On Reasonable Space and Time Cost Models for the λ-Calculus

Gabriele Vanoni

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN
COMPUTER SCIENCE AND ENGINEERING

Ciclo XXXIV

On Reasonable Space and Time Cost Models for the λ-Calculus

Presentata da
Gabriele Vanoni

Supervisore
Prof. Ugo Dal Lago

Giuria composta da:
Commissaria: Prof.ssa Viviana Bono (Univ. di Torino)
Commissario: Prof. Luca Foschini (Univ. di Bologna)
Commissaria: Prof.ssa Viviana Patti (Univ. di Torino)
Revisore: Prof. Dan Ghica (Univ. of Birmingham)
Revisora: Prof.ssa Delia Kesner (Univ. Paris Cité)

Coordinatore Dottorato: Prof. Davide Sangiorgi
Settore Scientifico Disciplinare: INF/01-INFORMATICA
Settore Concorsuale: 01/B1-INFORMATICA

Esame Finale: 23 Giugno 2022

Abstract

The λ-calculus is considered the paradigmatic model for functional programming languages. It comes with a beautiful mathematical theory, which has been studied and improved for more than 80 years. The λ-calculus is based on just one rewriting rule, β-reduction. Although rewriting expressions is common in computer science, this is not the way in which programs are executed by the hardware. Hence, there is a distance between the programming language model and the execution model. This gap is closed by compilers, that translate high-level (functional) programs to low-level executable machine code. While the semantics of programs remains unaltered during compilation, the use of resources, in particular time and space, is more difficult to track.

First of all, one should be able to measure the use of resources on the source program. Then, this amount of used resources should be preserved by compilation. The definition of resource consumption is typically done on Turing machines (TMs), where time is simply the number of steps a Turing machine needs to halt, and space is the maximum number of tape cells used during the computation. We would like to define some time and space measures on top of the λ-calculus, and we would like them being compatible with those of Turing machines. More formally, Slot and van Emde Boas Invariance Thesis states that a time (respectively, space) cost model is reasonable for a computational model C if there are mutual simulations between TMs and C such that the overhead is polynomial in time (respectively, linear in space). The rationale is that under the Invariance Thesis, complexity classes such as L, P, PSPACE, become robust, *i.e.* machine independent. More concretely, we can see these simulations as consisting of a compilation phase, followed by an execution phase.

The literature on the subject contains a lot of results about time cost model for the λ-calculus. In particular, the number of rewriting steps has been proved a reasonable cost model in the majority of the interesting cases, *e.g.* in the call-by-name/value/need cases. For space cost models the situation is different: except for a recent partial result by Forster et al., nothing is known. Indeed, the problem is far more difficult w.r.t. the one of finding a reasonable time measure. The main reason of this difficulty is that the required overhead for the space simulations is linear, and not polynomial, *i.e.* the space consumption should be the same on both the the sides of the simulations, only up to a multiplicative constant factor. This is very difficult to achieve for two different reasons. The former is that typical implementations of the λ-calculus rely on pointers, that give an extra undesired logarithmic overhead. The latter, instead, is that in the λ-calculus there is not distinction between programs and data. This fact does not allow to account for sub-linear complexity classes, if one considers the natural space cost model, *i.e.* the maximum size of terms encountered during a reduction, that by definition is at least as big as the input (this is what Forster et al. do).

In this dissertation, we tackle this problem from different perspectives. We start by considering an unusual evaluation mechanism for the λ-calculus, based on Girard's Geometry of Interaction, that was conjectured to be the key ingredient to obtain a space reasonable cost model. By a fine complexity analysis of this schema, based on new variants of non-idempotent intersection types, we disprove this conjecture. Then, we change the target of our analysis. We consider a variant over Krivine's abstract machine, a standard evaluation mechanism for the call-by-name λ-calculus, optimized for space complexity, and implemented without any pointer. A fine analysis of the execution of (a refined version of) the encoding of TMs into the λ-calculus allows us to conclude that the space consumed by this machine is a reasonable space cost model. In particular, for the first time we are able to measure also sub-linear space complexities. Moreover, we transfer this result also to the call-by-value case.

Finally, we provide also an intersection type system that characterizes compositionally this new reasonable space measure. This is done through a minimal, yet non trivial, modification of the original de Carvalho type system.

Acknowledgments

As a disclaimer, this acknowledgment section will mention job related people, only. All the others already know.

This thesis would not have been possible without the commitment of Ugo Dal Lago, who has carefully supervised me. Moreover, Beniamino Accattoli has acted a second, although unofficial, supervisor. They have been terrific mentors.

Then, I would like to thank all my colleagues in Bologna, Damiano Mazza, who hosted me in Paris, and Zhong Shao, who hosted me at Yale.

Finally, I thank Dan Ghica and Delia Kesner, who accepted to review this thesis.

Of course, I have met many people during these years, and I would like to thank them all.

Contents

Chapter 1

Introduction

The λ-calculus has been around for more than 80 years but still attracts the attention of many researchers around the world. It was originally created by Church (1932) as a logical formal system. However, soon afterwards, Kleene and Rosser (1935) discovered an inconsistency and then only the *functional* part was kept. Church (1936a,b) himself used it as a tool in the first proof of undecidability for the *Entscheidungsproblem*. In the following years, with Turing an Kleene, they realized that, together with Turing machines (TMs) and Gödel/Kleene partial recursive functions, the λ-calculus was a universal model of computation (Kleene, 1936; Turing, 1937): this is the Church-Turing thesis. Then, it was studied until the fifties in a pure mathematical way (Curry and Feys, 1958). With the birth of computer science, the λ-calculus had new life, as it was now considered a *programming language* (Landin, 1964, 1965a,b; Plotkin, 1975). Logicians, mathematicians and computer scientists continued to study this formidable tool that over the years was given a strong mathematical theory (Barendregt, 1984) and a clear connection with logic, trough the celebrated Curry-Howard correspondence (Howard, 1980). At present time, the λ-calculus is considered as *the* paradigmatic *functional* programming language, and an unavoidable tool in the study of programming languages in general. We invite the reader who wants to delve into the history of the λ-calculus (and combinatory logic (Schönfinkel, 1924)) to read the detailed survey by Cardone and Hindley (2009).

Programming Languages and Computational Complexity. While the first years of *theoretical* computer science were devoted to discover *what* is computable, later the focus shifted to *how* things are computed. The *extensional* view, was substituted by an *intensional* one. In particular, the use of resources needed to execute an *algorithm/program* was investigated, and it is still now a very active research area. The attention is mainly devoted to the time and space (*i.e.* memory) consumption of algorithms (Hartmanis and Stearns, 1965). Traditionally, the analysis is done considering Turing machines (TMs), where time and space have a precise meaning:

- *time:* the number of transitions of the TM;
- *space:* the maximum number of cells written during the computation of the TM.

This is a very nice theoretical framework, but it is impractical, since no one writes programs as TMs. In fact, they are very low-level devices, designed for theoretical purposes, rather than *real* programming purposes. However, the definition of computational complexity classes, such as L, P, NP, PSPACE, is done on TMs, since they seem to be the *correct* tool to study complexity. This is because a TM transition (respectively, a cell of the TM tape), looks like an *atomic* unit of measure for time (respectively, space).

How can we reconcile the study of computational complexity with real programming languages? A first solution was the introduction of *Random Access Machines* (RAMs), theoretical devices which feature direct access memory, thus being closer to actual hardware than to TMs, whose memory (*i.e.* tapes) can be accessed only sequentially. RAMs can be programmed in an assembly-like language and provide this way a first level of abstraction. From the complexity point of view, is it sensible to consider the time and space consumption of RAMs as a *reasonable* measure of time and space complexity? Moreover, what does "reasonable" mean? The idea is that one would like to have complexity measures which are machine independent. Unfortunately, this is impossible, already if one considers different variants of TMs. For example, simulating a multi

tape TM which has time complexity $\mathcal{O}(f(n))$ on a single tape TM takes a quadratic overhead, *i.e.* it can be done with complexity $\mathcal{O}(f^2(n))$. If we are not able to preserve specific complexities, we would like at least to preserve the most important complexity classes. In this spirit, Slot and Emde Boas (1988) coined the *invariance thesis*, which states:

Invariance Thesis. *Reasonable machine models simulate each other with polynomially bounded overhead in time and (multiplicative) constant overhead in space.*

This way, complexity classes such as L, P, NP, PSPACE become machine independent. Indeed, both TMs (in all their variants) and RAMs[1] satisfy the invariance thesis. High-level language, however, are quite distant also from RAMs. The reader can think about programming styles and features such as *object orientation*, *higher-order functions*, *logic programming*, *first-class list manipulation*. It is not at all obvious how to measure time and space complexity of programming languages with these characteristics. As an example, `JAVA` quite recently introduced *lambda functions*, thus giving the programmer the possibility to use first-class (higher-order) functions. Since libraries for stream processing make a large use of this feature (the so-called *map-reduce* paradigm), one would like to have guarantees on the complexity of the programs. In particular, since very large amounts of data are processed, one would expect to have guarantees about space efficiency. This is why a reasonable space measure is required. Otherwise, one would not be able to estimate the space consumption at the level of the source code, but rather should look at how the language is implemented.

In this dissertation, we will deeply analyze the problem of how to measure the space complexity of λ-terms, *i.e.* of programs written in the λ-calculus, that, as we have already said, we consider the core of modern functional programming languages, such as `Haskell` and `OCaml`, and the model behind the functional parts of imperative languages such as `JAVA`, `Python` or `Scala`. The main feature of the λ-calculus is that *everything is a function*. Concrete data types can be added, but this does not typically add more intricacies. Most of the interesting aspects are already present in the *pure* fragment. We start by considering the problem (considered solved) of measuring the time complexity of λ-terms.

The Time of the λ-Calculus. The λ-calculus comes with just one computational rule, namely β-reduction. This can be however applied to different reducible expressions (or redexes) inside a λ-term, thus giving rise to a form of nondeterminism. This is extensionally harmless, because the λ-calculus can be proved confluent, which implies that the result is unique, when it exists. However, from an intensional point of view, we have different λ-calculi, depending on the reduction strategy chosen. For example, call-by-name and call-by-value are typically studied separately, as well as strong w.r.t. weak strategies.

We are interested in cost models where time is the number of of β-steps taken by a fixed strategy. Since they attribute the cost of *one* to every β-steps, these time cost models are called *unitary*. The simulation of TMs into the λ-calculus is not problematic, since there is an encoding of TMs into a strategy independent, deterministic fragment of the weak λ-calculus, due to Dal Lago and Accattoli (2017), for which many strategies[2] simulate Turing machines with a linear number of β-steps.

The delicate part of showing that a unitary time cost model is reasonable is the simulation of the λ-calculus strategy in TMs, or another reasonable model, typically RAMs. The difficulty stems from a degeneracy, called *size explosion* in the literature. The point is that the size of a term can grow exponentially during the evaluation (thus also requiring exponential time to be written down). To circumvent the exponential explosion in space, λ-terms are usually evaluated *up to sharing*, that is, in calculi with sharing constructs (a.k.a. explicit substitutions (ES) or `let in`) or abstract machines (AMs) that compute shared representations of the results. These representations can be exponentially smaller than the results themselves: explosiveness is then encapsulated in the

[1] For RAMs, one has to be quite careful, both on the instruction set and on how the time/space complexity is measured. In fact, if one considers RAMs with multiplication/division as primitive operations, as a corollary one has that P = NP Hartmanis and Simon, 1974. Moreover, as far as space complexity is concerned, the choice of the RAM cost model is non trivial. The different possibilities have different properties. In particular, some are reasonable and some are not Slot and Emde Boas, 1988.

