$ *is* substitutive*:* $\quad R \mapsto_{\gamma} R'$ *implies* $R\{Q/x\} \mapsto_{\gamma} R'\{Q/x\}$.
3. Root linear swap*:* $\quad \underset{\neg\mathsf{s}}{\rightarrow}_{\beta_!} \cdot \mapsto_{\gamma} \subseteq \mapsto_{\gamma} \cdot \rightarrow_{\beta_!}^{*}$.

A.1.3 Restriction to Computations

In *Com*, let $\mapsto_{\rho}$ be a rule and $\rightarrow_{\rho}$ be its contextual closure. The restriction of reduction to computations preserves $\rightarrow_{\rho}$, $\underset{\mathsf{s}}{\rightarrow}_{\rho}$, $\underset{\neg\mathsf{s}}{\rightarrow}_{\rho}$, $\underset{\mathsf{w}}{\rightarrow}_{\rho}$, $\underset{\neg\mathsf{w}}{\rightarrow}_{\rho}$ steps. Thus, all properties that hold for $(\Lambda^{!}, \rightarrow_{\rho})$ (e.g. Fact A.1.1 and corollary A.1.2) also hold for $(\mathit{Com}, \rightarrow_{\rho})$.

In particular, Proposition 4.3.5 is immediate consequence of Proposition A.1.6.

APPENDIX B

OPERATIONAL PROPERTIES

B.1 Properties of Reduction in $\lambda_{\copyright}$

We now consider $\lambda_{\copyright}$, that is $(Com, \rightarrow_{\copyright})$. As we have just seen above, the properties we have studied in Appendix A.1 also hold when restricting reduction to computations. Moreover, $\lambda_{\copyright}$ satisfies also specific properties that do not hold in general, as the following.

Lemma B.1.1. *Assume $M \in Com$ and $M \underset{\neg\mathsf{s}}{\rightrightarrows}_{\copyright} L$. M is a* id*-redex (resp. a ι-redex) if and only if L is.*

Proof. If M is a id-redex, this means that $M = (\lambda z.[z])P \underset{\neg\mathsf{s}}{\rightrightarrows}_{\gamma} (\lambda z.[z])N = L$ where $P \underset{\neg\mathsf{s}}{\rightrightarrows}_{\gamma} N$, hence L is a id-redex. Moreover, if M is a ι-redex, then $P \neq [V]$, hence by fact A.1.1 $L \neq [V]'$ for any V'. Thus L is a ι-redex.

Let us prove that if L is a id-redex, so is M. Since $L = (\lambda z.[z])N$, by fact A.1.1, M is an application; we have the following cases:
(i.) Either $M = (\lambda z.P)N \underset{\neg\mathsf{s}}{\rightrightarrows}_{\gamma} (\lambda z.[z])N$ where $P \underset{\neg\mathsf{s}}{\rightrightarrows}_{\gamma} [z]$.
(ii.) or $M = (\lambda z.[z])P \underset{\neg\mathsf{s}}{\rightrightarrows}_{\gamma} (\lambda z.[z])N$ where $P \underset{\neg\mathsf{s}}{\rightrightarrows}_{\gamma} N$.
The case *(i.)* is impossible because otherwise $P = [V]$ for some value V, by Fact A.1.1, such that $V \rightarrow_{\gamma} z$, but such a V does not exist. Therefore we are necessarily in case *(ii.)*, *i.e.* M is a id-redex. Moreover, if L is a ι-redex, then $N \not\equiv [V]$, hence N is an application, and so is P by Fact A.1.1. □

Lemma B.1.2. *There is no $P \in Com$ such that $P \rightarrow_{\iota} [x]$*

Proofs of the lemma for the postponement of ι. Notation: $\Rightarrow_{①} ::= \rightarrow^{*}_{\beta_1} \rightarrow^{=}_{\iota}$

Lemma B.1.3 (ι vs. β_1)**.**

$$M \rightarrow_{\iota} L \rightarrow_{\beta_1} N \textit{ implies } M \rightarrow^{*}_{\beta_1} \cdot \rightarrow^{=}_{\iota} N$$

Proof. By induction on L. Note that if $L = [x]$ there is no β_1 reduction from it, so this case is not in the scope of the induction. Cases:

- $L = (\lambda x.[x])[V] \mapsto_{\beta_1} [V] = N$. Then, there are two possibilities.
  Either $M = (\lambda z.[z])L \mapsto_\iota L$ then
  $$M = (\lambda z.[z])((\lambda x.[x])[V]) \to_{\beta_1} (\lambda z.[z])[V] \to_{\beta_1} [V] = N.$$
  Or $M = (\lambda x.[x])[W]$ with $[W] \to_\iota [V]$, and then
  $$M = (\lambda x.[x])[W] \to_{\beta_1} [W] \to_\iota [V] = N$$
- $L = [\lambda x.P] \to_{\beta_1} [\lambda x.P'] = N$ where $P \to_{\beta_1} P'$. In this case note that M has necessary the shape $[\lambda x.Q]$ where $Q \to_\iota P$. Otherwise, M should have been $(\lambda z.[z])L \mapsto_\iota L$ but it is impossible by definition of $\mapsto_\iota$ since $L = [\lambda x.P]$. The thesis follows by induction, since we have $Q \to_\iota P \to_{\beta_1} P'$, $Q \Rightarrow_{①} P$.
- $L = VP \to_{\beta_1} V'P' = N$ where $\to_{\beta_1}$ is not root steps, that is:

  a either $V \to_{\beta_1} V'$ and $P = P'$;

  b or $V = V'$ and $P \to_{\beta_1} P'$.

  By Fact 4.3.2, M, L, N are applications. So, M has the following shape:

  1. $M = VQ$ with $Q \to_\iota P$
  2. $M = WP$ with $W \to_\iota V$
  3. $M = (\lambda x.[x])(VP) \mapsto_\iota VP = L$

  We distinguish six sub-cases:

  Case a1 We have $M = VQ \to_{\beta_1} V'Q \to_\iota V'P = N$, switching the steps $\to_\iota$ and $\to_{\beta_1}$, directly.

  Case b1 $Q \to_\iota P \to_{\beta_1} P'$, then the thesis follows by i.h., that is: $Q \Rightarrow_{①} P'$ and then $M = VQ \Rightarrow_{①} VP' = N$.

  Case a2 $W \to_\iota V \to_{\beta_1} V'$, then the thesis follows by i.h., that is: $W \Rightarrow_{①} V'$ and then $M = WP \Rightarrow_{①} V'P = N$.

  Case b2 We have $M = WP \to_{\beta_1} WP' \to_\iota VP' = N$, switching the steps $\to_\iota$ and $\to_{\beta_1}$, directly.

  Case a3 $M = (\lambda x.[x])(VP) \to_{\beta_1} (\lambda x.[x])(V'P) \mapsto_\iota V'P = N$

  Case b3 $M = (\lambda x.[x])(VP) \to_{\beta_1} (\lambda x.[x])(VP') \mapsto_\iota VP' = N$

□

Notation $\Rightarrow_{②} ::= \to^*_{\beta_1} \to^=_{\beta_2} \to^*_{\beta_1} \to^*_\iota$

Lemma B.1.4 (ι vs. β_2)**.**

$$M \to_\iota L \to_{\beta_2} N \textit{ implies } M \to^*_{\beta_1} \to^=_{\beta_2} \to^*_{\beta_1} \to^*_\iota N$$

Proof. By induction on L. Note that if $L = [x]$ there is no β_2 reduction from it, so this case is not in the scope of the induction. Cases:

- $L = (\lambda x.P')[V'] \mapsto_{\beta_2} P'\{V'/x\} = N$. Then, there are three possibilities.

i $M = (\lambda x.P)[V']$ with $P \to_\iota P'$

ii $M = (\lambda x.P')[V]$ with $V \to_\iota V'$

iii $M = (\lambda x.[x])L \mapsto_\iota L$

So by analizyng each of the three cases above, we can postpone the $\to_\iota$ step as follows:

Case i $M = (\lambda x.P)[V] \mapsto_{\beta_2} P\{V/x\} \to_\iota P'\{V/x\}$ where the last reduction step is possible by Fact A.0.1.1. Note that $P \neq [x]$ otherwise would not possible $P \to_\iota P'$, as assumed.

Case iii $M = (\lambda x.P)[V] \mapsto_{\beta_2} P\{V/x\} \to_\iota^* P\{V'/x\}$ where the last reduction step is possible by Fact A.0.1.2.

Case iii $M = (\lambda x.[x])L \mapsto_{\beta_2} (\lambda x.[x])N \mapsto_\iota N$

- $L = [\lambda x.P] \to_{\beta_2} [\lambda x.P'] = N$ where $P \to_{\beta_2} P'$. In this case note that M has necessary the shape $[\lambda x.Q]$ where $Q \to_\iota P$. Otherwise, M should have been $(\lambda z.[z])L \mapsto_\iota L$ but it is impossible by definition of $\mapsto_\iota$ since $L = [\lambda x.P]$. The thesis follows by induction, since we have $Q \to_\iota P \to_{\beta_2} P'$, $Q \Rightarrow_{②} P$.

- $L = VP \to_{\beta_2} V'P' = N$ where $\to_{\beta_2}$ is not root steps, that is:

  a either $V \to_{\beta_2} V'$ and $P = P'$;

  b or $V = V'$ and $P \to_{\beta_2} P'$.

  By Fact 4.3.2, M, L, N are applications. So, M has the following shape:

  1. $M = VQ$ with $Q \to_\iota P$
  2. $M = WP$ with $W \to_\iota V$
  3. $M = (\lambda x.[x])(VP) \mapsto_\iota VP = L$

  We distinguish six sub-cases:

  Case a1 We have $M = VQ \to_{\beta_2} V'Q \to_\iota V'P = N$, switching the steps $\to_\iota$ and $\to_{\beta_2}$, directly.

  Case b1 $Q \to_\iota P \to_{\beta_2} P'$, then the thesis follows by i.h., that is: $Q \Rightarrow_{②} P'$ and then $M = VQ \Rightarrow_{②} VP' = N$.

  Case a2 $W \to_\iota V \to_{\beta_2} V'$, then the thesis follows by i.h., that is: $W \Rightarrow_{②} V'$ and then $M = WP \Rightarrow_{②} V'P = N$.

  Case b2 We have $M = WP \to_{\beta_2} WP' \to_\iota VP' = N$, switching the steps $\to_\iota$ and $\to_{\beta_2}$, directly.