[2] In particular, all weak strategies, but also head, leftmost-outermost, open and strong call-by-value and call-by-need.

sharing unfolding process (which itself has to satisfy some reasonable properties, see Accattoli and Dal Lago, 2016; Condoluci et al., 2019). The number of β steps, polynomially related to the complexity of calculi with ES or AM according to various evaluation strategies, then turns out to be a reasonable time cost model (up to sharing), despite explosiveness. The first such result is for weak evaluation by Blelloch and Greiner (1995), then extended to strong evaluation by Accattoli and Dal Lago (2016), and very recently transferred to Strong CbV independently by Accattoli et al. (2021a) and Biernacka et al. (2021). As we shall see, the very sharing mechanism that makes natural time reasonable is unfortunately space unreasonable. Accattoli (2017) has surveyed time cost models quite recently.

The Space of the λ-Calculus. In contrast with time, the problem of finding a reasonable space cost model for the λ-calculus is not considered solved. The natural space cost model is the maximum size of λ-terms belonging to the reduction sequence. However, as we have already said, this can be exponentially bigger than the length of the reduction itself. If one is concerned with space reasonableness *only*, *i.e.* not considering the execution time, then it is possible to simulate the reduction of λ-terms on TMs, as well as to simulate TMs into the λ-calculus, both with a constant multiplicative overhead. So, why is the problem still considered unsolved?

1. This simulation of the λ-calculus into TMs *cannot* accommodate the unitary cost model for time, because of size explosion. This problem has been tackled by Forster et al. (2020) in the case of the weak call-by-value λ-calculus, but the employed technique is very general and can be applied also to many other calculi. Their solution rests on the fact that given two simulations, one that is reasonable for space but not time, and one that is reasonable for time but not space, there is a smart way of interleaving them as to obtain reasonability for time and space *simultaneously*. Their result therefore shows that, surprisingly, a computational theory can be reasonable for time *and* space, also in the case of size explosion. Although theoretically correct, however, their solution does not seem canonical because of the non-uniform interleaving mechanism, that operates in rounds.

2. The natural space cost model *cannot* accommodate sub-linear space complexity, and thus cannot reflect, *e.g.*, algorithms in L. This is because if space is the maximum size of terms in an evaluation sequence, the first of which contains the input, then space simply *cannot be* sub-linear.

How could we account for *logarithmic* reasonable space? One needs, as it is done with TMs, *log-sensitivity*, that is, a distinction between an immutable *input space*, which is not counted for space complexity (because otherwise the complexity would be at least linear), and a (smaller) mutable *work space*, that is counted. Moreover, logarithmic space usually requires manipulating *pointers* to the input (which are of logarithmic size) rather than pieces of the input (which can be linear). Log-sensitivity thus seems to clash with the natural approach based on the rewriting of λ-terms, which does not distinguish between input and work space and that manipulates actual sub-terms rather than pointers.

Unfortunately, the common way in which λ-terms (and functional programs) are (time) efficiently evaluated, *i.e.* via sharing, seems to go against space reasonability. In fact, in general, one sharing annotation is required for every β-redex, this way entangling space with time. In other words, using calculi with sharing or abstract machines leads to a space consumption which is *linear* in time, thus preventing this mechanisms from being reasonable for space. This observation has guided many researchers to look at *different* implementation mechanisms, in order to obtain results about space. The literature, in this respect, reports many results about space complexity based on Girard's Geometry of Interaction.

Geometry of Interaction. The birth of linear logic (Girard, 1987) gave new light on the understanding of the λ-calculus and Geometry of Interaction (Girard, 1989a) (GoI) is one of its byproducts. At the time, GoI was a radically new interpretation of proofs, arising from connections between linear logic and functional analysis, and based on an abstract notion of interactive execution for proofs. Indeed, it has inspired Abramsky et al. (2000) game semantics, a new kind of interactive

semantics that allowed to solve the long-standing open problem of full-abstraction for models of PCF (Milner, 1977). The computational content of GoI was first explored by Danos and Regnier (1999) and Mackie (1995), who proposed a new form of implementation schema, called interaction abstract machine (IAM). The IAM works in a fundamentally different way with respect to environment based abstract machines, which are the standard and time efficient way of implementing functional languages. In particular, the IAM does not use environments, instead using a data structure called *token*, saving information about the history of the computation. The key point is that the token does not store information about every single β-redex, thus disentangling space consumption from time consumption. In other words, the IAM is a good candidate for a space reasonable implementation schema. The price to pay is that the machine wastes a lot of time to retrieve β-redexes, so that time is sacrificed for space.

GoI and Space Complexity. The IAM has been used in the literature for obtaining sub-linear space bounds for functional programs. To our knowledge, Schöpp (2006, 2007) was the first one to use ideas from the GoI to craft L-constrained programming languages, in the spirit of the Implicit Computational Complexity (ICC) program. Later, together with Dal Lago, they extended his results to cover a more practical programming language (Dal Lago and Schöpp, 2016). Soon after the first work by Schöpp, Ghica (2007) independently exploited ideas from the GoI and game semantics (GS), itself very related to the GoI, to design a compiler from a higher-order functional language directly to digital circuits, which have a *finite* amount of memory by definition. This was the first one of series of works titled "The Geometry of Synthesis", that applied semantic techniques such as the GoI and GS to the synthesis of digital circuits. Again belonging to the ICC program, Aubert et al. (2014) used the GoI to prove that a subset of Prolog is sound and complete for L. Mazza (2015) did the same for a substructurally typed programming language. These results have then suggested the folklore conjecture that the IAM space usage could be a reasonable space measure for the λ-calculus. Having a closer look at the cited results for space based on the IAM, however, one realizes that those bounds rely crucially on some tweaks (*i.e.* restricting to certain λ-terms or extending the language with ad-hoc constructs) and that they do not seem to be achievable on ordinary pure λ-terms.

GoI and Time Complexity. Concerning time consumption, it is known since the first works by Mackie (1995) and Danos et al. (1996) that the IAM overhead can be exponential in the number of β-steps of the implemented strategy[3]. Therefore, does not show that the unitary cost model is reasonable. It might however still be possible that the IAM itself is reasonable, that is, that its number of steps is a reasonable time cost model, if the exponential gap with β-steps never materializes on the λ-terms that encode Turing machines. From the point of view of practical efficiency, however, the exponential gap remains problematic. Danos and Regnier (1999) have proposed an optimization called *jumping abstract machine*, which affects both the time *and* the space behavior. An interesting fact is that the JAM is claimed equivalent to the *pointer abstract machine* (PAM), which was defined by Danos et al. (1996) as the computation mechanism behind Hyland and Ong (2000) game semantics. However, no complexity analyses are known about these machines. Muroya and Ghica (2019) mix the token passing mechanism of the IAM with graph rewriting in the style of proof-nets in order to obtain an efficient implementation of different strategies for the λ-calculus.

The Complexity of GoI. While studied at length, no general theory about the complexity of the GoI is known in the literature. Since the IAM is essentially an automaton, analyzing its complexity by executing it can be cumbersome, in particular because its states can be very big. We need a better, *compositional* way of reasoning about the IAM time and space consumption. Since the IAM is deeply rooted in linear logic, the natural direction that we shall follow is the one of non-idempotent intersection types, that have recently been used to give quantitative (time) bounds on several

[3] As an example, the family of terms t_n defined as $t_1 := \mathsf{I}$ and $t_{n+1} := t_n\mathsf{I}$ (where I is the identity combinator) takes time exponential in n to be evaluated by the IAM, but only linear time when evaluated in the λ-calculus (with or without ES), and on any environment machine.

languages and machines. Being a syntactic presentation of the relational model of linear logic, they turn out to be very related, also quantitatively, to how the IAM evaluates λ-terms.

Intersection Types. Intersection types (IT) were introduced by Coppo and Dezani-Ciancaglini (1978) as an extension of simple types able not only to guarantee, but also to characterize termination. Different variants appeared in the literature that were able to characterize different *qualitative* properties, *e.g.* strong, weak or head normalization. They were used to derive (filter) models of the λ-calculus and became a standard tool in programming language theory. A recent survey (Bono and Dezani-Ciancaglini, 2020) gives a complete overview.

More recently, a non-idempotent variant of IT rediscovered by de Carvalho (2018) after having been introduced by Gardner (1994) and later used by Kfoury (2000) and Neergaard and Mairson (2004), has been proved to reflect *quantitative* properties of λ-terms[4], such as the number of β-steps to normal form, or the number of steps of the abstract machine by Krivine (2007) (KAM). The variant requires dropping the idempotency of the intersection operator, therefore considering $A \wedge A$ as *not* equivalent to A, and ultimately making the type system strongly related to the modeling of resources as in linear logic. Such types are sometimes called *multi types*, as intersections become multisets. In the last few years, de Carvalho's results have been dissected and generalized in various ways, adapting the technique to many different calculi and evaluation strategies (Accattoli et al., 2020b; Accattoli and Guerrieri, 2018; Accattoli et al., 2019a; Alves et al., 2019; Bucciarelli et al., 2020; Dal Lago et al., 2021; Kesner and Vial, 2020). In particular, research on the topic received renewed attention after the recent progress in the study of reasonable cost models that made evident that counting β-steps gives rise to a reasonable cost model for time.

Caveat. Being the topics of this thesis so different and broad, and never studied together before, it is not easy to explain them all, in full details. We apologize to the reader for this lack of completeness. This is why we have tried to cite as many surveys as possible, so that the interested reader could use them as a reference.

1.1 Structure of the Thesis and Contributions

We explain in this section the structure of the thesis, listing our main contributions. the manuscript is divided into two parts: the first one devoted to the study of the complexity of the IAM, and the second one mainly dedicated to the study of the KAM, as a space reasonable device. Before starting with original content of the thesis, we have given a technical gentle introduction to abstract machines and intersection types in Chapter 2.

1. Part I is devoted to the Interaction Abstract Machine and to its complexity. We analyze quantitatively the behaviour of the IAM, so as to prove, actually to *disprove*, the conjecture about its space (and eventually time) reasonability. Technically, this is done through a new definition of the machine and thanks to the introduction of two intersection type systems that are able to characterize the IAM space and time consumption.

    - In Chapter 3, we redefine the IAM as a machine acting directly on λ-terms, dubbed λIAM, rather than on linear logic proof nets. This way, a clear inductive structure emerges and this can be used as the basis for a new, conceptually simple, and self-contained, proof of the correctness of the IAM. The change of presentation allows a finer understanding of the IAM complexity and will be the basis for the analyzes carried out in the following chapters. This results appeared in (Accattoli, Dal Lago, and Vanoni, 2020a).
    - In Chapter 4, we develop the time complexity analyzes of the IAM. We exploit non-idempotent intersection types and, in particular, we explain the existing correspondence between the evaluation mechanism at work in the IAM and non-idempotent intersection

[4] Actually, non-idempotent IT are able to reflect *also* the qualitative properties of the idempotent variant. Moreover, the proofs of soundness and completeness are far more easier. Indeed, the *reducibility* technique needed in the idempotent case is replaced by simple inductive and combinatorial arguments. (Bucciarelli et al., 2017) is a recent survey on the subject.

type derivations. We show how to build a machine on top of type derivations and we show that this machine is strongly bisimilar to the IAM. Moreover, assigning weights to types and type judgments, we are able to characterize the IAM time consumption by the way of the weighted type system, in the same way de Carvalho (2007, 2018) did for the KAM. These results appeared in (Accattoli, Dal Lago, and Vanoni, 2021c).