  Case a3 $M = (\lambda x.[x])(VP) \to_{\beta_2} (\lambda x.[x])(V'P) \mapsto_\iota V'P = N$

  Case b3 $M = (\lambda x.[x])(VP) \to_{\beta_2} (\lambda x.[x])(VP') \mapsto_\iota VP' = N$

□

Notation: $\Rightarrow_{①} ::= (\to_\sigma \cup \to_{\beta_1})^* \to_\iota^=$

Lemma B.1.5 (ι vs. σ).

$$M \to_\iota L \to_\sigma N \textit{ implies } M \to_\sigma^* \cdot \to_\iota^= N$$

Proof. By induction on L. Distinguishing if the last $\to_\sigma$ is a root step or not.

If $L \mapsto_\sigma N$ then $L = V((\lambda x.P)Q)$ and $N = (\lambda x.VP)Q$. In this case, M can be of 4 different shapes:

i $M = (\lambda z.[z])(V((\lambda x.P)Q))$

ii $M = V((\lambda z.[z])((\lambda x.P)Q))$

iii $M = V((\lambda x.(\lambda z.[z])P)Q)$

iv $M = (V((\lambda x.P)((\lambda z.[z])Q)))$

v $M = W((\lambda x.P)Q)$ with $W \to_\iota V$ and $\to_\iota$ is not a root step

vi $M = V((\lambda x.R)Q)$ with $R \to_\iota P$ and $\to_\iota$ is not a root step

vii $M = V((\lambda x.P)R)$ with $R \to_\iota Q$ and $\to_\iota$ is not a root step

So by analizyng each of the four cases above, we can postpone the $\to_\iota$ step as follows:

Case i $M = (\lambda z.[z])(V((\lambda x.P)Q)) \to_\sigma (\lambda z.[z])((\lambda x.VP)Q) \mapsto_\iota (\lambda x.VP)Q = N$

Case ii $M = V((\lambda z.[z])((\lambda x.P)Q)) \to_\sigma V((\lambda x.(\lambda z.[z])P)Q) \to_\sigma (\lambda x.V((\lambda z.[z])P))Q \to_\gamma (\lambda x.VP)Q = N$ where in the last step γ is ι or β_1 depending on whether P is of the form $[W]$ or not.

Case iii $M = V((\lambda x.(\lambda z.[z])P)Q) \to_\sigma (\lambda x.V((\lambda z.[z])P))Q \to_\gamma (\lambda x.VP)Q = N$ where in the last step γ is ι or β_1 depending on whether P is of the form $[W]$ or not.

Case iv $M = V((\lambda x.P)((\lambda z.[z])Q)) \underset{\mathsf{s}}{\to}_\sigma (\lambda x.VP)((\lambda z.[z])Q) \to_\gamma (\lambda x.VP)Q = N$ where in the last step γ is ι or β_1 depending on whether Q is of the form $[W]$ or not.

Case v $M = W((\lambda x.P)Q) \mapsto_\sigma (\lambda x.WP)Q \to_\iota (\lambda x.VP)Q = N$

Case vii $M = V((\lambda x.R)Q) \mapsto_\sigma (\lambda x.VR)Q \to_\iota (\lambda x.VP)Q = N$

Case vii $M = V((\lambda x.P)R) \mapsto_\sigma (\lambda x.VP)R \to_\iota (\lambda x.VP)Q = N$

Consider the case where $L = [\lambda x.P] \to_\sigma [\lambda x.P'] = N$ where $P \to_\sigma P'$. In this case note that M has necessary the shape $[\lambda x.Q]$ where $Q \to_\iota P$. Otherwise, M should have been $(\lambda z.[z])L \mapsto_\iota L$ but it is impossible by definition of $\mapsto_\iota$ since $L = [\lambda x.P]$. The thesis follows by induction, since we have $Q \to_\iota P \to_\sigma P'$, $Q \Rightarrow_{③} P$.

The last case to consider is $L = VP \to_\sigma V'P' = N$ where $\to_\sigma$ is not root steps, that is:

a either $V \to_\sigma V'$ and $P = P'$;

b or $V = V'$ and $P \to_\sigma P'$.

By Fact 4.3.2, M, L, N are applications. So, M has one of the following shapes:

1. $M = (\lambda z.[z])L \mapsto_\iota VP = L$
2. $M = WP$ with $W \to_\iota V$
3. $M = VQ$ with $Q \to_\iota P$

Hence, we distinguish six sub-cases:

Case a1 $M = (\lambda x.[x])(VP) \to_\sigma (\lambda x.[x])(V'P) \mapsto_\iota V'P = N$

Case b1 $M = (\lambda x.[x])(VP) \to_\sigma (\lambda x.[x])(VP') \mapsto_\iota VP' = N$

Case a2 $W \to_\iota V \to_\sigma V'$, then the thesis follows by i.h., that is: $W \Rightarrow_{③} V'$ and then $M = WP \Rightarrow_{③} V'P = N$.

Case b2 We have $M = WP \to_\sigma WP' \to_\iota VP' = N$, switching the steps $\to_\iota$ and $\to_\sigma$, directly.

Case a3 We have $M = VQ \to_\sigma V'Q \to_\iota V'P = N$, switching the steps $\to_\iota$ and $\to_\sigma$, directly.

Case b3 $Q \to_\iota P \to_\sigma P'$, then the thesis follows by i.h., that is: $Q \Rightarrow_{③} P'$ and then $M = VQ \Rightarrow_{③} VP' = N$.

□

B.2 Normalization of $\lambda_{©}$

Lemma 5.0.3. *Let $\to = \to_\sigma \cup \to_{\beta_c}$ and $\mathsf{e} \in \{\mathsf{w}, \mathsf{s}\}$. Assume $M \underset{\neg\mathsf{e}}{\to} N$. Then, M is* e*-normal if and only if N is* e*-normal.*

Proof. By easy induction on the shape of M. Observe that M and N have the same shape, because the step $M \underset{\neg\mathsf{e}}{\to} N$ is not a root step.

- $M = [V]$, and $N = [V']$: the claim is trivial.
- $M = VP$ and $N = V'P'$. Either $V \underset{\neg\mathsf{e}}{\to} V'$ (and $P' = P$) or $P \underset{\neg\mathsf{e}}{\to} P'$ (and $V = V'$). Assume $M = VP$ is e-normal. Since V and P are e-normal, by *i.h.* so are V' and P'. Moreover, N is not a redex, by Corollary A.1.2, so N is normal. Assuming $N = V'P'$ normal is similar.

□

Fact B.2.1. *The reduction ι is quasi-diamond. Therefore, if $S \to_\iota^k N$ where N is ι-normal, then any maximal ι-sequence from S ends in N, in k steps.*

Lemma 5.0.1. *Assume $M \to_\iota N$.*

1. *M is β_c-normal if and only if N is β_c-normal.*
2. *If M is σ-normal, so is N.*

Proof. Easy to prove by induction on the structure of terms. □

Fact B.2.2 (Shape preservation of ι-sequences)**.** *If S is not an ι-redex, and $S \to_\iota^k N$ then no term in the sequence is an ι-redex, and so N has the same shape as S:*

1. *$S = [(\lambda x.Q)]$ implies $N = [(\lambda x.N_Q)]$. Moreover, $Q \to_\iota^k N_Q$.*
2. *$S = xP$ implies $N = xN_P$. Moreover, $P \to_\iota^k N_P$.*
3. *$S = (\lambda x.Q)P$ implies $N = (\lambda x.N_Q)N_P$. Moreover, $Q \to_\iota^{k_1} N_Q$, $P \to_\iota^{k_2} N_P$ and $k = k_1 + k_2$.*

Lemma 5.1.2. *Assume $M \to_\iota^k N$, where $k > 0$, and N is $\sigma\iota$-normal. If M is not σ-normal, then there exist M' and N' such that either $M \to_\sigma M' \to_\iota N' \to_\iota^{k-1} N$ or $M \to_\sigma M' \to_{\beta_c} N' \to_\iota^{k-1} N$.*

Proof. The proof is by induction on M.

Assume M is a is a σ-redex, *i.e.* $M = V((\lambda x.P)L)$ where V is an abstraction.

- If M is also an ι-redex, then $V = \mathbf{I}$, and:

  1. If $P = [U]$, then $M = \mathbf{I}((\lambda x.[U])L) \to_\iota (\lambda x.[U])L \to_\iota^{k-1} N$ and $M \to_\sigma (\lambda x.\mathbf{I}[U])L \to_\beta (\lambda x.[U])L$.
  2. If $P \neq [U]$, then $M = \mathbf{I}((\lambda x.P)L) \to_\iota (\lambda x.P)L \to_\iota^{k-1} N$ and $M \to_\sigma (\lambda x.\mathbf{I}P)L \to_\iota (\lambda x.P)L$.

- Otherwise, if $M = V(\mathbf{I}L)$, then $M \to_\iota VL \to_\iota^{k-1} N$ and $M \to_\sigma (\lambda z.V[z])L \to_\beta VL$.

- No other case is possible, because $M = V((\lambda x.P)L)$ not an ι-redex implies (by Fact B.2.2) that $N = N_1N_2$, with $N_1 \in \mathtt{Abs}$. If $(\lambda x.P) \neq \mathbf{I}$, then N would be a σ-redex, because again $N_2 = N_2'N_2''$ with $N_2' \in \mathtt{Abs}$ (by Fact B.2.2).

Assume M is not a σ-redex. We examine the shape of S and use Fact B.2.2.

- $M = [\lambda x.Q]$. We have $N = [\lambda x.N_Q]$, $Q \to_\iota^k N_Q$, where Q is not σ-normal, and N_Q is $\sigma\iota$-normal. We conclude by *i.h.*.

- $M = \mathbf{I}P$. We have $\mathbf{I}P \to_\iota P \to_\iota^{k-1} N$. Since P is not σ-normal, we use the *i.h.* on $P \to_\iota^{k-1} N$, obtaining that $P \to_\iota N' \to_\iota^{k-2} N$ and $P \to_\sigma P' \to_{\beta_c\iota} N'$. Therefore also $\mathbf{I}P \to_\sigma \mathbf{I}P' \to_{\beta_c\iota} \mathbf{I}N' \underset{\mathsf{i}}{\to} N' \underset{\mathsf{i}}{\to}^{k-1} N$.

- $M = (\lambda x.Q)P$ (M is not an ι-redex). We have $N = (\lambda x.N_Q)N_P$, where N_Q and N_P are $\sigma\iota$-normal. We distinguish two cases.