- In Chapter 5, we enrich the previously defined type system in such a way that we are able to capture also the IAM *space* consumption. This modification is non-trivial and was not present in the literature. Capturing the space consumption is particularly interesting as the IAM is supposed to be a space *efficient* evaluation mechanism, with no need for external garbage collection. This feature explains why the problem is hard, indeed at run time the memory used by the IAM sometimes grows and sometimes shrinks. These results appeared in (Accattoli, Dal Lago, and Vanoni, 2021d).
- In Chapter 6, we give, through the use of the type systems mentioned above, a proof of *unreasonability* of the IAM, both in time and space, when Turing machines are encoded in the λ-calculus in the standard way. This is in contrast with the folklore, because of the many results about space complexity that used the IAM as a proof technique. These results appeared in (Accattoli, Dal Lago, and Vanoni, 2021b,d).

2. Part II is mainly devoted to the quest for a reasonable space cost model for the λ-calculus. Since we have shown in Part I that the IAM cannot be a space reasonable cost model, we explore other directions. We start from an interesting (time) optimization of the IAM, very related to Hyland and Ong (2000) game semantics. However, we realize soon that this is not the right tool. Then, we turn our attention to the KAM, but with a perspective which is very different from the usual one. First, we optimize it, so as to remove all possible space overheads (and thus not caring about time efficiency). Then, after having revised the encoding of Turing machines into the λ-calculus, we are able to prove the modified KAM the first space reasonable cost model for the λ-calculus accommodating sub-linear space. Finally, after having transferred this result also to call-by-value evaluation, we are able to characterize compositionally, through an intersection type system, this new reasonable space measure.

   - In Chapter 7, as a first tentative, we redefine the Jumping Abstract Machine and the Pointer Abstract Machine in our framework, this way being able to prove their equivalence (actually, a strong bisimulation). Moreover, we are able to analyze their complexity, thus defining a clear complexity theoretic hierarchy of the abstract machines inspired by the GoI and GS. Unfortunately, these machines do not appear to be good candidates for space reasonable cost models. These results appeared in (Accattoli, Dal Lago, and Vanoni, 2021c).
   - In Chapter 8, we finally turn our attention on the KAM, again after Chapter 2. We define a variant of the KAM, dubbed Space KAM, whose implementation does not rest on the sharing of the environments/closures and that performs eager garbage collection. We prove that the space cost model defined as the space consumption of this machine is *reasonable*, this way solving a long standing open problem in the theory of the λ-calculus[5]. Moreover, we able to transfer our results also to the call-by-value and to give the Space KAM also a reasonable time cost model. These results will appear soon as (Accattoli, Dal Lago, and Vanoni, 2022b).
   - In Chapter 9, we formulate an intersection type system that exactly captures the space consumption of the Space KAM. This is obtained through a small, yet non trivial, modification and adaptation of de Carvalho original type system. This way, a compositional way of measuring reasonable space is given. Also these results will appear soon as (Accattoli, Dal Lago, and Vanoni, 2022a).

[5]The importance of this problem is witnessed by its mention on the Wikipedia page on the λ-calculus: `https://en.wikipedia.org/wiki/Lambda_calculus#Complexity` (accessed on 25 November 2021).

Chapter 2

Types, Machines and the Complexity of the λ-Calculus

In this section we present all the preliminaries related to what will be discussed in the following parts of the dissertation.

2.1 The Closed Call-by-Name λ-Calculus

Church's λ-calculus will be our main object of study throughout this dissertation. We consider it the paradigmatic functional programming language and we want to study its computational complexity, or better the complexity of its implementation. We start by defining the language of terms.

Let $\mathcal{V}$ be a countable set of variables. Terms of the *λ-calculus* Λ are defined as follows:

$$\lambda\text{-TERMS} \qquad t, u, r \quad ::= \quad x \in \mathcal{V} \;\Big|\; \lambda x.t \;\Big|\; tu.$$

Free and *bound variables* are defined as usual: $\lambda x.t$ binds x in t. Terms are considered modulo α-equivalence[1], and $t\{x \leftarrow u\}$ denotes capture-avoiding (meta-level) substitution of all the free occurrences of x for u in t. The operational semantics of the λ-calculus is given just one rewriting rule, called the β-rule:

$$(\lambda x.t)u \mapsto_\beta t\{x \leftarrow u\}$$

Terms like $(\lambda x.t)u$, called reducible expressions, or redexes, can occur as sub-terms inside a bigger term. Restricting the scope of the β-rule to only some of the redexes, *i.e.* defining a *reduction strategy*, gives rise to the different λ-calculi. In this dissertation we deal with the simple possible fragment, which we call Closed Call-by-Name (Closed CbN). It is dubbed *closed*, because we consider only closed terms, *i.e.* terms which do not have any free variable, and *call-by-name*, because arguments are substituted inside function bodies unevaluated. Moreover, Closed CbN is a *weak* strategy, called in fact also weak head reduction, because does not evaluate under abstractions[2], which are, this way, the all and only normal forms, and because it evaluates first the head redex. We can define Closed CbN in the following way:

$$(\lambda x.t)ur_1 \dots r_h \quad \to_{wh} \quad t\{x \leftarrow u\}r_1 \dots r_h.$$

Alternatively, we can use contexts to define weak evaluation in a more "inductive" way, often useful in proofs. Contexts are just λ-terms containing one occurrence of a special symbol, the hole $\langle\cdot\rangle$, which is a placeholder intuitively standing for a removed sub-term. CbN contexts are defined as follows:

$$\text{CbN CONTEXTS} \quad H \quad ::= \quad \langle\cdot\rangle \;\Big|\; Hu$$

[1] Two λ-terms are considered α-equivalent if they are α-convertible. α-conversion is the process of renaming bound variables. Out of the context of the λ-calculus, α-equivalence is usually implicitly assumed, *e.g.* $f(x) = x$ is considered the same as $f(y) = y$.

[2] This is the common practice in functional programming languages.

The operation replacing the hole $\langle\cdot\rangle$ with a term t in a context C is noted $C\langle t\rangle$ and called *plugging*. Then, Closed CbN can be defined as the contextual closure of the β-rule by CbN contexts:

$$\frac{t \mapsto_\beta u}{C\langle t\rangle \to_{wh} C\langle u\rangle}$$

It is already evident from these first definitions that the evaluation of λ-terms according to the definition of Closed CbN reduction is *not* an atomic operation from the complexity-theoretic point of view. Indeed, performing the substitution $t\{x\leftarrow u\}$ has a cost which is at least linear in the number of occurrences of x in t (itself bounded by the size of t) *and* in the size of u. The situation is even worse, because the size of λ-terms can grow exponentially in the length of a reduction sequence, this is the already cited *size explosion*.

This is why β-reduction is never *implemented* as it is *specified*, otherwise it would be impossible to reach the good performances that functional programming languages have. Typically, *abstract machines* are employed to evaluate functional programs, the `OCaml` ZINC (Leroy, 1990), for call-by-value, and the `Haskell` G-machine (Jones, 1992), for call-by-need, being two notable examples.

2.2 The Krivine Abstract Machine

In this section we give the definition and some basic results about an abstract machine for Closed CbN due to Krivine (2007) and dubbed KAM. While giving the correctness results, we have already in mind the complexity analyses, that is, the quantitative properties of the KAM. Since many abstract machines will be introduced next, we first give some definition abstracting from the specific implementation.

Abstract Machines Glossary. An *abstract machine* $\mathrm{M} = (S, \to_\mathrm{M}, \mathsf{init}(\cdot), \cdot\!\downarrow)$ is a transition system $\to$ over a set of states, noted S, together with two functions:

- *Compilation.* $\mathsf{init}(\cdot) : \Lambda \to S$, turning λ-terms into states;
- *Decoding.* $(\cdot)\!\downarrow : S \to \Lambda$, turning states into λ-terms and such that $\mathsf{init}(t)\!\downarrow = t$ for every closed λ-term t.

A state is composed by the *(immutable) code* t_0, the *active term* t, and some data structures. Since the code never changes, it is usually omitted from the state itself, focussing on *dynamic states* that do not mention it. A state $s \in S$ is *initial* for t if $\mathsf{init}(t) = s$. In this dissertation, $\mathsf{init}(t)$ is always defined as the state having t as both the code and the active term and having all the data structures empty. Additionally, the code t shall always be *closed*, without further mention. A state is *final* if no transitions apply. A *run* $\rho : s \to^* s'$ is a possibly empty sequence of transitions, whose length is noted $|\rho|$. If a and b are transitions labels (that is, $\to_a \subseteq \to$ and $\to_b \subseteq \to$) then $\to_{a,b} := \to_a \cup \to_b$, $|\rho|_a$ is the number of a transitions in ρ, and $|\rho|_{\neg a}$ is the size of transitions in ρ that are not $\to_a$. An *initial run* is a run from an initial state $\mathsf{init}(t)$, and it is also called *a run from* t. A state s is *reachable* if it is the target state of an initial run. A *complete run* is an initial run ending on a final state.

Implementation Theorem for Environment Machines. We state the correctness of an abstract machine as an implementation theorem. An environment abstract machine M implements a strategy $\to$ if from the initial state s_t of code t it computes a representation of the normal form $\mathsf{nf}_\to(t)$. In particular, the machine somehow maintains the representation of how the strategy $\to$ modifies the term t they both evaluate. The implementation theorem states a weak bisimulation between the transitions $s \to_\mathrm{M} s'$ of the machine and the steps $t \to u$ of the strategy. In particular, a run ρ_t of the machine on t passes through some states representing u, *i.e.* that can be decoded to u.

The Krivine Abstract Machine, Concretely. The KAM is a standard environment-based machine for Closed CbN, often defined as in Fig. 2.1. The machine evaluates closed λ-terms to weak head normal form via three transitions, the union of which is noted $\to_\mathrm{KAM}$:

CLOSURES	ENVIRONMENTS	STACKS	STATES
$c ::= (t, e)$	$e ::= \epsilon \mid [x{\leftarrow}c] \cdot e$	$\pi ::= \epsilon \mid c \cdot \pi$	$s ::= (t, e, \pi)$

Term	Env	Stack		Term	Env	Stack	
tu	e	π	$\to_{\mathsf{sea}}$	t	e	$(u, e)\cdot\pi$	
$\lambda x.t$	e	$c\cdot\pi$	$\to_{\beta}$	t	$[x{\leftarrow}c]\cdot e$	π	
x	e	π	$\to_{\mathsf{sub}}$	u	e'	π	if $e(x) = (u, e')$

FIGURE 2.1: KAM data structures and transitions.