  – If Q is not σ-normal, we note that $Q \to_\iota^{k_1} N_Q$, and conclude by *i.h.* Indeed, by *i.h.*, we obtain that $Q \to_\iota N_Q' \to_\iota^{k_1-1} N_Q$, and $Q \to_\sigma Q' \to_{\beta_c\iota} N_Q'$. So, $(\lambda x.Q)P \to_\iota (\lambda x.N_Q')P \to_\iota^{k_1-1} (\lambda x.N_Q)P \to_\iota^{k_2} (\lambda x.N_Q)N_P$, and $(\lambda x.Q)P \to_\sigma (\lambda x.Q')P \to_{\beta_c\iota} (\lambda x.N_Q')P$.

  – If P is not σ-normal, we note that $P \to_\iota^{k_2} N_P$, and conclude by *i.h.* □

Lemma 5.1.3 (Termination of $\sigma\mathsf{id}$). *$\to_{\mathsf{id}} \cup \to_\sigma$ is strongly normalizing.*

Proof. We define two sizes $\mathsf{s}(M)$ and $\mathsf{s}_\sigma(M)$ for any term M.

$$\begin{aligned} \mathsf{s}(x) &= 1 & \mathsf{s}_\sigma(x) &= 1 \\ \mathsf{s}(\lambda x.M) &= \mathsf{s}(M) + 1 & \mathsf{s}_\sigma(\lambda x.M) &= \mathsf{s}_\sigma(M) + \mathsf{s}(M) \\ \mathsf{s}(VM) &= \mathsf{s}(V) + \mathsf{s}(M) & \mathsf{s}_\sigma(VM) &= \mathsf{s}_\sigma(V) + \mathsf{s}_\sigma(M) + 2\mathsf{s}(V)\mathsf{s}(M) \\ \mathsf{s}([M]) &= \mathsf{s}(M) & \mathsf{s}_\sigma([M]) &= \mathsf{s}_\sigma(M) \end{aligned}$$

Note that $\mathsf{s}(M) > 0$ and $\mathsf{s}_\sigma(M) > 0$ for any term M. It easy to check that if $M \to_{\mathsf{id}} \cup \to_\sigma N$, then $(\mathsf{s}(N), \mathsf{s}_\sigma(N)) <_{\text{lex}} (\mathsf{s}(M), \mathsf{s}_\sigma(M))$, where $<_{\text{lex}}$ is the strict lexicographical order on $\mathbb{N}^2$. Indeed, if $M \to_{\mathsf{id}} N$, then $\mathsf{s}(M) > \mathsf{s}(N)$; and if $M \to_\sigma N$ then $\mathsf{s}(M) = \mathsf{s}(N)$ and $\mathsf{s}_\sigma(M) > \mathsf{s}_\sigma(N)$. The proof is by straightforward induction on M. We show only the root-cases, the other cases follow from the *i.h.* immediately.

- If $(\lambda x.[x])M \mapsto_{\mathsf{id}} M$ then $\mathsf{s}((\lambda x.[x])M) = \mathsf{s}(\lambda x.[x]) + \mathsf{s}(M) > \mathsf{s}(M)$.
- If $(\lambda x.M)((\lambda y.N)L) \mapsto_{\sigma} (\lambda y.(\lambda x.M)N)L$ then clearly $\mathsf{s}((\lambda x.M)((\lambda y.N)L)) = \mathsf{s}((\lambda y.(\lambda x.M)N)L)$ and

$$
\begin{aligned}
&\mathsf{s}_{\sigma}((\lambda x.M)((\lambda y.N)L))\\
&= \mathsf{s}_{\sigma}(\lambda x.M) + \mathsf{s}_{\sigma}((\lambda y.N)L) + 2\mathsf{s}(\lambda x.M)\mathsf{s}((\lambda y.N)L)\\
&= \mathsf{s}_{\sigma}(\lambda x.M) + \mathsf{s}_{\sigma}(\lambda y.N) + \mathsf{s}_{\sigma}(L) + 2\mathsf{s}(\lambda y.N)\mathsf{s}(L) + 2\mathsf{s}(\lambda x.M)\mathsf{s}((\lambda y.N)L)\\
&= \mathsf{s}_{\sigma}(\lambda x.M) + \mathsf{s}_{\sigma}(N) + \mathsf{s}(N) + \mathsf{s}_{\sigma}(L) + 2\mathsf{s}(N)\mathsf{s}(L) + 2\mathsf{s}(L) + 2\mathsf{s}(\lambda x.M)\mathsf{s}(\lambda y.N) + 2\mathsf{s}(\lambda x.M)\mathsf{s}(L)\\
&= \mathsf{s}_{\sigma}(\lambda x.M) + \mathsf{s}_{\sigma}(N) + \mathsf{s}(N) + \mathsf{s}_{\sigma}(L) + 2\mathsf{s}(N)\mathsf{s}(L) + 2\mathsf{s}(L)\acute{\mathsf{t}} + \underline{2\mathsf{s}(\lambda x.M)} + 2\mathsf{s}(\lambda x.M)\mathsf{s}(N) + 2\mathsf{s}(\lambda x.M)\mathsf{s}(L)\\
&> \mathsf{s}_{\sigma}(\lambda x.M) + \mathsf{s}_{\sigma}(N) + 2\mathsf{s}(\lambda x.M)\mathsf{s}(N) + \underline{\mathsf{s}(\lambda x.M)} + \mathsf{s}(N) + \mathsf{s}_{\sigma}(L) + 2\mathsf{s}(\lambda x.M)\mathsf{s}(L) + 2\mathsf{s}(N)\mathsf{s}(L) + 2\mathsf{s}(L)\\
&= \mathsf{s}_{\sigma}(\lambda x.M) + \mathsf{s}_{\sigma}(N) + 2\mathsf{s}(\lambda x.M)\mathsf{s}(N) + \mathsf{s}(\lambda x.M) + \mathsf{s}(N) + \mathsf{s}_{\sigma}(L) + 2\mathsf{s}((\lambda x.M)N)\mathsf{s}(L) + 2\mathsf{s}(L)\\
&= \mathsf{s}_{\sigma}((\lambda x.M)N) + \mathsf{s}((\lambda x.M)N) + \mathsf{s}_{\sigma}(L) + 2\mathsf{s}(\lambda y.(\lambda x.M)N)\mathsf{s}(L)\\
&= \mathsf{s}_{\sigma}(\lambda y.(\lambda x.M)N) + \mathsf{s}_{\sigma}(L) + 2\mathsf{s}(\lambda y.(\lambda x.M)N)\mathsf{s}(L)\\
&= \mathsf{s}_{\sigma}((\lambda y.(\lambda x.M)N)L) \qquad \square
\end{aligned}
$$

APPENDIX C

RELATING CALCULI

C.1 Equational Correspondence between $\lambda_{\copyright}$ and λ_{ml^*}

Proposition 3.1.1. *The following hold:*

1. $M =_{\copyright\eta} (M^\circ)^\bullet$ *for every term* M *in* $\lambda_{\copyright}$;
2. $(P^\bullet)^\circ =_{ml^*} P$ *for every term* P *in* λ_{ml^*};
3. $M =_{\copyright\eta} N$ *implies* $M^\circ =_{ml^*} N^\circ$;
4. $P =_{ml^*} Q$ *implies* $P^\bullet =_{\copyright\eta} Q^\bullet$.

Proof. (1.) By induction over terms in $\lambda_{\copyright\eta}$

$x^{\circ\bullet} \equiv x$

$(\lambda x.M)^{\circ\bullet} = \lambda x.M^{\circ\bullet} \overset{i.h.}{=}_{\copyright\eta} \lambda x.M$

$([V])^{\circ\bullet} = [V^\circ]^\bullet = [V^{\circ\bullet}] \overset{i.h.}{=}_{\copyright\eta} [V]$

$((\lambda x.M)[V])^{\circ\bullet} = ((\lambda x.M)^\circ V^\circ)^\bullet = ((\lambda x.M)^{\circ\bullet}\ [V^{\circ\bullet}]) \overset{i.h.}{=}_{\copyright\eta} (\lambda x.M)[V]$

$(xM)^{\circ\bullet} = (\mathsf{let}\, y \coloneqq M^\circ \,\mathsf{in}\, xy)^\bullet = (\lambda y.(x[y]))M^{\circ\bullet} \rightarrow_\eta xM^{\circ\bullet} \overset{i.h.}{=}_{\copyright\eta} xM$

$((\lambda x.N)M)^{\circ\bullet} = (\mathsf{let}\, x \coloneqq M^\circ \,\mathsf{in}\, N^\circ)^\bullet = (\lambda x.N^{\circ\bullet})M^{\circ\bullet} \overset{i.h.}{=}_{\copyright\eta} (\lambda x.N)M$

(2.) By induction over terms in λ_{ml^*}.

$x^{\bullet\circ} \equiv x$ by definition

$(\lambda x.M)^{\bullet\circ} = (\lambda x.M^\bullet)^\circ = (\lambda x.M^{\bullet\circ}) \overset{i.h.}{=}_{ml^*} \lambda x.M$

$[V]^{\bullet\circ} = ([V^\bullet])^\circ = [V^{\bullet\circ}] \overset{i.h.}{=}_{ml^*} [V]$

$(VW)^{\bullet\circ} = (V^\bullet[W^\bullet])^\circ = V^{\bullet\circ}W^{\bullet\circ} \overset{i.h.}{=}_{ml^*} VW$

$(\mathsf{let}\, x := M \,\mathsf{in}\, N)^{\bullet\circ} = ((\lambda x.N^\bullet)M^\bullet)^\circ$.
We distinguish two subcases: a. $M = [V]$, b. $M \neq [V]$ for any V.

a. $((\lambda x.N^\bullet)[V^\bullet])^\circ = (\lambda x.N^{\bullet\circ})V^{\bullet\circ} \mapsto_{c.\beta} N^{\bullet\circ}\{V^{\bullet\circ}/x\}$. On the other side, $(\mathsf{let}\, x := [V] \,\mathsf{in}\, N)^{\bullet\circ} \mapsto_{c.\mathsf{let}.\beta} (N\{V/x\})^{\bullet\circ}$. The thesis follows by substitutivity of both the translations.

b.$((\lambda x.N^\bullet)M^\bullet)^\circ = (\mathsf{let}\, x := M^{\bullet\circ} \,\mathsf{in}\, N^{\bullet\circ}) \overset{i.h.}{=}_{ml^*} \mathsf{let}\, x := M \,\mathsf{in}\, N$

(3.) By cases over reductions in $\lambda_{©\eta}$

(β_c) $(\lambda x.N)[V] \to N\{V/x\}$
$((\lambda x.N)[V])^\circ = \mathsf{let}\, x := [V^\circ] \,\mathsf{in}\, N^\circ \to_{c.\mathsf{let}.\beta} N^\circ\{V^\circ/x\} = (N\{V/x\})^\circ$ by substitutivity of $^\circ$

(id) $(\lambda x.[x])M \to M$
$(\lambda x.[x])M)^\circ = \mathsf{let}\, x := M^\circ \,\mathsf{in}\, [x] \to_{c.\mathsf{let}.\eta} M^\circ$

(η) trivial.

(σ) $(\lambda y.N)((\lambda x.M)L) \to (\lambda x.(\lambda y.N)M)L$
We distinguish 4 cases:

1. $L = [V]$ and $M = [W]$
$(\lambda y.N)((\lambda x.[W])[V])^\circ = \mathsf{let}\, y := (\lambda x.[W^\circ])V^\circ \,\mathsf{in}\, N^\circ \to_\beta \mathsf{let}\, y := [W^\circ\{V^\circ/x\}] \,\mathsf{in}\, N^\circ \to_{c.\mathsf{let}.\beta} N^\circ\{W^\circ\{V^\circ/x\}/y\}$
$((\lambda x.(\lambda y.N)[W])[V])^\circ = (\lambda x.((\lambda y.N^\circ)W^\circ))V^\circ \twoheadrightarrow_\beta N^\circ\{W^\circ\{V^\circ/x\}/y\}$
2. $L = [V]$ and $M \neq [W]$
$(\lambda y.N)((\lambda x.M)[V])^\circ = \mathsf{let}\, y := (\lambda x.M^\circ)V^\circ \,\mathsf{in}\, N^\circ \to_\beta \mathsf{let}\, y := M^\circ\{V^\circ/x\} \,\mathsf{in}\, N^\circ$
$((\lambda x.(\lambda y.N)M)[V])^\circ = \lambda x.(\mathsf{let}\, y := M^\circ \,\mathsf{in}\, N^\circ)V^\circ \to_\beta \mathsf{let}\, y := M^\circ\{V^\circ/x\} \,\mathsf{in}\, N^\circ$
3. $L \neq [V]$ and $M = [W]$
$(\lambda y.N)((\lambda x.[W])L)^\circ = \mathsf{let}\, y := (\mathsf{let}\, x := L^\circ \,\mathsf{in}\, [W^\circ]) \,\mathsf{in}\, N^\circ \to_{c.\mathsf{let}.ass} \mathsf{let}\, x := L^\circ \,\mathsf{in}\, (\mathsf{let}\, y := [W^\circ] \,\mathsf{in}\, N^\circ) \to_{c.\mathsf{let}.\beta} \mathsf{let}\, x := L^\circ \,\mathsf{in}\, N^\circ\{W^\circ/y\}$
$((\lambda x.(\lambda y.N)[W])L)^\circ = \mathsf{let}\, x := L^\circ \,\mathsf{in}\, (\lambda y.N^\circ)[W^\circ] \to_\beta \mathsf{let}\, x := L^\circ \,\mathsf{in}\, (N^\circ)\{W^\circ/y\}$
4. $L, M \neq [V]$
$(\lambda y.N)((\lambda x.M)L)^\circ = \mathsf{let}\, y := (\mathsf{let}\, x := L^\circ \,\mathsf{in}\, M^\circ) \,\mathsf{in}\, N^\circ \to_{c.\mathsf{let}.ass} \mathsf{let}\, x := L^\circ \,\mathsf{in}\, (\mathsf{let}\, y := M^\circ \,\mathsf{in}\, N^\circ) = ((\lambda x.(\lambda y.N)M)L)^\circ$

(4.) By cases over reductions in λ_{ml^*}.

$(c.\beta)$ $(\lambda x.M)V \to M\{V/x\}$
$((\lambda x.M)V)^\bullet = (\lambda x.M^\bullet)[V^\bullet] \to_{\beta_c} M^\bullet\{V^\bullet/x\} = (M\{V/x\})^\bullet$ by substitutivity of $^\bullet$.

$(c.\eta)$ $\lambda x.Vx \to V \qquad x \notin \mathsf{fv}(V)$
$(\lambda x.Vx)^\bullet \equiv \lambda x.V^\bullet[x] \to_\eta V^\bullet$

$(c.\mathbf{let}.\beta)$ $\mathsf{let}\, x := [V] \,\mathsf{in}\, N \to N\{V/x\}$
$(\mathsf{let}\, x := [V] \,\mathsf{in}\, N)^\bullet = (\lambda x.N^\bullet)[V^\bullet] \to_{\beta_c} N^\bullet\{V^\bullet/x\} = (M\{V/x\})^\bullet$ by substitutivity of $^\bullet$.

$(c.\mathbf{let}.\eta)$ $\mathsf{let}\, x := M \,\mathsf{in}\, [x] \to M \qquad x \notin \mathsf{fv}(M)$
$(\mathsf{let}\, x := M \,\mathsf{in}\, [x])^\bullet = (\lambda x.[x])M^\bullet \to_{\mathsf{id}} M^\bullet$

$(c.\textbf{let}.\boldsymbol{ass})$ $\mathsf{let}\, y := (\mathsf{let}\, x := L \,\mathsf{in}\, M) \,\mathsf{in}\, N \to \mathsf{let}\, x := L \,\mathsf{in}\, (\mathsf{let}\, y := M \,\mathsf{in}\, N)$
$(\mathsf{let}\, y := (\mathsf{let}\, x := L \,\mathsf{in}\, M) \,\mathsf{in}\, N)^\bullet \;=\; (\lambda y.N^\bullet)((\lambda x.M^\bullet)L^\bullet) \to_\sigma$
$(\lambda x.(\lambda y.N^\bullet)M^\bullet)L^\bullet \;=\; \mathsf{let}\, x := L \,\mathsf{in}\, (\mathsf{let}\, y := M \,\mathsf{in}\, N)$

□

The following is a more informative version of Proposition 3.1.1.

Proposition C.1.1 (Correspondence between $\lambda_{©}$ and λ_{ml^*})**.** *The following hold:*

1. *$M \to^*_{©\eta} (M^\circ)^\bullet$ for every term M in $\lambda_{©}$;*
2. *$(P^\bullet)^\circ =_{\beta,\mathsf{let}.\beta} P$ for every term P in λ_{ml^*};*
3. *if $M \to_{©} N$ then $M^\circ =_{ml^*\smallsetminus\eta} N^\circ$; if $M \to_{©\eta} N$ then $M^\circ =_{ml^*} N^\circ$;*
4. *if $P \to_{ml^*\smallsetminus\eta} Q$ then $P^\bullet \to_{©} Q^\bullet$; if $P \to_{©\eta} Q$ then $P^\bullet \to_{©\eta} Q^\bullet$.*

Proof.

1. By induction over terms in $\lambda_{©}$.
   - $x^{\circ\bullet} = x^\bullet = x$.
   - $(\lambda x.M)^{\circ\bullet} = (\lambda x.M^\circ)^\bullet = \lambda x.(M^{\circ\bullet}) \overset{i.h.}{\to^*_\eta} \lambda x.M$.
   - $([V])^{\circ\bullet} = [V^\circ]^\bullet = [V^{\circ\bullet}] \overset{i.h.}{\to^*_\eta} !V$.
   - $(W\,[V])^{\circ\bullet} = (W^\circ V^\circ)^\bullet = W^{\circ\bullet}\,[V^{\circ\bullet}] \overset{i.h.}{\to^*_\eta} W[V]$.
   - For $M \neq [V]$ for any value V, $(xM)^{\circ\bullet} = (\mathsf{let}\, y := M^\circ \,\mathsf{in}\, xy)^\bullet = (\lambda y.(x\,[y]))M^{\circ\bullet} \to_\eta xM^{\circ\bullet} \overset{i.h.}{\to^*_\eta} xM$.
   - For $M \neq [V]$ for any value V, $((\lambda x.N)M)^{\circ\bullet} = (\mathsf{let}\, x := M^\circ \,\mathsf{in}\, N^\circ)^\bullet = (\lambda x.N^{\circ\bullet})M^{\circ\bullet} \overset{i.h.}{\to^*_\eta} (\lambda x.N)M$.
2. By induction over terms in λ_{ml^*}.
   - $x^{\bullet\circ} = x^\circ = x$.
   - $(\lambda x.P)^{\bullet\circ} = (\lambda x.P^\bullet)^\circ = \lambda x.(P^{\bullet\circ}) \overset{i.h.}{=}_{\beta,\mathsf{let}.\beta} \lambda x.P$.
   - $[V]^{\bullet\circ} = ([V^\bullet])^\circ = [V^{\bullet\circ}] \overset{i.h.}{=}_{\beta,\mathsf{let}.\beta} [V]$.
   - $(VW)^{\bullet\circ} = (V^\bullet\,[W^\bullet])^\circ = V^{\bullet\circ}\,W^{\bullet\circ} \overset{i.h.}{=}_{\beta,\mathsf{let}.\beta} VW$.
   - $(\mathsf{let}\, x := P \,\mathsf{in}\, Q)^{\bullet\circ} = ((\lambda x.Q^\bullet)P^\bullet)^\circ$. We distinguish two subcases:
     (a) $P = [V]$ and so $P^\bullet = [(V)^\bullet]$: $((\lambda x.Q^\bullet)[V^\bullet])^\circ = (\lambda x.Q^{\bullet\circ})V^{\bullet\circ} \overset{i.h.}{=}_{\beta,\mathsf{let}.\beta} (\lambda x.Q)V \mapsto_{c.\beta} Q\{V/x\} \;{}_{c.\mathsf{let}.\beta}\!\!\leftarrow\!\mapsto \mathsf{let}\, x := [V] \,\mathsf{in}\, Q$;
     (b) $P \neq [V]$ for any V: $((\lambda x.Q^\bullet)P^\bullet)^\circ = (\mathsf{let}\, x := P^{\bullet\circ} \,\mathsf{in}\, Q^{\bullet\circ}) \overset{i.h.}{=}_{\beta,\mathsf{let}.\beta} \mathsf{let}\, x := P \,\mathsf{in}\, Q$.
3. By induction over the reduction $M \to_{©\eta} N$ in $\lambda_{©\eta}$. First, we consider the root-steps.

   (β_c) If $(\lambda x.N)[V] \mapsto_{\beta_c} N\{V/x\}$ then $((\lambda x.N)[[]V])^\circ = (\lambda x.N^\circ)V^\circ \mapsto_{c.\beta} N^\circ\{V^\circ/x\} = (N\{V/x\})^\circ$ by substitutivity of $^\circ$.