- $\to_{\mathsf{sea}}$ looks for redexes descending on the left of topmost applications of the active term, accumulating arguments on the stack;
- $\to_{\beta}$ fires a β redex (given by an abstraction as active term having as argument the first entry of the stack) but delays the associated meta-level substitutions, adding a corresponding explicit substitution to the environment;
- $\to_{\mathsf{sub}}$ is a form of micro-step substitution: when the active term is x, the machine looks up the environment and retrieves the delayed replacement for x.

The data structures used by the KAM are *local environments*, *closures*, and a *stack*. *Local environments*, that we shall simply refer to as *environments*, are defined by mutual induction with *closures*. The idea is that every (potentially open) term t in a dynamic state comes with an environment e that *closes* it, thus forming a closure $c = (t, e)$, and, in turn, environments are lists of entries $[x{\leftarrow}c]$ associating to each open variable x of t, a closure c *i.e.*, morally, a closed term. The *stack* simply collects the closures associated to the arguments met during the search for β-redexes.

A dynamic state s of the KAM is the pair (c, π) of a closure c and a stack π, but we rather see it as a triple (t, e, π) by spelling out the two components of the closure $c = (t, e)$. Initial dynamic states of the KAM are defined as $\mathsf{init}(t_0) := (t_0, \epsilon, \epsilon)$ (where t_0 is a closed λ-term, and also the code). The decoding of closures and states is as follows:

$$\begin{array}{rlcl@{\qquad}rlcl} \textsc{Closures} & (t,\epsilon)\downarrow & := & t & & (t,[x{\leftarrow}c]\cdot e)\downarrow & := & (t\{x{\leftarrow}c\downarrow\},e)\downarrow \\ \textsc{States} & (t,e,\epsilon)\downarrow & := & (t,e)\downarrow & & (t,e,\pi\cdot c)\downarrow & := & (t,e,\pi)\downarrow c\downarrow \end{array}$$

Basic Qualitative Properties. Some standard facts about the KAM follow. Let $\rho : \mathsf{init}(t_0) \to_{\mathrm{KAM}}{}^{*} s$ be a run.

- *Closures-are-closed invariant*: if the code t_0 is closed (that is the only case we consider here) then every closure (u, e) in s is *closed*, that is, for any free variable x of u there is an entry $[x{\leftarrow}c]$ in e, and recursively so for the closures in c. Thus $(u, e)\downarrow$is a closed term, whence the name *closures*.
- *Final states*: the previous fact implies that the machine is never stuck on the left of a $\to_{\mathsf{sub}}$ transition because the environment does not contain an entry for the active variable. Final states then have shape $(\lambda x.u, e, \epsilon)$.

Theorem 2.2.1 (Implementation (Accattoli et al., 2014a)). *The KAM implements Closed CbN, that is, there is a complete $\to_{wh}$-sequence $t \to_{wh}^{n} u$ if and only if there is a complete run $\rho : \mathsf{init}(t) \to_{\mathrm{KAM}}{}^{*} s$ such that $s\downarrow = u$ and $|\rho|_{\beta} = n$.*

The key point is that there is a bijection between $\to_{wh}$ steps and $\to_{\beta}$ transitions, so that we can identify the two. Moreover, the number of $\to_{wh}$ steps is a reasonable cost model for time, as first proved by Sands et al. (2002).

Quantitative Properties. We recall also some less known *quantitative* facts, for runs as above, from papers by Accattoli and co-authors (Accattoli et al., 2014a; Accattoli and Barras, 2017; Accattoli and Dal Lago, 2012). The aim is to bound quantities relative to the run ρ and the reachable state s. The bounds are given with respect to two parameters: the size $|t_0|$ of the code and the number $|\rho|_\beta$ of β-transitions, which, as mentioned, is an abstract notion of time for Closed CbN.

- *Number of transitions*: the number $|\rho|_{\mathsf{sub}}$ of sub transitions in ρ is bounded by $\mathcal{O}(|\rho|_\beta^2)$, and there are terms on which the bound is tight. The number $|\rho|_{\mathsf{sea}}$ of sea transitions is bounded by $\mathcal{O}(|\rho|_\beta^2 \cdot |t_0|)$, but on complete runs the bound improves to $\mathcal{O}(|\rho|_\beta)$ and in particular $|\rho|_{\mathsf{sea}} = |\rho|_\beta$.
- *Sub-term invariant*: every term u in every closure (u, e) in every reachable state is a literal (that is, *not* up to α-renaming) sub-term of the code t_0 . Therefore, in particular $|u| \leq |t_0|$.
- *The length of a single environment*: the number of entries in a single environment is bounded by the size $|t_0|$ of the code.
- *The number of environments*: the number of distinct environments in s is bounded only by $|\rho|_\beta$.
- *The length of the stack*: the length of the stack in s is bounded by $\mathcal{O}(|\rho|_\beta^2 \cdot |t_0|)$, but on complete runs the bound improves to $\mathcal{O}(|\rho|_\beta)$.

Sub-Term Pointers and Data Pointers. The KAM is usually implemented using *two* forms of pointers:

1. *Sub-term pointers*: the initial term t_0 provides the initial immutable code. The essential sub-term invariant mentioned above allows us to represent the terms u in every closure (u, e) of reachable states with a pointer to t_0 instead that with a copy of u.
2. *Data (structure) pointers*: to ensure that the duplication of the environment e in transition $\to_{\mathsf{sea}}$ can be implemented efficiently (in time), environments, implemented as linked lists, are shared so that what is duplicated is just a pointer to an environment, and not the environment itself.

Both kinds of pointers shall draw our attention. Sub-term pointers have size $\mathcal{O}(\log |t_0|)$. For the present discussion they are *space-friendly*, because their size does not depend on the length of the run—we shall inspect them in Chapter 8, where we shall ensure that their number is under control. Data pointers, on the other hand, are *space-hostile*, because (as recalled above) the number of environments is bounded only by $|\rho|_\beta$, that is, *time*. Data pointers on complete runs have thus size $\mathcal{O}(\log |\rho|_\beta)$, entangling space with time, which is unreasonable for space.

Cost of KAM Transitions. The idea is that environments are implemented as linked lists, so that the duplication and insertion operations in transitions $\to_{\mathsf{sea}}$ and $\to_\beta$ can be implemented in time proportional to the size of manipulated pointers. Actually, these are the only transitions that contribute to space complexity, creating new pointers. Transition $\to_{\mathsf{sub}}$ needs to access the environment, whose size is bounded by $|t_0|$, the size of the initial term t_0 of the run. By adopting smarter implementations of environments, such as *random access lists* or *balanced trees*, one $\to_{\mathsf{sub}}$ transition costs $\log |t_0|$ plus the cost of manipulating data pointers—see Accattoli and Barras (2017) for discussions about implementations of the KAM. No space is consumed by the $\to_{\mathsf{sub}}$ transition.

The Time and Space Complexity of the KAM. Since on complete runs ρ manipulated pointers are just sub-term pointers, of size $\log |t_0|$, and data pointers, of size $\log |\rho|_\beta$, considering the complexity analyses of the previous paragraphs, we have the following theorem.

Theorem 2.2.2. *Let $\rho : \mathsf{init}(t_0) \to_{\mathrm{KAM}}{}^* s$ be a complete KAM run. It can be implemented on RAMs in time $\mathcal{O}(|\rho|_\beta^2 \cdot \log(|\rho|_\beta \cdot |t_0|))$ and space $\mathcal{O}(|\rho|_\beta \cdot \log(|\rho|_\beta \cdot |t_0|))$.*

$$\frac{}{x : [A] \vdash x : A}\ \text{T-VAR} \qquad \frac{\Gamma, x : M \vdash t : A}{\Gamma \vdash \lambda x.t : M \to A}\ \text{T-}\lambda \qquad \frac{}{\vdash \lambda x.t : \star}\ \text{T-}\lambda_\star$$

$$\frac{\Gamma \vdash t : [B_1, \dots, B_n] \to A \quad [\Delta_i \vdash u : B_i]_{i \in [1,\dots,n]}}{\Gamma \uplus \biguplus_{i \in [1,\dots,n]} \Delta_i \vdash tu : A}\ \text{T-@}$$

FIGURE 2.2: The multi type system for Closed CbN.

As one can immediately notice, the time overhead of the KAM implementation on RAMs, which are a time reasonable model, is polynomial in $|\rho|_\beta$[3]. Since we have identified $|\rho|_\beta$ and the number of $\to_{wh}$-steps, this is a proof that the number of $\to_{wh}$-steps, which is the natural time cost model for the Closed CbN λ-calculus, *is* a reasonable time cost model[4].

At the same time, the theorem above, tell us that the space consumed by the KAM is an *unreasonable* cost model. In fact, it depends quasi-linearly from time. This is a problem when one has to implement, say, TMs into the λ-calculus. If a TM halts in n steps, then a linear-time simulation of TMs into the λ-calculus would require space, when executed via on the KAM, quasi-linear in n. This would mean that space on the λ-calculus is more than linear in *time* of TMs. For the invariance thesis to hold, one requires instead that space in the λ-calculus is linear in the *space* of TMs.

Related Work on The Complexity of Environment Machines. The *time* efficiency of environment machines has been recently closely scrutinized. Before 2014, the topic had been mostly neglected—the only two counterexamples being Blelloch and Greiner (1995) and Sands et al. (2002). Since 2014—motivated by advances by Accattoli and Dal Lago on time cost models for the λ-calculus Accattoli and Dal Lago, 2016—Accattoli and co-authors have explored time analyses of environment machines from different angles (Accattoli et al., 2014a; Accattoli and Barras, 2017; Accattoli et al., 2021a; Accattoli and Guerrieri, 2019; Accattoli et al., 2019b). Recently, also Biernacka et al. (2021) gave similar results. A (slightly outdated) survey on the topic by Accattoli (2016) gives a gentle introduction to the topic.

2.3 Non-Idempotent Intersection Types

We give in this section an introduction to non-idempotent intersection types (a.k.a. multi types). First, we consider them from the *qualitative* point of view, and the we refine our analysis *quantitatively*. We will review them quite quickly, inviting the reader to consult (Bucciarelli et al., 2017) about the qualitative part, and (Accattoli et al., 2020b) about the quantitative one. Multi types are very related to linear logic in how they track the use of resources, and in particular their grammar is reminiscent of the (call-by-name, in our case) translation $\cdot^\dagger$ of intuitionistic logic into linear logic $(A \to B)^\dagger = !A^\dagger \multimap B^\dagger$. Semantically, they can be seen as a syntactical presentation of the relational model (Bucciarelli and Ehrhard, 2001) of the λ-calculus, when the latter is interpreted into linear logic.

$$\begin{array}{rrcll} \text{LINEAR TYPES} & A, B & ::= & \star \mid M \to A & \\ \text{MULTI TYPES} & M, N & ::= & [A_1, \dots, A_n] & n \geq 0 \end{array}$$

Note that there is a ground type $\star$, which can be thought as the type of normal forms, that in Closed CbN are precisely abstractions. Note also that arrow (linear) types $M \to A$ can have a multiset only on the left. The empty multiset is noted $[\cdot]$, and the union of two multisets M and N is noted $M \uplus N$.

[3] It is possible to make the overhead quasi-linear instead of quasi-quadratic by adopting an optimization called *unchaining*. This shall be presented in Chapter 8, since it is related also to space consumption

[4] Actually, to prove that a cost model is reasonable one has also to show the inverse direction, *i.e.* that there is a polynomially bounded simulation of a reasonable model into the Closed CbN λ-calculus. This can be proved by simulating TMs, *e.g.* as it is done in Appendix D

$$\frac{}{\vdash^{1}}\ \text{T-VAR} \qquad \frac{\vdash^{w}}{\vdash^{w+1}}\ \text{T-}\lambda \qquad \frac{}{\vdash^{0}}\ \text{T-}\lambda_{\star} \qquad \frac{\vdash^{w} \quad [\vdash^{v_i}]_{i\in[1,\dots,n]}}{\vdash^{w+\sum v_i+1}}\ \text{T-@}$$

FIGURE 2.3: The weight assignments $\mathbf{W}_{KAM}(\cdot)$.

Type judgments have the form $\Gamma \vdash t : A$, where Γ is a type environment, defined below. The typing rules are in Fig. 2.2, type derivations are noted π and we write $\pi \triangleright \Gamma \vdash t : A$ for a type derivation π of ending judgment $\Gamma \vdash t : A$. Type environments, ranged over by Γ, Δ are total maps from variables to multi types such that only finitely many variables are mapped to non-empty multi types, and we write $\Gamma = x_1 : M_1, \dots, x_n : M_n$ if $dom(\Gamma) = \{x_1, \dots, x_n\}$. Given two type environments Γ, Δ, the expression $\Gamma \uplus \Delta$ stands for the type environment assigning to every variable x the multiset $\Gamma(x) \uplus \Delta(x)$.

Characterization of Termination. Multi types characterize Closed CbN termination, that is, they type all and only those λ-terms that terminate with respect to Closed CbN. Differently from what happens with the idempotent version, proving this result is surprisingly easy. Indeed, one is required to prove in quantitative way the substitution lemma plus subject reduction for correctness, and anti-substitution lemma, subject expansion, and typability of all normal forms for completeness (here trivial, because all normal forms are typed by T-$\lambda_{\star}$). All the proofs can be carried out by a simple induction on the structure of terms/types. In Appendix C, the reader can find the proof for a little variant over this multi type system.

Theorem 2.3.1. *A closed term t has weak head normal form if and only if $\vdash t : \star$.*

Multi Types and the KAM Time Consumption. Multi types have been successfully applied in quantitative analyses of normalization, starting with de Carvalho (2007, 2018) who used them to give a bound to the length of KAM runs. de Carvalho's technique can be re-phrased and distilled as a decoration of type derivations with *weights*, that is, cost annotations, following the scheme of Fig. 2.3. Please note that the weight assignment is blind to types, and thus relies only on the structure of the type derivation. de Carvalho's result can be formulated as follows.

Theorem 2.3.2 (de Carvalho)**.** *There is a complete KAM run of length n from t if and only if exists π such that $\pi \triangleright \vdash t : \star$ and $\mathbf{W}_{KAM}(\pi) = n$.*

The KAM being deterministic, one has that all derivations $\vdash t : \star$ induce the same weights. Moreover, there is a stronger correspondence between the rules of the type system, and the transitions of the KAM. Every T-@ rule corresponds to a $\to_{\mathsf{sea}}$ transition, every T-λ rule corresponds to a $\to_{\beta}$ transition[5], and every T-VAR rule corresponds to a $\to_{\mathsf{sub}}$ transition. The only T-$\lambda_{\star}$ rule in a type derivation for t corresponds to the final state of the KAM run on t. The correspondence is deep, each state of the KAM run on a λ-term t is linked to a judgment in the type derivation of t in such a way that the sub-term of the state is exactly the same sub-term of the judgment. In other words, the KAM and multi types *compute* in the *same* way, they are only different syntactical styles. Moreover, this is the same mechanism Accattoli and Kesner's Linear Substitution Calculus (Accattoli and Dal Lago, 2012; Accattoli et al., 2014b), which will be introduced in the next chapter, uses to evaluate λ-terms.

Related Works on Multi Types. Various works in the literature explore multi types from various points of view. Several papers study multi types qualitatively, *e.g.* in call-by-name (Bucciarelli et al., 2018, 2021; Kesner and Vial, 2017; Olimpieri, 2021; Tsukada and Ong, 2016) and in call-by-value (Carraro and Guerrieri, 2014; Ehrhard, 2012). Quantitatively, many works by Accattoli, Kesner and coauthors have already been cited, *e.g.* (Accattoli and Guerrieri, 2018; Accattoli et al., 2019a; Alves et al., 2019; Bucciarelli et al., 2020; Kesner and Vial, 2020). Dal Lago et al. (2021) used multi types to give a quantitative characterization of the probability of convergence of *probabilistic* λ-terms.

[5]This way, multi type derivations can be used to count also the number of $\to_{wh}$-steps a λ-term needs to normalize.

Scoping. In this thesis we shall define several weighted type systems in the same style of the one presented in this chapter. In order to not to make the notation too heavy, we do not have given different symbols to the different systems. We assume a scoping by-chapter. Indeed, no type system is used outside the chapter in which it is defined. The only exception being Chapter 6, where the two sections are clearly using two different type systems.

Part I

The IAM: Types and Complexity

Chapter 3

The Interaction Abstract Machine

The original GoI machine is the *Interaction Abstract Machine* (IAM). It was developed at the same time by Mackie and Danos & Regnier, and its first appearance is in a paper by Mackie (1995), dealing with implementations. Danos and Regnier study it in two papers, one in 1996 together with Herbelin (Danos et al., 1996), where it is dealt with quickly, and its implementation theorem (or correctness[1]) is proved via game semantics, and one by themselves (Danos and Regnier, 1999), published only in 1999 but reporting work dating back to a few years before, dedicated to the IAM and to an optimization based on a fine analysis of IAM runs. These papers differ on many details but they all formulate the IAM on linear logic proof-nets as a reversible, bideterministic automaton. Danos, Herbelin, and Regnier (1996) prove that the IAM implements *linear head evaluation* $\to_{\mathsf{lh}}$ (shortened to LHE), a refinement of head evaluation, arising from the linear logic decomposition of the λ-calculus.

New Proof of the Implementation Theorem. We present an alternative proof of the implementation theorem for the IAM, which is independent of game semantics and other abstract machines. Our proof is direct and based on a natural notion of bisimulation, namely a variant on Sands' *improvements* (Sands, 1996).

The implementation theorem of GoI machines amounts to showing that their result is an adequate and sound semantics for LHE, that is, it is invariant by LHE (soundness) and it is non-empty if and only if LHE terminates (adequacy). The key point for soundness is that—in contrast to the study of environment machines—steps of the GoI machine are not mapped to LHE steps, because the GoI computes differently. What is shown is that if $t \to_{\mathsf{lh}} u$ then the run of the machine on t is "akin" to the run on u, and they produce akin results—see Sect. 3.6 for more details. In our proof, "akin" is interpreted as *bisimilar*. An *improvement* is a bisimulation asking that the run on u is no longer than the run on t. Building on such a quantitative refinement, we prove adequacy. The proof of our implementation theorem is arguably conceptually simpler than Danos, Herbelin, and Regnier's. Of course, their deep connection with game semantics is an important contribution that is not present here.

The Lambda Interaction Abstract Machine. The second contribution of this chapter is a formulation of the IAM as a machine acting directly on λ-terms rather than on linear logic proof nets. Our proof might also have been carried out on proof nets, but we prefer switching to λ-terms for two reasons. First, manipulating terms rather than proof nets is easier and less error-prone for the technical development. Second, we aim at minimizing the background required for understanding the IAM, and so doing we remove any explicit reference to linear logic and graphical syntaxes.

The starting point of our *Lambda* Interaction Abstract Machine (λIAM) is seeing a position in the code t (what is usually the position of the token on the proof net representation of t) as a pair (u, C) of a sub-term u and a context C such that $C\langle u\rangle = t$. These positions are simply a readable presentation of pointers[2].

[1] The result that an abstract machine implements a strategy is sometimes called *correctness* of the machine. We prefer to avoid such a terminology, because it suggests the existence of a dual *completeness* result, that is never given because already contained in the statement of correctness. We then simply talk of an *implementation theorem*.

[2] For the acquainted reader, they play a role akin to the initial labels in Lévy's labeled λ-calculus, itself having deep connections with the IAM (Asperti et al., 1994).

The main novelty of the new presentation is that some of the exponential transitions on proof nets are packed together in macro transitions. The shape of our transitions makes a sort of backtracking mechanism more evident. *Careful*: that the IAM rests on backtracking is the key point of Danos and Regnier (1999), and therefore it is not a novelty in itself. What is new is that such a mechanism is already visible at the level of transitions, while on proof nets it requires sophisticated analyses of runs.

More About the λIAM. The original papers on the IAM (Danos et al., 1996; Danos and Regnier, 1999; Mackie, 1995) differ on many points. Here we follow (Danos et al., 1996), modeling the λIAM on the call-by-name translation of the λ-calculus in linear logic and considering only the path/runs starting on the distinguished conclusion corresponding to the output of the net/term. This is natural for terms, and also along the lines of how AJM games interpret terms. Similarly to AJM games, then, our GoI semantics is sound also for open terms with respect to erasing steps.

An original point of our work is the identification of a new invariant of the λIAM—probably of independent interest—based on what we call *exhaustible states*. Informally, a state of the λIAM is exhaustible if its token can be emptied in a certain way, somehow mimicking the computation which leads to the state itself.

Related Work on GoI. The literature about the GoI and its applications is huge, and goes from the original paper by Girard (1989b), to the reformulation by (Abramsky et al., 2002) using the INT-construction, Danos and Regnier (1993) using path algebras, the application by Ghica (2007) to circuit synthesis, together with extensions by Hoshino et al. (2014) to languages with various kinds of effects, and Laurent (2001) extension to the additive connectives of linear logic. In all these cases, the GoI interpretation, even when given on λ-terms, goes *through* linear logic (or symmetric monoidal categories) in an essential way. The only notable exceptions are perhaps the recent contributions by Schöpp (2014, 2015) on the relations between GoI, CPS, and defunctionalization in which, indeed, some deep relations are shown to exist between GoI and classic tools in the theory of λ-calculus[3]. Even there, however, GoI is seen as obtained through the INT-construction (Abramsky et al., 2002; Joyal et al., 1996), although applied to a syntactic category of terms.

The GoI has been studied in relationship with implementations of functional languages, by Gonthier et al. (1992), who studied Lévy's optimal evaluation, and by Mackie (1995, 2017) with his GoI machine for PCF, Gödel System T, and—with Fernandez—for call-by-value (Fernández and Mackie, 2002). The space-efficiency studied by Dal Lago and Schöpp (2016) is exploited by Mazza (2015) and by Mazza and Terui (2015). Dal Lago et al. (2017a, 2014, 2015, 2017b) have also introduced variants of the IAM acting on proof nets for a number of extensions of the λ-calculus. Curien and Herbelin (1998, 2007) study abstract machines related to game semantics and the IAM. Muroya and Ghica (2017) have recently studied the GoI in combination with rewriting and abstract machines.