(id) If $(\lambda x.[x])M \mapsto_{\mathsf{id}} M$ then there are two subcases:

  (a) if $M = [V]$ for some value V, then $((\lambda x.[x])M)^\circ = ((\lambda x.[x])[V])^\circ = (\lambda x.[x])V^\circ \mapsto_{c.\beta} [V^\circ] = M^\circ$;

  (b) if $M \neq [V]$ for any value V, then $((\lambda x.[x])M)^\circ = \mathsf{let}\, x \coloneqq M^\circ \,\mathsf{in}\, [x] \mapsto_{c.\mathsf{let}.\eta} M^\circ$.

(η) If $\lambda x.V[x] \mapsto_{c.\eta} V$ with $x \notin \mathsf{fv}(V)$, then $(\lambda x.V[x])^\circ = \lambda x.V^\circ x \mapsto_{c.\eta} V^\circ$.

(σ) $(\lambda y.N)((\lambda x.M)L) \mapsto_\sigma (\lambda x.(\lambda y.N)M)L$ with $x \notin \mathsf{fv}(N) \cup \{y\}$. We distinguish 4 cases:

  (a) if $L = [V]$ and $M = [W]$, then

$$\begin{aligned}
((\lambda y.N)((\lambda x.[W])[V]))^\circ &= \mathsf{let}\, y \coloneqq (\lambda x.[W^\circ])V^\circ \,\mathsf{in}\, N^\circ \\
&\underset{\mathsf{w}}{\rightarrow}_{c.\beta} \mathsf{let}\, y \coloneqq [W^\circ\{V^\circ/x\}] \,\mathsf{in}\, N^\circ \\
&\rightarrow_{c.\mathsf{let}.\beta} N^\circ\{W^\circ\{V^\circ/x\}/y\} \\
&{}_{c.\beta}\!\mapsfrom (\lambda y.N^\circ)W^\circ\{V^\circ/x\} \\
&{}_{c.\beta}\!\mapsfrom (\lambda x.((\lambda y.N^\circ)W^\circ))V^\circ \\
&= ((\lambda x.(\lambda y.N)[W])[V])^\circ;
\end{aligned}$$

  (b) if $L = [V]$ and $M \neq [W]$, then

$$\begin{aligned}
((\lambda y.N)((\lambda x.M)[V]))^\circ &= \mathsf{let}\, y \coloneqq (\lambda x.M^\circ)V^\circ \,\mathsf{in}\, N^\circ \\
&\underset{\mathsf{w}}{\rightarrow}_{c.\beta} \mathsf{let}\, y \coloneqq M^\circ\{V^\circ/x\} \,\mathsf{in}\, N^\circ \\
&{}_{c.\beta}\!\mapsfrom (\lambda x.(\mathsf{let}\, y \coloneqq M^\circ \,\mathsf{in}\, N^\circ))V^\circ \\
&= ((\lambda x.(\lambda y.N)M)[V])^\circ;
\end{aligned}$$

  (c) if $L \neq [V]$ and $M = [W]$, then

$$\begin{aligned}
((\lambda y.N)((\lambda x.[W])L))^\circ &= \mathsf{let}\, y \coloneqq (\mathsf{let}\, x \coloneqq L^\circ \,\mathsf{in}\, [W^\circ]) \,\mathsf{in}\, N^\circ \\
&\mapsto_{c.\mathsf{let}.ass} \mathsf{let}\, x \coloneqq L^\circ \,\mathsf{in}\, (\mathsf{let}\, y \coloneqq [W^\circ] \,\mathsf{in}\, N^\circ) \\
&\underset{\neg\mathsf{w}}{\rightarrow}_{c.\mathsf{let}.\beta} \mathsf{let}\, x \coloneqq L^\circ \,\mathsf{in}\, N^\circ\{W^\circ/y\} \\
&{}_{c.\beta}\!\leftarrow \mathsf{let}\, x \coloneqq L^\circ \,\mathsf{in}\, (\lambda y.N^\circ)W^\circ \\
&= ((\lambda x.(\lambda y.N)[W])L)^\circ;
\end{aligned}$$

  (d) if $L \neq [V]$ and $M \neq [W]$ for any values V, W, then

$$\begin{aligned}
(\lambda y.N)((\lambda x.M)L)^\circ &= \mathsf{let}\, y \coloneqq (\mathsf{let}\, x \coloneqq L^\circ \,\mathsf{in}\, M^\circ) \,\mathsf{in}\, N^\circ \\
&\mapsto_{c.\mathsf{let}.ass} \mathsf{let}\, x \coloneqq L^\circ \,\mathsf{in}\, (\mathsf{let}\, y \coloneqq M^\circ \,\mathsf{in}\, N^\circ) \\
&= ((\lambda x.(\lambda y.N)M)L)^\circ
\end{aligned}$$

Let us now consider the inductive cases.

Application right: If $M = VM' \rightarrow_\rho VN' = N$ with $M' \rightarrow_\rho N'$ and $\rho \in \{ml^* \setminus \eta, \eta\}$, then $M'^\circ =_\rho N'^\circ$ by *i.h.*, and there are five subcases:

- if $M' = [W]$, then $N' = [W_0]$ for some value W_0 and hence $M^\circ = V^\circ W^\circ =_\rho V^\circ W_0^\circ = N^\circ$;
- if $V = x$ and $M' \neq [W] \neq N'$ for any value W, then $M^\circ = \mathsf{let}\, y \coloneqq M'^\circ \,\mathsf{in}\, xy =_\rho \mathsf{let}\, y \coloneqq N'^\circ \,\mathsf{in}\, xy = N^\circ$;

- if $V = x$ and $M' \neq [W]$ but $N' = [W]$ for some value W, then $M = xM'$ and $N = x\,[W]$, hence $M^\circ = \mathsf{let}\,y := M'^\circ\,\mathsf{in}\,xy =_\rho \mathsf{let}\,y := N'^\circ\,\mathsf{in}\,xy = \mathsf{let}\,y := [W^\circ]\,\mathsf{in}\,xy \to_{\mathsf{c.let}.\beta} x\,W^\circ = N^\circ$;
- if $V = \lambda x.L$ and $M' \neq [W] \neq N'$ for any value W, then $M^\circ = \mathsf{let}\,x := M'^\circ\,\mathsf{in}\,L^\circ =_\rho \mathsf{let}\,x := N'^\circ\,\mathsf{in}\,L^\circ = N^\circ$;
- if $V = \lambda x.L$ and $M' \neq [W]$ but $N' = [W]$ for some value W, then $M = (\lambda x.L)M'$ and $N = (\lambda x.L)[W]$, hence $M^\circ = \mathsf{let}\,x := M'^\circ\,\mathsf{in}\,L^\circ =_\rho \mathsf{let}\,x := N'^\circ\,\mathsf{in}\,L^\circ = \mathsf{let}\,x := [W^\circ]\,\mathsf{in}\,L^\circ \mapsto_{\mathsf{c.let}.\beta} L^\circ\{W^\circ/x\} \;{}_{c.\beta}\!\!\leftarrow\!\dashv (\lambda x.L^\circ)W^\circ = N^\circ$.

Application left: If $M = (\lambda x.M')[V] \to_\rho (\lambda x.N')[V] = N$ with $M' \to_\rho N'$ and $\rho \in \{ml^* \smallsetminus \eta, \eta\}$, then $M'^\circ =_\rho N'^\circ$ by *i.h.*, and hence $M^\circ = (\lambda x.M'^\circ)V^\circ =_\rho (\lambda x.N'^\circ)V^\circ = N^\circ$.

Unit: If $M = [\lambda x.M'] \to_\rho [\lambda x.N'] = N$ with $M' \to_\rho N'$ and $\rho \in \{ml^* \smallsetminus \eta, \eta\}$, then $M'^\circ =_\rho N'^\circ$ by *i.h.*, so $M^\circ = [\lambda x.M'^\circ] =_\rho [\lambda x.N'^\circ] = N^\circ$.

4. By induction over the reduction $P \to_{ml^*} Q$ in λ_{ml^*}.

   First, we consider the root-steps.

   ($c.\beta$) If $(\lambda x.P)V \mapsto_{c.\beta} P\{V/x\}$ then $((\lambda x.P)V)^\bullet = (\lambda x.P^\bullet)[V^\bullet] \mapsto_{\beta_c} P^\bullet\{V^\bullet/x\} = (P\{V/x\})^\bullet$ by substitutivity of $(\cdot)^\bullet$.

   ($c.\eta$) If $\lambda x.Vx \mapsto_{c.\eta} V$ with $x \notin \mathsf{fv}(V)$ then $(\lambda x.Vx)^\bullet = \lambda x.V^\bullet[x] \mapsto_\eta V^\bullet$.

   ($c.\mathsf{let}.\beta$) If $\mathsf{let}\,x := [V]\,\mathsf{in}\,P \mapsto_{\mathsf{c.let}.\beta} P\{V/x\}$ then $(\mathsf{let}\,x := [V]\,\mathsf{in}\,P)^\bullet = (\lambda x.P^\bullet)[V^\bullet] \mapsto_{\beta_c} P^\bullet\{V^\bullet/x\} = (P\{V/x\})^\bullet$ by substitutivity of $(\cdot)^\bullet$.

   ($c.\mathsf{let}.\eta$) If $\mathsf{let}\,x := P\,\mathsf{in}\,[x] \mapsto_{\mathsf{c.let}.\eta} P$ then $(\mathsf{let}\,x := P\,\mathsf{in}\,[x])^\bullet = (\lambda x.!x)P^\bullet \mapsto_{\mathsf{id}} P^\bullet$.

   ($c.\mathsf{let}.ass$) If $\mathsf{let}\,y := (\mathsf{let}\,x := L\,\mathsf{in}\,P)\,\mathsf{in}\,Q \mapsto_{\mathsf{c.let}.ass} \mathsf{let}\,x := L\,\mathsf{in}\,(\mathsf{let}\,y := P\,\mathsf{in}\,Q)$, then $(\mathsf{let}\,y := (\mathsf{let}\,x := L\,\mathsf{in}\,P)\,\mathsf{in}\,Q)^\bullet = (\lambda y.Q^\bullet)((\lambda x.P^\bullet)L^\bullet) \mapsto_\sigma (\lambda x.(\lambda y.Q^\bullet)P^\bullet)L^\bullet = (\mathsf{let}\,x := L\,\mathsf{in}\,(\mathsf{let}\,y := P\,\mathsf{in}\,Q))^\bullet$.

   Let us consider now the inductive cases.