3.1 A Gentle Introduction to the λIAM

This section introduces the λIAM gradually. There are various mechanisms at work in the λIAM. Most of them are also part of the simpler machine that evaluates linear λ-terms (where each variable occurs exactly once). Then we first see the Linear λIAM , which is easier to grasp, and later refine it with non-linearity.

Defining the Linear λIAM . An essential point is that the initial code t of the machine never changes. The λIAM only moves over it, in a local way, with no rewriting of the code and without ever substituting terms for variables. The current position in the code t is represented as a pair (u, C) where C is a context (that is, a term with a hole) and $C\langle u\rangle = t$. A state s of the Linear λIAM

[3] In particular, Schöpp highlights how GoI can be seen as an optimized form of CPS transformation, followed by defunctionalization.

Sub-tm	Context	Token		Sub-tm	Context	Token
$\underline{tu}$	C	T	$\to_{@1}$	$\underline{t}$	$C\langle\langle\cdot\rangle u\rangle$	$@\cdot T$
$\underline{\lambda x.t}$	C	$@\cdot T$	$\to_{@2}$	$\underline{t}$	$C\langle\lambda x.\langle\cdot\rangle\rangle$	T
$\underline{x}$	$C\langle\lambda x.D\rangle$	T	$\to_{\mathsf{var}}$	$\lambda x.D\langle x\rangle$	$\underline{C}$	$x\cdot T$
$\underline{\lambda x.D\langle x\rangle}$	C	$\mathsf{b}\cdot T$	$\to_{\mathsf{bt2}}$	x	$\underline{C\langle\lambda x.D\rangle}$	T
t	$\underline{C\langle\langle\cdot\rangle u\rangle}$	$\lambda\cdot T$	$\to_{\lambda 2}$	tu	$\underline{C}$	T
t	$\underline{C\langle\lambda x.\langle\cdot\rangle\rangle}$	T	$\to_{\lambda 1}$	$\lambda x.t$	$\underline{C}$	$\lambda\cdot T$
t	$\underline{C\langle\langle\cdot\rangle u\rangle}$	$x\cdot T$	$\to_{\mathsf{arg}}$	$\underline{u}$	$C\langle t\langle\cdot\rangle\rangle$	T
t	$\underline{C\langle u\langle\cdot\rangle\rangle}$	T	$\to_{\mathsf{bt1}}$	$\underline{u}$	$C\langle\langle\cdot\rangle t\rangle$	$\mathsf{b}\cdot T$

FIGURE 3.1: Linear λIAM transitions.

has the shape (t, C, T, d) where (t, C) is a position, while T and d are the linear token (which is a stack) and the direction, defined by:

$$\begin{array}{rrcl} \text{(LINEAR) TOKEN} & T & ::= & \epsilon \mid @\cdot T \mid \lambda\cdot T \mid x\cdot T \mid \mathsf{b}\cdot T \\ \text{DIRECTION} & d & ::= & \downarrow \mid \uparrow \end{array}$$

The token T can contain occurrences of @ and λ, variables, and occurrences of b. The use of these symbols is explained below via examples. Directions $\downarrow$ and $\uparrow$, pronounced *down* and *up*, shall be represented mostly via colors and underlining: the code term in red and underlined, for $\downarrow$, and the code context in blue and underlined, for $\uparrow$. This way, the fourth component of states is often omitted.

The transitions of the Linear λIAM are in Fig. 3.1. Roughly, when the direction is $\downarrow$, the machine looks for the head variable of t. The direction changes to $\uparrow$ when the head variable x is found, moving also the machine to the position $(\lambda x.u, D)$ of the binder of x. In direction $\uparrow$, the λIAM explores D looking for the argument that head evaluation would substitute on x. *Initial states* have the form $s_t := (\underline{t}, \langle\cdot\rangle, \epsilon)$, where t is a term.

Mechanism 1: Search Up to β-Redexes. A basic mechanism of the Linear λIAM is the search of the head variable in direction $\downarrow$ without recording β-redexes, via transitions $\to_{@1}$ and $\to_{@2}$. Consider the following run, with $\mathsf{I}_z := \lambda z.z$ and $\mathsf{I}_w := \lambda w.w$.

Sub-tm	Context	Tok.	
$\underline{((\lambda y.\lambda x.xy)\mathsf{I}_z)\mathsf{I}_w}$	$\langle\cdot\rangle$	ϵ	$\to_{@1}$
$\underline{(\lambda y.\lambda x.xy)\mathsf{I}_z}$	$\langle\cdot\rangle\mathsf{I}_w$	$@$	$\to_{@1}$
$\underline{\lambda y.\lambda x.xy}$	$(\langle\cdot\rangle\mathsf{I}_z)\mathsf{I}_w$	$@\cdot@$	$\to_{@2}$
$\underline{\lambda x.xy}$	$((\lambda y.\langle\cdot\rangle)\mathsf{I}_z)\mathsf{I}_w$	$@$	$\to_{@2}$
$\underline{xy}$	$((\lambda y.\lambda x.\langle\cdot\rangle)\mathsf{I}_z)\mathsf{I}_w$	ϵ	$\to_{@1}$
$\underline{x}$	$((\lambda y.\lambda x.\langle\cdot\rangle y)\mathsf{I}_z)\mathsf{I}_w$	$@$	

When the machine faces an application, transition $\to_{@1}$ records on the token the presence of an argument by pushing the symbol @, but not the argument itself (as it would instead do an environment machine), and moves to the left sub-term. Dually, on an abstraction, if the top of the token is @ then the machine pops it and moves to the body of the abstraction. This way, β-redexes are simply skipped. Note indeed that after the first 4 transitions the machine has crossed 2 β-redexes and the token is empty, as at the beginning.

Mechanism 2: Finding Variables and Arguments. In the example, the Linear λIAM finds the head variable x. Then transition $\to_{\mathsf{var}}$ applies, changing the direction to $\uparrow$ and moving the position to the binder λx. The machine now looks for the argument of the binder, exploring the context (now underlined and in blue) rather than the sub-term of the current position.

$$\begin{array}{c|c|cc} \underline{x} & ((\lambda y.\lambda x.\langle\cdot\rangle y)\mathsf{I}_z)\mathsf{I}_w & @ & \to_{\mathsf{var}} \\ \lambda x.xy & \underline{((\lambda y.\langle\cdot\rangle)\mathsf{I}_z)\mathsf{I}_w} & x\cdot@ & \to_{\lambda 1} \\ \lambda y.\lambda x.xy & \underline{(\langle\cdot\rangle\mathsf{I}_z)\mathsf{I}_w} & \lambda\cdot x\cdot@ & \to_{\lambda 2} \\ (\lambda y.\lambda x.xy)\mathsf{I}_z & \underline{\langle\cdot\rangle\mathsf{I}_w} & x\cdot@ & \to_{\mathsf{arg}} \\ \underline{\mathsf{I}_w} & ((\lambda y.\lambda x.xy)\mathsf{I}_z)\langle\cdot\rangle & @ & \end{array}$$

Note that $\to_{\mathsf{var}}$ adds x to the token, recording that the variable x has been found. The search for the argument is, again, *up to* β*-redexes*, via transitions $\to_{\lambda 1}$ and $\to_{\lambda 2}$, adding a symbol λ to the token on abstractions and removing it on applications. The $\uparrow$ phase ends with transition $\to_{\mathsf{arg}}$, that fires when the hole of the context is facing an argument and the token contains a variable, here x—such an argument matches the binder λx of the previously found variable. The transition removes x from the token, moves to the found argument, and switches direction.

Mechanism 3: Backtracking. On the example, the Linear λIAM continues by looking for the head variable of I_w, as expected.

$$\begin{array}{c|c|cc} \underline{\lambda w.w} & ((\lambda y.\lambda x.xy)\mathsf{I}_z)\langle\cdot\rangle & @ & \to_{@1} \\ \underline{w} & ((\lambda y.\lambda x.xy)\mathsf{I}_z)(\lambda w.\langle\cdot\rangle) & \epsilon & \to_{\mathsf{var}} \\ \lambda w.w & \underline{((\lambda y.\lambda x.xy)\mathsf{I}_z)\langle\cdot\rangle} & w & \end{array}$$

Now something different happens. The machine is looking for the argument of I_w. Such an argument is not readily available, as the hole has no arguments in the current context, because $\langle\cdot\rangle$ is the right sub-term of an application. Since I_w is a replacement for x, its argument is actually the argument y of x. Then the machine *backtracks* to x. Backtracking is started by transition $\to_{\mathsf{bt1}}$, which adds b to the token, moves to the left sub-term of the application, and changes direction. The next 3 steps look for λx up to β-redexes:

$$\begin{array}{c|c|cc} \mathsf{I}_w & \underline{((\lambda y.\lambda x.xy)\mathsf{I}_z)\langle\cdot\rangle} & w & \to_{\mathsf{bt1}} \\ \underline{(\lambda y.\lambda x.xy)\mathsf{I}_z} & \langle\cdot\rangle\mathsf{I}_w & \mathsf{b}\cdot w & \to_{@1} \\ \underline{(\lambda y.\lambda x.xy)} & (\langle\cdot\rangle\mathsf{I}_z)\mathsf{I}_w & @\cdot\mathsf{b}\cdot w & \to_{@2} \\ \underline{\lambda x.xy} & ((\lambda y.\langle\cdot\rangle)\mathsf{I}_z)\mathsf{I}_w & \mathsf{b}\cdot w & \to_{\mathsf{bt2}} \\ x & \underline{((\lambda y.\lambda x.\langle\cdot\rangle y)\mathsf{I}_z)\mathsf{I}_w} & w & \to_{\mathsf{arg}} \\ \underline{y} & ((\lambda y.\lambda x.x\langle\cdot\rangle)\mathsf{I}_z)\mathsf{I}_w & \epsilon & \end{array}$$

When λx is found, transition $\to_{\mathsf{bt2}}$ ends the backtracking, going back to the unique occurrence of x in the body of the abstraction (here linearity is crucial), and restores the search for arguments, changing direction and removing b. The next step is given by transition $\to_{\mathsf{arg}}$, that finally finds the argument y of I_w, removing w from the token. In two more transitions, the Linear λIAM reaches a final state, signifying that t is normalizing when evaluated under Closed CbN.

$$\begin{array}{c|c|cc} \underline{y} & ((\lambda y.\lambda x.x\langle\cdot\rangle)\mathsf{I}_z)\mathsf{I}_w & \epsilon & \to_{\mathsf{var}} \\ (\lambda y.\lambda x.xy) & \underline{(\langle\cdot\rangle\mathsf{I}_z)\mathsf{I}_w} & y & \to_{\mathsf{arg}} \\ \underline{\mathsf{I}_z} & ((\lambda y.\lambda x.xy)\langle\cdot\rangle)\mathsf{I}_w & \epsilon & \end{array}$$

Two Observations: @ and λ Can Be Merged, and Locality. Note that the symbols @ and λ of the token play dual roles and never interact. In the following sections, we then harmlessly merge them into a single symbol $\bullet$, pronounced *dot*. Moreover, the reader can observe that transitions $\to_{\mathsf{var}}$ and $\to_{\mathsf{bt2}}$ move from a variable to its binder and vice versa. These moves are local if one assumes that λ-terms are represented by implementing variable occurrences as pointers to their binders, as in the proof net representation of λ-terms, see Sect. 3.8 for a precise comparison.