   Explicit substitution left: If $\mathsf{let}\,x := P\,\mathsf{in}\,Q \to_{ml^*} \mathsf{let}\,x := P'\,\mathsf{in}\,Q$ and $P \to_{ml^*} P'$, then $P^\bullet \to_{\copyright\eta} P'^\bullet$ by *i.h.*, and hence $(\mathsf{let}\,x := P\,\mathsf{in}\,Q)^\bullet = (\lambda x.Q^\bullet)P^\bullet \to_{\copyright\eta} (\lambda x.Q^\bullet)P'^\bullet = (\mathsf{let}\,x := P'\,\mathsf{in}\,Q)^\bullet$.

   Explicit substitution right: If $\mathsf{let}\,x := Q\,\mathsf{in}\,P \to_{ml^*} \mathsf{let}\,x := Q\,\mathsf{in}\,P'$ and $P \to_{ml^*} P'$, then $P^\bullet \to_{\copyright\eta} P'^\bullet$ by *i.h.*, and hence $(\mathsf{let}\,x := Q\,\mathsf{in}\,P)^\bullet = (\lambda x.P^\bullet)Q^\bullet \to_{\copyright\eta} (\lambda x.P'^\bullet)Q^\bullet = (\mathsf{let}\,x := Q\,\mathsf{in}\,P)^\bullet$.

   Application left: If $(\lambda x.P)V \to_{ml^*} (\lambda x.P')V$ and $P \to_{ml^*} P'$, then $P^\bullet \to_{\copyright\eta} P'^\bullet$ by *i.h.*, and hence $((\lambda x.P)V)^\bullet = (\lambda x.P^\bullet)[V^\bullet] \to_{\copyright\eta} (\lambda x.P'^\bullet)[V^\bullet] = ((\lambda x.P')V)^\bullet$.

   Application right: If $V(\lambda x.P) \to_{ml^*} V(\lambda x.P')$ and $P \to_{ml^*} P'$, then $P^\bullet \to_{\copyright\eta} P'^\bullet$ by *i.h.*, and so $(V(\lambda x.P))^\bullet = V^\bullet\,[\lambda x.P^\bullet] \to_{\copyright\eta} V^\bullet\,[\lambda x.P'^\bullet] = (V(\lambda x.P'))^\bullet$.

   Unit: If $[\lambda x.P] \to_{ml^*} [\lambda x.P']$ and $P \to_{ml^*} P'$, then $P^\bullet \to_{\copyright\eta} P'^\bullet$ by *i.h.*, and hence $[\lambda x.P]^\bullet = [\lambda x.P^\bullet] \to_{\copyright\eta} [\lambda x.P'^\bullet] = [\lambda x.P']^\bullet$. ☐

Remark C.1.2. Proposition C.1.1.4 can be refined to weak reduction in the following sense: if $P \underset{\mathsf{w}}{\rightarrow}_{ml^* \smallsetminus \eta} Q$ then $P^\bullet \underset{\mathsf{w}}{\rightarrow}_{\copyright} Q^\bullet$; if $P \underset{\mathsf{w}}{\rightarrow}_{\copyright\eta} Q$ then $P^\bullet \underset{\mathsf{w}}{\rightarrow}_{\copyright\eta} Q^\bullet$.

C.2 $(\boldsymbol{Com}, \to_{\beta_c})$ and $(\boldsymbol{Com}^v, \to_{\beta_v})$ are Isomorphic

To establish the isomorphism between $(Com, \to_{\beta_c})$ (the fragment of $\lambda_{©}$ with β_c as unique reduction rule) and $(Com^v, \to_{\beta_v})$ (the *kernel* of Plotkin's CbV calculus defined in Section 3.2.2), consider the translation $(\cdot)^\bullet$ from *Term* to *Term*^{v}, and, conversely, the translation $(\cdot)^\circ$ from *Term*^{v} to *Term*, where $Term^v := Val^v \cup Com^v$.

	$(\cdot)^\bullet : Term \to Term^v$	$(\cdot)^\circ : Term^v \to Term$
variables	$(x)^\bullet = x$	$(x)^\circ = [x]$
abstraction	$(\lambda x.P)^\bullet = \lambda x.(P)^\bullet$	$(\lambda x.M)^\circ = [\lambda x.(M)^\circ]$
returned values	$([V])^\bullet = V^\bullet$	
bind/application	$(VP)^\bullet = V^\bullet P^\bullet$	$(VP)^\circ = \begin{cases} xP^\circ & \text{if } V = x \\ (\lambda x.Q^\circ)P^\circ & \text{if } V = \lambda x.Q \end{cases}$

Essentially, the translation $(\cdot)^\bullet$ simply forgets the operator $[\cdot]$, and dually the translation $(\cdot)^\circ$ inserts each value that is not in the functional position of an application in the operator $[\cdot]$. These two translations form an isomorphism.

Proposition C.2.1.

1. *$(M^\circ)^\bullet = M$ for every term M in Term^v;*
2. *$(P^\bullet)^\circ = P$ for every term P in Term;*
3. *$M \to_{\beta_v} N$ implies $M^\circ \to_{\beta_c} N^\circ$;*
4. *$P \to_{\beta_c} Q$ implies $P^\bullet \to_{\beta_v} Q^\bullet$.*

Proof. Immediate by definition unfolding. □

C.3 Equational Correspondence between $\lambda_{©}$ and Moggi's λ_C-calculus

Definition C.3.1 (Values and computations). *The* Moggi's computational λ-calculus*, shortly λ_C, is a calculus of two sorts of expressions:*

$$\begin{array}{rrcl} Terms: & e, e' & ::= & v \mid n \\ Values: & v & ::= & x \mid \lambda x.e \\ NonValues: & n & ::= & \mathsf{let}\, x := e \,\mathsf{in}\, e' \mid ee' \end{array}$$

Definition C.3.2 (Reduction)**.** *The reduction relation $> \subseteq$ Term $\times$ Term is defined as follows:*

$$\begin{array}{lrcl} \beta_v & (\lambda x.e)v & > & e\{v/x\} \\ \eta_v & \lambda x.vx & > & v \\ id & \mathsf{let}\, x := e \,\mathsf{in}\, x & > & e \\ comp & (\mathsf{let}\, x_2 := (\mathsf{let}\, x_1 := e_1 \,\mathsf{in}\, e_2) \,\mathsf{in}\, e) & > & \\ & (\mathsf{let}\, x_1 := e_1 \,\mathsf{in}\, (\mathsf{let}\, x_2 := e_2 \,\mathsf{in}\, e)) & & \\ let_v & (\mathsf{let}\, x := v \,\mathsf{in}\, e) & > & e\{v/x\} \\ let_{.1} & ne & > & (\mathsf{let}\, x := n \,\mathsf{in}\, xe) \\ let_{.2} & vn & > & (\mathsf{let}\, x := n \,\mathsf{in}\, vx) \end{array}$$

where $e\{v/x\}$ denotes the capture avoiding substitution of v for all free occurrences of x in e.

Definition C.3.3 (Compatible Closure)**.** *Let $e, e_1, e_2 \in Term$ and suppose $e_1 > e_2$:*

$$\begin{array}{rcl} \lambda x.e_1 & > & \lambda x.e_2 \\ e_1 e > e_2 e & & e e_1 > e e_2 \\ \mathsf{let}\, x := e_1 \,\mathsf{in}\, e & > & \mathsf{let}\, x := e_2 \,\mathsf{in}\, e \\ \mathsf{let}\, x := e \,\mathsf{in}\, e_1 & > & \mathsf{let}\, x := e \,\mathsf{in}\, e_2 \end{array}$$

C.3.1 Translation of $\lambda_{©}$ into λ_C

Define a function from $\lambda_{©}$-*Term* to λ_C-*Term* as follows:
$(\cdot)^\circ : \lambda_{©} - Term \to \lambda_C - Term$

$$\begin{array}{ll} (x)^\circ \equiv x & (\lambda x.M)^\circ \equiv \lambda x.(M)^\circ \\ ([V])^\circ \equiv (V)^\circ & (M \star V)^\circ \equiv \mathsf{let}\, x := (M)^\circ \,\mathsf{in}\, (V)^\circ x \text{ (for } x \text{ fresh)} \end{array}$$

Lemma C.3.4 (Substitution lemma for $(\cdot)^\circ$)**.** *Let $M, V, W \in \lambda_{©}$-Term*

(i) $(V\{W/x\})^\circ \equiv (V)^\circ\{(W)^\circ/x\}$

(ii) $(M\{W/x\})^\circ \equiv (M)^\circ\{(W)^\circ/x\}$

Proof. By induction on the complexity of $\lambda_{©}$-*Term*:

$$\begin{array}{rcll} (x\{W/x\})^\circ & \equiv & (W)^\circ \equiv (x)^\circ\{(W)^\circ/x\} & \\ (x\{W/x\})^\circ & \equiv & y \equiv (y)^\circ\{(W)^\circ/x\} & \text{where } y \not\equiv x \\ ((\lambda z.M)\{W/x\})^\circ & \equiv & (\lambda z.M\{W/x\})^\circ & \\ & \equiv & \lambda z.(M\{W/x\})^\circ & \\ & \equiv & \lambda z.(M)^\circ\{(W)^\circ/x\} & byi.h. \\ & \equiv & (\lambda z.M)^\circ\{(W)^\circ/x\} & \end{array}$$

$$\begin{array}{rcll} (([V])\{W/x\})^\circ & \equiv & ([V]\{W/x\})^\circ & \\ & \equiv & (V\{W/x\})^\circ & \text{by i.h.} \\ & \equiv & (V)^\circ\{(W)^\circ/x\} & \\ & \equiv & ([V])^\circ\{(W)^\circ/x\} & \\ ((M \star V)\{W/x\})^\circ & \equiv & (M\{W/x\} \star V\{W/x\})^\circ & \\ & \equiv & \mathsf{let}\, z := (M\{W/x\})^\circ \,\mathsf{in}\, (V\{W/x\})^\circ z & \\ & \equiv & \mathsf{let}\, z := (M)^\circ\{(W)^\circ/x\} \,\mathsf{in}\, (V)^\circ\{(W)^\circ/x\} z & \text{by i.h.} \\ & \equiv & (\mathsf{let}\, z := (M)^\circ \,\mathsf{in}\, (V)^\circ z)\{(W)^\circ/x\} & \\ & \equiv & ((M \star V)^\circ)\{(W)^\circ/x\} & \end{array}$$

□

Lemma C.3.5. *Let $M, N \in \lambda_{©}$-Term. If $M \to N$, then $(M)^\circ = (N)^\circ$, where $=$ stands for the convertibility relation induced by $>$.*

Proof. Proof by induction on the generation of $M \to N$.

(β_c) $([V]) \star (\lambda x.M) \to M\{V/x\}$

$$\begin{aligned}(([V]) \star (\lambda x.M))^\circ &\equiv \mathsf{let}\, z := ([V])^\circ \,\mathsf{in}\, ((\lambda x.M))^\circ z \\ &\equiv \mathsf{let}\, z := (V)^\circ \,\mathsf{in}\, (\lambda x.(M)^\circ) z\end{aligned}$$

Since $(V)^\circ \in \mathit{Values}$, one has:

$$\begin{aligned}\mathsf{let}\, z := (V)^\circ \,\mathsf{in}\, (\lambda x.(M)^\circ) z &> (\lambda x.(M)^\circ)(V)^\circ && \text{by } \mathit{let}_v \text{ and } z \notin \mathsf{fv}((M)^\circ) \\ &> (M)^\circ\{(V)^\circ/x\} && \text{by } \beta_v \\ &\equiv (M\{V/x\})^\circ && \text{by substitution lemma C.3.4}\end{aligned}$$

(id) $M \star \lambda x.[x] \to M$

$$\begin{aligned}(M \star \lambda x.[x])^\circ &\equiv \mathsf{let}\, z := (M)^\circ \,\mathsf{in}\, ((\lambda x.[x]))^\circ z \\ &\equiv \mathsf{let}\, z := (M)^\circ \,\mathsf{in}\, (\lambda x.x) z \\ &> \mathsf{let}\, z := (M)^\circ \,\mathsf{in}\, z > (M)^\circ \text{ by } \lambda_C\text{-}\mathit{id}\end{aligned}$$

(ass) $(L \star \lambda x.M) \star (\lambda y.N) \to L \star \lambda x.(M \star \lambda y.N)$ where $x \notin \mathsf{fv}(N)$

$$\begin{aligned}((L \star \lambda x.M) \star (\lambda y.N))^\circ &\equiv \mathsf{let}\, z_2 := ((L \star \lambda x.M))^\circ \,\mathsf{in}\, ((\lambda y.N))^\circ z_2 \\ &\equiv \mathsf{let}\, z_2 := \mathsf{let}\, z_1 := (L)^\circ \,\mathsf{in}\, (\lambda x.M)^\circ z_1 \,\mathsf{in}\, ((\lambda y.N))^\circ z_2 \\ &> \mathsf{let}\, z_1 := (L)^\circ \,\mathsf{in}\, (\mathsf{let}\, z_2 := (\lambda x.(M)^\circ z_1) \,\mathsf{in}\, ((\lambda y.N))^\circ z_2) \\ &\quad \text{by } \lambda_C\text{-comp} \\ &\equiv \mathsf{let}\, z_1 := (L)^\circ \,\mathsf{in}\, (\mathsf{let}\, z_2 := (\lambda x.(M)^\circ) z_1 \,\mathsf{in}\, ((\lambda y.N))^\circ z_2) \\ &\equiv \mathsf{let}\, z_1 := (L)^\circ \,\mathsf{in}\, (\mathsf{let}\, z_2 := (M\{z_1/x\})^\circ \,\mathsf{in}\, ((\lambda y.N))^\circ z_2) \\ &< \mathsf{let}\, z_1 := (L)^\circ \,\mathsf{in}\, (\lambda x.\mathsf{let}\, z_2 := (M)^\circ \,\mathsf{in}\, ((\lambda y.N))^\circ z_2) z_1 \\ &\equiv (L \star \lambda x.(M \star \lambda y.N))^\circ\end{aligned}$$

Since $z \not\equiv z_2$, $x \notin \mathsf{fv}(N)$ and then $x \notin \mathsf{fv}((N)^\circ)$. From this consideration, one has: $(\mathsf{let}\, z_2 := (M)^\circ \,\mathsf{in}\, ((\lambda y.N)^\circ) z_2)\{z_1/x\} \equiv \mathsf{let}\, z_2 := (M\{z_2/x\})^\circ \,\mathsf{in}\, (\lambda y.N)^\circ z_2$

□

Remark C.3.6. Reduction $\to$ over $\lambda_{©}$-*Term* has been defined devoid of η nor ξ rules. Both can be added by extending reduction relation over values:

$$\xi : \quad \frac{M \to N}{\lambda x.M \to \lambda x.N} \qquad \eta : \quad \lambda x.[x] \star V \to V \quad \text{if } x \notin \mathsf{fv}(V)$$

With respect to previous lemma, concerning ξ-rule: $(M)^\circ = (N)^\circ$ by induction hypothesis and $(\lambda x.M)^\circ = \lambda x.(M)^\circ = \lambda x.(N)^\circ = (\lambda x.N)^\circ$.
Concerning η:

$$\begin{aligned}(\lambda x.[x] \star V)^\circ &\equiv \lambda x.\mathsf{let}\, z := x \,\mathsf{in}\, (V)^\circ z \\ &> \lambda x.(V)^\circ x && \text{by } \mathit{let}_v \\ &> (V)^\circ && \text{by } \eta_v\end{aligned}$$

C.3.2 Translation of λ_C into $\lambda_{©}$

Define a function from λ_C-*Term* to $\lambda_{©}$-*Term* as follows:
$(\cdot)^{\bullet Val} : \lambda_C - Values \to Val \qquad (\cdot)^{\bullet Com} : \lambda_C - NonValues \to Com$

$$(x)^{\bullet Val} = x$$

$$(\lambda x.v)^{\bullet Val} = \lambda x.[(v)^{\bullet Val}] \qquad\qquad (\lambda x.n)^{\bullet Val} = \lambda x.(n)^{\bullet Com}$$

$$\begin{array}{ll}
(vn)^{\bullet Com} = (n)^{\bullet Com} \star (v)^{\bullet Val} & \\
(vv')^{\bullet Com} = [(v')^{\bullet Val}] \star (v)^{\bullet Val} & \\
(nv)^{\bullet Com} = (n)^{\bullet Com} \star (\lambda x.[(v)^{\bullet Val}] \star x) & \text{for fresh } x \\
(nn')^{\bullet Com} = (n)^{\bullet Com} \star (\lambda x.(n')^{\bullet Com} \star x) & \text{for fresh } x \\
(\mathsf{let}\, x := n \,\mathsf{in}\, n')^{\bullet Com} = (n)^{\bullet Com} \star \lambda x.(n')^{\bullet Com} & \\
(\mathsf{let}\, x := v \,\mathsf{in}\, n')^{\bullet Com} = [(v)^{\bullet Val}] \star \lambda x.(n')^{\bullet Com} & \\
(\mathsf{let}\, x := n \,\mathsf{in}\, v)^{\bullet Com} = (n)^{\bullet Com} \star \lambda x.[(v)^{\bullet Val}] & \\
(\mathsf{let}\, x := v \,\mathsf{in}\, v')^{\bullet Com} = [(v)^{\bullet Val}] \star \lambda x.[(v')^{\bullet Val}] &
\end{array}$$

Lemma C.3.7 (Substitution lemma for $(\cdot)^{\bullet}$). *Let $w, v \in Values$ and $n \in NonValues$;*

(i) $(w\{v/x\})^{\bullet Val} \equiv (w)^{\bullet Val}\{(v)^{\bullet Val}/x\}$

(ii) $(n\{Wv/x\})^{\bullet Com} \equiv (n)^{\bullet Com}\{(v)^{\bullet Val}/x\}$

Proof. (i)

$w \equiv x$:

$$(x\{v/x\})^{\bullet Val} = (v)^{\bullet Val} = (x)^{\bullet Val}\{(v)^{\bullet Val}/x\}$$

$w \equiv y \not\equiv x$:

$$(y\{v/x\})^{\bullet Val} = (y)^{\bullet Val} = y = (y)^{\bullet Val}\{(v)^{\bullet Val}/x\}$$

$w \equiv \lambda y.w$, where $y \not\equiv x$ and $y \notin \mathsf{fv}(w)$:

$$\begin{array}{lll}
(\lambda y.w\{v/x\})^{\bullet Val} & = (\lambda y.(w\{v/x\}))^{\bullet Val} & \\
& = \lambda y.[(w\{v/x\})^{\bullet Val}] & \\
& = \lambda y.[(w)^{\bullet Val}\{(v)^{\bullet Val}/x\}] & \text{by i.h.} \\
& = (\lambda y.[(w)^{\bullet Val}])\{(v)^{\bullet Val}/x\} & \\
& = (\lambda y.w)^{\bullet Val}\{(v)^{\bullet Val}/x\} &
\end{array}$$

$w \equiv \lambda y.n$:

$$\begin{array}{lll}
((\lambda y.n)\{v/x\})^{\bullet Val} & = (\lambda y.(n\{v/x\}))^{\bullet Val} & \\
& = \lambda y.(n\{v/x\})^{\bullet Com} & \\
& = \lambda y.(n)^{\bullet Com}\{(v)^{\bullet Val}/x\} & \text{by i.h.} \\
& = (\lambda y.(n)^{\bullet Com})\{(v)^{\bullet Val}/x\} & \\
& = (()^{\bullet Val}\lambda y.n)\{(v)^{\bullet Val}/x\} &
\end{array}$$

(ii)

$n \equiv wn'$:

$$\begin{aligned}
((wn')\{v/x\})^{\bullet Com} &= ((w\{v/x\})(n'\{v/x\}))^{\bullet Com} \\
&= (n'\{v/x\})^{\bullet Com} \star (w\{v/n\})^{\bullet Val} \\
&= (n')^{\bullet Com}\{(v)^{\bullet Val}/x\} \star (w)^{\bullet Val}\{(v)^{\bullet Val}/x\} \quad \text{by i.h.} \\
&= ((n')^{\bullet Com} \star (w)^{\bullet Val})\{(v)^{\bullet Val}/x\} \\
&= (wn')^{\bullet Com}\{(v)^{\bullet Val}/x\}
\end{aligned}$$

$n \equiv ww'$

$$\begin{aligned}
((ww')\{v/x\})^{\bullet Val} &= ((w\{v/x\})(w'\{v/x\}))^{\bullet Val} \\
&= [(w'\{v/x\})^{\bullet Val}] \star (w\{v/n\})^{\bullet Val} \\
&= [(w')^{\bullet Val}\{(v)^{\bullet Val}/x\}] \star (w)^{\bullet Val}\{(v)^{\bullet Val}/x\} \quad \text{by i.h.} \\
&= ((w')^{\bullet Val} \star (w)^{\bullet Val})\{(v)^{\bullet Val}/x\} \\
&= (ww')^{\bullet Com}\{(v)^{\bullet Val}/x\}
\end{aligned}$$