Two Issues with Non-Linearity. Generalizing the Linear λIAM to arbitrary, potentially non-linear λ-terms needs to address two difficulties. The first one is that, when the machine faces a sub-term $\lambda x.t$ and needs to end backtracking via transition $\to_{\mathsf{bt2}}$, it is no longer clear to which occurrence of x in t one should move, if there are many, or what to do, if x does not occur in t. The second issue is related to duplication. Consider the term $t := (\lambda x.x(xu))(\lambda y.r)$. The head

Sub-term	Context	Log	Tape		Sub-term	Context	Log	Tape
$\underline{tu}$	C	L	T	$\to_{\bullet 1}$	$\underline{t}$	$C\langle\langle\cdot\rangle u\rangle$	L	$\bullet \cdot T$
$\underline{\lambda x.t}$	C	L	$\bullet \cdot T$	$\to_{\bullet 2}$	$\underline{t}$	$C\langle\lambda x.\langle\cdot\rangle\rangle$	L	T
$\underline{x}$	$C\langle\lambda x.D_n\rangle$	$L_n \cdot L$	T	$\to_{\mathsf{var}}$	$\lambda x.D_n\langle x\rangle$	$\underline{C}$	L	$l_n \cdot T$
$\underline{\lambda x.D_n\langle x\rangle}$	C	L	$l_n \cdot T$	$\to_{\mathsf{bt2}}$	x	$\underline{C\langle\lambda x.D_n\rangle}$	$L_n \cdot L$	T
where $l_n := (x, \lambda x.D_n, L_n)$.								
t	$\underline{C\langle\langle\cdot\rangle u\rangle}$	L	$\bullet \cdot T$	$\to_{\bullet 3}$	tu	$\underline{C}$	L	T
t	$\underline{C\langle\lambda x.\langle\cdot\rangle\rangle}$	L	T	$\to_{\bullet 4}$	$\lambda x.t$	$\underline{C}$	L	$\bullet \cdot T$
t	$\underline{C\langle\langle\cdot\rangle u\rangle}$	L	$l \cdot T$	$\to_{\mathsf{arg}}$	$\underline{u}$	$C\langle t\langle\cdot\rangle\rangle$	$l \cdot L$	T
t	$\underline{C\langle u\langle\cdot\rangle\rangle}$	$l \cdot L$	T	$\to_{\mathsf{bt1}}$	$\underline{u}$	$C\langle\langle\cdot\rangle t\rangle$	L	$l \cdot T$

FIGURE 3.2: λIAM transitions.

evaluation of t makes two copies of $\lambda y.r$ that are used differently, because one is applied to u and the other one to $(\lambda y.r)u$. The machine does not duplicate sub-terms but still it has to somehow distinguish different uses of a sub-term.

Towards the λIAM. The two issues are solved via three correlated modifications of the machine. First, the symbols x and b for the token are generalized to variable positions $(x, \lambda x.C)$ inside the scope of their binder. Replacing b with $(x, \lambda x.C)$ in particular removes the non-determinism on $\to_{\mathsf{bt2}}$, when x has many occurrences[4]. Second, the token is split into two components, called *log* and *tape*. Roughly, the tape is the token of the Linear λIAM (generalized to variable positions) while the log stores, after transition $\to_{\mathsf{arg}}$, the variable position for which the machine found the argument, so as to be able to know to which occurrence to backtrack to in transition $\to_{\mathsf{bt1}}$. Third, there is a mechanism for distinguishing different uses of sub-terms. The log actually stores more than one variable position, and every position comes with its own log, acting as a sort of identifier for the use of that position. The next section formally develops this subtle mechanism. Logs and the way they distinguish uses of sub-terms without duplicating are far from being intuitive: they are both the mystery and the magic of the geometry of interaction.

3.2 The λIAM

In this section we introduce the data structures used by the λIAM and its transition rules.

Leveled Contexts. The study of the λIAM requires *contexts*, that are terms with a single occurrence of a special constant $\langle\cdot\rangle$, called *the hole*, that is a place-holder for a removed sub-term. In fact, we need a notion of context more informative than the usual one, introduced next.

LEVELED CONTEXTS
$$\begin{array}{rll} C_0 & ::= \langle\cdot\rangle \mid \lambda x.C_0 \mid C_0 t; \\ C_{n+1} & ::= \lambda x.C_{n+1} \mid C_{n+1} t \mid t C_n. \end{array}$$

The index n in C_n counts the number of arguments into which the hole $\langle\cdot\rangle$ is contained[5]. The level of a context shall be omitted when not relevant to the discussion—note that any ordinary context can be written *in a unique way* as a leveled context, so that the omission is anyway harmless. The *plugging* $C_n\langle t\rangle$ of a term t in C_n is defined by replacing the hole $\langle\cdot\rangle$ with t, potentially capturing free variables of t. Plugging $C_n\langle C_m\rangle$ of a context for a context is defined similarly. A *position* (of level n) in a term u is a pair (t, C_n) such that $C_n\langle t\rangle = u$.

[4] An invariant shall guarantee that one never backtracks inside an abstraction whose variable has no occurrences.

[5] Such an index has a natural interpretation in linear logic terms. According to the standard (call-by-name) translation of the λ-calculus into linear logic proof nets, in a context C_n the hole lies inside exactly n !-boxes.

Logs and Logged Positions. The λIAM relies on two mutually recursive notions, namely *logged positions* and *logs*: a logged position is a position (t, C_n) together with a log[6] L_n, that is a list of logged positions, having length n.

$$\begin{array}{ccc} \text{LOGGED POSITIONS} & & \text{LOGS} \\ l ::= (t, C_n, L_n) & L_0 ::= \epsilon & L_{n+1} ::= l \cdot L_n \end{array}$$

We use $\cdot$ also to concatenate logs, writing, *e.g.*, $L_n \cdot L$, using L for a log of unspecified length. Intuitively, logs contain some minimal information for backtracking to the associated position.

Tape, Token, Direction, State. The *tape* T is a finite sequence of elements of two kinds, namely logged positions, and occurrences of the special symbol $\bullet$, needed to cross abstractions and applications.

$$\text{TAPES} \qquad T \quad ::= \quad \epsilon \;\Big|\; \bullet \cdot T \;\Big|\; l \cdot T.$$

A *token* is a log plus a tape.

Definition 3.2.1 (λIAM State). *A state s of the λIAM is a quintuple (t, C, L, T, d) where t is a λ-term, called the* code term*, C is a context, called the* code context*, L is a log, T is a tape, and d is an element of $\{\uparrow, \downarrow\}$, called the* direction*.*

As for the Linear λIAM , directions shall be represented mostly via colors and underlinings, omitting the fifth component.

Initial States. The λIAM starts on *initial states* of the form $s_t := (t, \langle\cdot\rangle, \epsilon, \epsilon, \downarrow)$, where t is a λ-term[7], and ϵ is the empty log/tape.

Transitions. The transitions of the λIAM are in Fig. 3.2. Their union is noted $\to_{\lambda\text{IAM}}$. A state s is *reachable* if $s_t \to^*_{\lambda\text{IAM}} s$ for an initial state s_t and it is *final* if there exists no s' such that $s \to_{\lambda\text{IAM}} s'$. The shape of final states is characterized in Sect. 3.3.

As for the Linear λIAM , $\downarrow$-states $(\underline{t}, C, L, T)$ are queries about the head variable of (the head normal form of) t and $\uparrow$-states $(t, \underline{C}, L, T)$ are queries about the argument of an abstraction. With respect to the Linear λIAM , transitions $\to_{@1}, \to_{@2}, \to_{\lambda 1}$, and $\to_{\lambda 2}$ are respectively renamed $\to_{\bullet 1}, \to_{\bullet 2}, \to_{\bullet 4}$, and $\to_{\bullet 3}$, because $\bullet$ subsumes both token symbols @ and λ. The role of both symbols x and b is instead played by logged positions. Note that transition $\to_{\mathsf{arg}}$ moves the logged position from the tape to the log, and that transition $\to_{\mathsf{bt1}}$ moves it back to the tape, as it shall specify, at the end of the backtracking, to which variable occurrence $\to_{\mathsf{bt2}}$ has to move. The less intuitive aspect of the λIAM is the splitting of the log in transition $\to_{\mathsf{var}}$ and the dual concatenation of logs in $\to_{\mathsf{bt2}}$. To justify it, note first that the log is extended every time the machine enters into an argument via $\to_{\mathsf{arg}}$, which is also when the level of the position increases of 1. Note also that transitions $\to_{\mathsf{bt1}}$ moves out of an argument (decreasing the level of 1) and removes an entry from the log. To preserve the invariant that the log length is exactly the depth of the context, transition $\to_{\mathsf{var}}$ splits the log between the two positions $(\lambda x.D_n\langle x\rangle, C)$ and $(x, \lambda x.D_n)$, according to the depth of their contexts, and $\to_{\mathsf{bt2}}$ merges them back.

3.3 Properties of the λIAM

Here we first discuss a few invariants of the data structures of the machine, and then we analyze final states and the semantic interpretation defined by the λIAM.

[6] In computer science logs are traces that can only grow, while here they also shrink. The terminology suggests a tracing mechanism—*trace* is avoided because related to categorical formulations of the GoI.

[7] To be precise, one needs a *well-named* term, that is, one in which all bound variables have distinct names, also distinct wrt free names. Since the code is immutable, this detail is only needed to assure that the relationship between variables and binders is unambiguous, for the definition of the transitions.

The Code Invariant. An inspection of the rules shows that, along a computation, the machine travels on a λ-term without altering it.

Proposition 3.3.1 (Code Invariant). *If* $(t, C, L, T, d) \to_{\lambda\text{IAM}} (u, D, L', T', d')$, *then* $C\langle t\rangle = D\langle u\rangle$.

The Balance Invariant. Given a state (t, C, L, T, d), the log and the tape, *i.e.* the token, verify two easy invariants connecting them to the position (t, C) and the direction d. The log L, together with the position (t, C), form a logged position, *i.e.* the length of L is exactly the level of the code context C. This fact guarantees that the λIAM never gets stuck because the log is not long enough for transitions $\to_{\mathsf{var}}$ and $\to_{\mathsf{bt1}}$ to apply.

About the tape, note that every time the machine switches from a $\downarrow$-state to an $\uparrow$-state (or vice versa), a logged position is pushed (or popped) from the tape T. Thus, for reachable states, the number of logged positions in T gives the direction of the state. These intuitions are formalized by the balance invariant below. Given a direction d we use d^n for the direction obtained by switching d exactly n times (i.e., $\downarrow^0 = \downarrow$, $\uparrow^0 = \uparrow$, $\downarrow^{n+1} = \uparrow^n$ and $\uparrow^{n+1} = \downarrow^n$).