$n \equiv n'w$

$$\begin{aligned}
((n'w)\{v/x\})^{\bullet Com} &= ((n'\{v/x\})(w\{v/x\}))^{\bullet Com} \\
&= (n'\{v/x\})^{\bullet Com} \star (\lambda y.[(w\{v/n\})^{\bullet Val}] \star y) \\
&= (n')^{\bullet Com}\{(v)^{\bullet Val}/x\} \star (\lambda y.[(w)^{\bullet Val}\{(v)^{\bullet Val}/n\}] \star y) \\
&= ((n')^{\bullet Com} \star (\lambda y.[(w)^{\bullet Val}] \star y))\{(v)^{\bullet Val}/x\} \\
&= (n'w)^{\bullet Com}\{(v)^{\bullet Val}/x\}
\end{aligned}$$

$n \equiv n'm$

$$\begin{aligned}
((n'm)\{v/x\})^{\bullet Com} &= ((n'\{v/x\})(m\{v/x\}))^{\bullet Com} \\
&= (n'\{v/x\})^{\bullet Com} \star (\lambda y.(m\{v/n\})^{\bullet Com} \star y) \\
&= (n')^{\bullet Com}\{(v)^{\bullet Val}/x\} \star (\lambda y.(m)^{\bullet Com}\{(v)^{\bullet Val}/n\} \star y) \\
&= ((n')^{\bullet Com} \star (\lambda y.(m)^{\bullet Com} \star y)\{(v)^{\bullet Val}/n\} \\
&= (n'm)^{\bullet Com}\{(v)^{\bullet Val}/n\}
\end{aligned}$$

$n \equiv (\mathsf{let}\, y := m \,\mathsf{in}\, n')$

$$\begin{aligned}
((\mathsf{let}\, y := m \,\mathsf{in}\, n')\{v/x\})^{\bullet Com} &= ((\mathsf{let}\, y := m\{v/x\} \,\mathsf{in}\, n'\{v/x\}))^{\bullet Com} \\
&= (m\{v/x\})^{\bullet Com} \star \lambda y.(n'\{v/x\})^{\bullet Com} \\
&= (m)^{\bullet Com}\{(v)^{\bullet Val}/x\} \star \lambda y.(n')^{\bullet Com}\{(v)^{\bullet Val}/x\} \quad \text{by i.h.} \\
&= ((m)^{\bullet Com} \star \lambda y.(n')^{\bullet Com})\{(v)^{\bullet Val}/x\} \\
&= (\mathsf{let}\, y := m \,\mathsf{in}\, n')^{\bullet Com}\{(v)^{\bullet Val}/x\}
\end{aligned}$$

Similarly, the statement is proved for remaining cases. □

Definition C.3.8. *Let's define a function $(\cdot)^\bullet$ that maps λ_C-Term into $\lambda_{\circledcirc}$-Com, as follows:*
For all $n \in NonValues$ and $v \in Values$

$$(n)^\bullet = (n)^{\bullet Com} \quad (v)^\bullet = (v)^{\bullet Val}$$

Corollary C.3.9. $(e\{v/x\})^\bullet = (e)^\bullet\{(v)^{\bullet Val}/x\}$

Proof. By induction on the structure of $e \in \lambda_C\text{-}Term$:
$e \equiv w \in Values$, then $(w\{v/x\})^\bullet = [(w\{v/x\})^{\bullet Val}] = [(w)^{\bullet Val}\{(v)^{\bullet Val}/x\}]$ by part *(i)* of C.3.7. This is equivalent to $([(w)^{\bullet Val}])\{(v)^{\bullet Val}/x\}$, that is equivalent to $(w)^\bullet\{(v)^{\bullet Val}/x\}$ hence the thesis by definition C.3.8.
If $e \equiv n \in NonValues$:

$$\begin{aligned}(n\{v/x\})^\bullet &= (n\{v/x\})^{\bullet Com}\\ &= (n)^{\bullet Com}\{(v)^{\bullet Val}/x\} \quad \text{by part } \textit{(ii)} \text{ of C.3.7}\\ &= (n)^\bullet\{(v)^{\bullet Val}/x\}\end{aligned}$$

□

Lemma C.3.10. $e > e' \Rightarrow (e)^\bullet \xrightarrow{*} (e')^\bullet$

Proof. By induction on the definition of $>$:

$$\begin{aligned}&((\lambda x.w)v)^\bullet && = ((\lambda x.w)v)^{\bullet Com}\\ &= [(v)^{\bullet Val}] \star \lambda x.[(w)^{\bullet Val}] && \to_{\beta_c} ([(w)^{\bullet Val}])\{(v)^{\bullet Val}/x\}\\ &&& = (w)^\bullet\{(v)^{\bullet Val}/x\} \qquad \text{by definition C.3.8}\\ &&& = (w\{v/x\})^\bullet \qquad \text{by the previous corollary}\end{aligned}$$

$$\begin{aligned}&((\lambda x.n)v)^\bullet && = ((\lambda x.n)v)^{\bullet Com}\\ &= [(v)^{\bullet Val}] \star \lambda x.(n)^{\bullet Com} && \to_{\beta_c} (n)^{\bullet Com}\{(v)^{\bullet Val}/x\}\\ &&& = (n\{(v)^{\bullet Val}/x\})^{\bullet Com} \qquad \text{by part } \textit{(ii)} \text{ of C.3.7}\\ &&& = (n\{v/x\})^\bullet \qquad \text{by definition C.3.8}\end{aligned}$$

η_v: $(\lambda x.vx)^\bullet = \lambda x.(vx)^{\bullet Com} = \lambda x.[x] \star (v)^{\bullet Val} \to_\eta (v)^{\bullet Val}$
(id).1: $(\mathsf{let}\, x := v \,\mathsf{in}\, x) > v$

$$\begin{aligned}(\mathsf{let}\, x := v \,\mathsf{in}\, x)^\bullet &= (\mathsf{let}\, x := v \,\mathsf{in}\, x)^{\bullet Com}\\ &= [(v)^{\bullet Val}] \star \lambda x.[(x)^{\bullet Val}]\\ &= [(v)^{\bullet Val}] \star \lambda x.[x]\\ &\to_{(id)} [(v)^{\bullet Val}] = (v)^\bullet\end{aligned}$$

Regarding rule *(comp)*, the proof is done after proving 8 different cases, here we will show just two of them.

$$\begin{aligned}(\mathsf{let}\, y := \mathsf{let}\, x := m \,\mathsf{in}\, n \,\mathsf{in}\, n')^\bullet &= (\mathsf{let}\, y := \mathsf{let}\, x := m \,\mathsf{in}\, n \,\mathsf{in}\, n')^{\bullet Com}\\ &= ((m)^{\bullet Com} \star \lambda x.(n)^{\bullet Com}) \star \lambda y.(n')^{\bullet Com}\\ &\to_{(ass)} (m)^{\bullet Com} \star \lambda x.((n)^{\bullet Com} \star \lambda y.(n')^{\bullet Com})\\ &= (\mathsf{let}\, x := m \,\mathsf{in}\, \mathsf{let}\, y := n \,\mathsf{in}\, n')^\bullet\end{aligned}$$

$$\begin{aligned}(\mathsf{let}\, y := \mathsf{let}\, x := v \,\mathsf{in}\, n \,\mathsf{in}\, n')^\bullet &= ([(v)^{\bullet Val}] \star \lambda x.(n)^{\bullet Com}) \star \lambda y.(n')^{\bullet Com}\\ &\to_{(ass)} [(v)^{\bullet Val}] \star \lambda x.((n)^{\bullet Com}) \star \lambda y.(n')^{\bullet Com})\\ &= \mathsf{let}\, x := [(v)^{\bullet Val}] \,\mathsf{in}\, \mathsf{let}\, y := (n)^{\bullet Com} \,\mathsf{in}\, (n')^{\bullet Com}\\ &= (\mathsf{let}\, x := v \,\mathsf{in}\, \mathsf{let}\, y := n \,\mathsf{in}\, n')^\bullet\end{aligned}$$

let_v case splits in two different sub-cases:

$$\begin{aligned}
(\mathsf{let}\, x := v \,\mathsf{in}\, n)^{\bullet Com} &= [(v)^{\bullet Val}] \star \lambda x.(n)^{\bullet Com} \\
&\to_{\beta_v} (n)^{\bullet Com}\{(v)^{\bullet Val}/n\} \\
&= (n\{v/x\})^{\bullet} && \text{by part } \textit{(ii)} \text{ of C.3.7}
\end{aligned}$$

$$\begin{aligned}
(\mathsf{let}\, x := v \,\mathsf{in}\, w)^{\bullet Com} &= [(v)^{\bullet Val}] \star \lambda x.[(w)^{\bullet Val}] \\
&\to_{\beta_v} ([(w)^{\bullet Val}])\{(v)^{\bullet Val}/n\} \\
&= (w)^{\bullet}\{(v)^{\bullet Val}/n\} && \text{by definition of } (\cdot)^{\bullet Val} \\
&= (w\{v/x\})^{\bullet} && \text{by the previous corollary}
\end{aligned}$$

let_1 case splits in two different sub-cases:

$$\begin{aligned}
&(nv)^{\bullet Com} = (n)^{\bullet Com} \star \lambda x.[(v)^{\bullet Val}] \star x \\
&(\mathsf{let}\, x := n \,\mathsf{in}\, xv)^{\bullet Com} = (n)^{\bullet Com} \star \lambda x.(xv)^{\bullet Com} = (n)^{\bullet Com} \star (\lambda x.[(v)^{\bullet Val}] \star x)
\end{aligned}$$

Since the above equalities, one has proved that $(nv)^{\bullet Com} = (\mathsf{let}\, x := n \,\mathsf{in}\, xv)^{\bullet Com}$. Then, a fortiori $(nv)^{\bullet Com} \to^{*} (\mathsf{let}\, x := n \,\mathsf{in}\, xv)^{\bullet Com}$, since $\to^{*}$ is also a reflexive closure of $\to$.

$$\begin{aligned}
&(nm)^{\bullet Com} = (n)^{\bullet Com} \star \lambda x.(m)^{\bullet Com} \star x \\
&(\mathsf{let}\, x := n \,\mathsf{in}\, xm)^{\bullet Com} = (n)^{\bullet Com} \star \lambda x.(xm)^{\bullet Com} = (n)^{\bullet Com} \star (\lambda x.(m)^{\bullet Com} \star x)
\end{aligned}$$

The conclusion follows analogous considerations stated in the previous sub-case. □

Theorem C.3.11. *There exists an interpretation $(\cdot)^{\bullet}$ from λ_C into $\lambda_{©}$ that preserves reductions.*
There exists an interpretation $(\cdot)^{\circ}$ from $\lambda_{©}$ into λ_C that preserves the convertibility relation.