Lemma 3.3.2 (Balance Invariant). *Let* $s = (t, C_n, L, T, d)$ *be a reachable state and* $|T|_l$ *the number of logged positions in* T. *Then*

1. Position and log: (t, C_n, L) *is a logged position, and*
2. Tape and direction: $d = \downarrow^{|T|_l}$.

Proof. By induction on the execution $s_0 \to^k_{\lambda\text{IAM}} s$ from the initial state s_0. If $k = 0$, $s = i = (\underline{t}, \langle\cdot\rangle, \epsilon, \bullet^k)$. Clearly $\langle\cdot\rangle$ is a level 0 context, and $|L| = 0$. Moreover, $|T|_{\mathsf{e}} = 0$ and $\downarrow^0 = \downarrow$. Now, let us consider a IAM run of length $k > 0$ and let $\{s_h\}_{0\le h\le k}$ be the sequence of states of this run. By induction hypothesis $s_{k-1} = (t, C_n, T, L, d)$ is a logged position i.e $|L| = n$ and $\downarrow^{|T|_{\mathsf{e}}} = d$. We can show, by cases, that the Lemma holds for s_k. □

Note that, because of the invariant, the tape T of a reachable $\uparrow$-state always contains at least one logged position, which is why it can be seen as the answer to a query about the head variable. More generally, the parity of a logged position l on the tape determines the role of l. If l is the n-th position on T (from the right) and n is odd, then l was added by $\to_{\mathsf{var}}$ and denotes a found variable waiting for an argument, while if n is even then l was added by $\to_{\mathsf{bt1}}$, its argument was already found, and the machine is backtracking to l.

The Exhaustible State Invariant. The study of the λIAM requires to prove that some bad configurations never arise. On states such as $(\underline{\lambda x.D\langle x\rangle}, C, L, l\cdot T)$, transition $\to_{\mathsf{bt2}}$ requires the logged position l to have shape $(x, \lambda x.D, L')$, that is, to contain a position isolating an occurrence of x in $\lambda x.D\langle x\rangle$, otherwise the machine is stuck. The *exhaustible state invariant* guarantees that the machine never gets stuck for this reason. The invariant being technical, it is developed in the Section 3.5. Here we only mention a key consequence.

Proposition 3.3.3 (Logged Positions Never Block the λIAM). *Let* $(\underline{\lambda x.D\langle x\rangle}, C, L, l\cdot T)$ *be a reachable state. Then* $l = (x, \lambda x.D, L')$.

Reversibility. The proof of Prop. 3.3.3 relies on a key property of the λIAM, bi-determinism, or reversibility: the machine is deterministic, and moreover for each state s there is at most one state s' such that $s' \to_{\lambda\text{IAM}} s$. The property follows by simply inspecting the rules. Moreover, a run can be reverted by just switching the direction.

Proposition 3.3.4 (Reversibility). *If*
$(t, C, L, T, d) \to_{\lambda\text{IAM}} (u, D, L', T', d')$, *then* $(u, D, L', T', d'^1) \to_{\lambda\text{IAM}} (t, C, L, T, d^1)$.

Final States. A run of initial state $s_t = (\underline{t}, \langle\cdot\rangle, \epsilon, \epsilon)$ may either never stop or end in a final state of the shape $(\underline{\lambda x.u}, C, L, \epsilon)$. This is the machine's way of saying that t terminates when evaluated under Closed CbN strategy.

The Semantics. The characterization of final states induces a semantic interpretation of terms, that we shall show to be sound and adequate with respect to weak (linear) head evaluation.

Definition 3.3.5 (λIAM Semantics)**.** *We define the λIAM semantics of λ-terms by way of a function $[\![\cdot]\!] : \Lambda \to \{\Downarrow, \bot\}$, defined as follows.*

$$[\![t]\!] := \begin{cases} \Downarrow & \text{if } s_t \to^*_{\lambda\text{IAM}} (\lambda x.u, C, L, \epsilon), \\ \bot & \text{if the } \lambda\text{IAM diverges on } s_t. \end{cases}$$

Lifting. The λIAM verifies a sort of context-freeness with respect to the tape T. Intuitively, *lifting* the tape preserves the shape of the run and of the final state (up to lifting).

Lemma 3.3.6 (Lifting)**.** *If $(t, C, L, T, d) \to^n_{\lambda\text{IAM}} (u, D, L', T', d')$, then $(t, C, L, T{\cdot}T'', d) \to^n_{\lambda\text{IAM}} (u, D, L', T'{\cdot}T'', d')$.*

Proof. We proceed by induction on n. Thus we have that if $(t, C, L, T, d) \to^{n-1}_{\lambda\text{IAM}} (u, D, L', T', d')$, then $(t, C, L, T{\cdot}T'', d) \to^{n-1}_{\lambda\text{IAM}} (u, D, L', T' \cdot T'', d')$. The proof now proceeds analyzing all possible transitions from (u, D, T, T, d') and $(u, D, L', T{\cdot}T'', d')$. The key point is that every transition of the λIAM consumes at most 1 element of the tape. This is why the pushed stack T'' never gets touched. □

3.4 Micro-Step Refinement

The proof of soundness of the λIAM cannot be directly carried out with respect to weak head evaluation: this is specified using meta-level substitutions, here noted $t\{x{\leftarrow}u\}$, which is a macro operation, potentially making *many* copies of u and modifying t in *many* places, while the λIAM does a minimalist evaluation that in general does not even pass through most of those many places. It is very hard—if possible at all—to define explicitly a bisimulation of λIAM runs (as required for soundness) that relates states whose code is modified by meta-level substitution. We then switch to weak *linear* head evaluation (shortened to WLHE), a refinement of weak head evaluation in which substitution is performed in *micro-steps*, replacing only the head variable occurrence, and keeping the substitution suspended for all the other occurrences. This is also the approach followed by Danos et al. (1996). We depart from their approach, however, in the way we formally define LHE. We adopt a formulation where the suspension of the substitution is formalized via a sharing constructor $t[x{\leftarrow}u]$, which is nothing else but a compact notation for let $x = u$ in t, and the rewriting is modified accordingly. They instead avoid sharing, by encoding $t[x{\leftarrow}u]$ as $(\lambda x.t)u$, which is more compact but conflates different concepts and makes the technical development less clean. An important point is that weak head evaluation and its linear variant are observationally equivalent, that is, one terminates on t if and only if the other terminates on t, and they produce the same head variable, as we shall discuss below.

The Adopted Presentation. (Weak) linear head evaluation was introduced by Mascari and Pedicini (1994) and Danos and Regnier (2004) as a strategy on proof nets. It is to proof nets for the λ-calculus what (weak) head evaluation is to the λ-calculus. The presentation adopted here, noted $\to_{\mathsf{lh}}$, was introduced by Accattoli and Kesner (Accattoli, 2012; Accattoli et al., 2014b), formulated as a strategy in a λ-calculus with explicit sharing, the *linear substitution calculus*[8] (shortened to LSC). The LSC presentation of $\to_{\mathsf{lh}}$ is isomorphic to the one on proof nets (Accattoli, 2018), while the one used by Danos and Regnier—although closely related to proof nets—is not. It is isomorphic only up to the σ-equivalence by Regnier (1994).

[8]The LSC is a subtle reformulation of Milner's calculus with explicit substitutions (Kesner and Conchúir, 2008; Milner, 2007), inspired by the structural λ-calculus of Accattoli and Kesner (2010).

RULES AT TOP LEVEL

$$\begin{array}{rcll} \langle \lambda x.t\rangle S u & \mapsto_{\mathsf{dB}} & \langle t[x\leftarrow u]\rangle S & \\ H\langle x\rangle[x\leftarrow t] & \mapsto_{\mathsf{ls}} & H\langle t\rangle[x\leftarrow t] & \\ t[x\leftarrow u] & \mapsto_{\mathsf{gc}} & t & \text{if } x\notin \mathsf{fv}(t) \end{array}$$

CONTEXTUAL CLOSURE

$$\frac{t \mapsto_{\mathsf{a}} u}{H\langle t\rangle \to_{\mathsf{a}} H\langle u\rangle}\ \mathsf{a}\in\{\mathsf{dB},\mathsf{ls},\mathsf{gc}\}$$

NOTATION

$$\to_{\mathsf{lh}} := \to_{\mathsf{dB}} \cup \to_{\mathsf{ls}} \cup \to_{\mathsf{gc}}$$

FIGURE 3.3: Rewriting rules for linear head evaluation $\to_{\mathsf{lh}}$.

LSC Terms and Leveled contexts. Let $\mathcal{V}$ be a countable set of variables. Terms of the *linear substitution calculus* (LSC) are defined by the following grammar.

$$\text{LSC TERMS} \qquad t,u,r \ ::= \ x\in\mathcal{V} \ \big|\ \lambda x.t \ \big|\ tu \ \big|\ t[x\leftarrow u]$$

The construct $t[x\leftarrow u]$ is called an *explicit substitution* or ES, not to be confused with meta-level substitution $t\{x\leftarrow u\}$. As it is standard, $t[x\leftarrow u]$ binds x in t, but not in u—terms are still considered up to α-conversion. Leveled contexts naturally extend to the LSC.

LEVELED CONTEXTS

$$\begin{array}{rcl} C_0 & ::= & \langle\cdot\rangle \ \big|\ \lambda x.C_0 \ \big|\ C_0 t \ \big|\ C_0[x\leftarrow t] \\ C_{n+1} & ::= & \lambda x.C_{n+1} \ \big|\ C_{n+1}t \ \big|\ C_{n+1}[x\leftarrow t] \ \big|\ tC_n \ \big|\ t[x\leftarrow C_n] \end{array}$$

Contexts and Plugging. The LSC makes a crucial use of contexts to define its operational semantics. First of all, we need *substitution contexts*, that simply pack together ES and *weak linear head contexts*, under which reduction rules are closed.

$$\begin{array}{rrcl} \text{SUBSTITUTION CTXS} & S & ::= & \langle\cdot\rangle \ \big|\ S[x\leftarrow t] \\ \text{WEAK LINEAR HEAD CTXS} & H & ::= & \langle\cdot\rangle \ \big|\ H[x\leftarrow t] \ \big|\ Ht \end{array}$$

When plugging is used for substitution contexts, we write it in a post-fixed manner, that is $\langle t\rangle S$, to stress that the ES actually appears on the right of t.

Weak Linear Head Evaluation. The LSC comes with a notion of reduction that resembles the decomposed, micro-step process of cut-elimination in linear logic proof-nets. Essentially, the meta-level substitution $t\{x\leftarrow u\}$ is decomposed into a sequence replacements from $t[x\leftarrow u]$ of one occurrence of x in t with u at the time. Weak linear head evaluation, moreover, is the reduction that replaces only the head variable occurrence y, if it is bound by an ES $[y\leftarrow r]$ and leaves the other occurrences of y, if any, bound by $[y\leftarrow r]$.

The rewriting rules, in Figure 3.3, are first defined at top level and then closed by weak head contexts. A feature of the LSC is that contexts are also used to define the linear substitution rule at top level $\mapsto_{\mathsf{ls}}$. In plugging t in H, rule $\to_{\mathsf{ls}}$ may perform on-the-fly renaming of bound variables in H, to avoid capture of free variables of t. Often, the literature does not include rule $\to_{\mathsf{gc}}$, responsible for erasing steps, in the definition of $\to_{\mathsf{lh}}$. The reason is that $\to_{\mathsf{gc}}$ is strongly normalizing and it can be postponed. Note that our definition of $\to_{\mathsf{lh}}$ is non-deterministic, for instance $t := (\lambda x.(y[y\leftarrow u]))r \to_{\mathsf{ls}} (\lambda x.(u[y\leftarrow u]))r$ and $t \to_{\mathsf{dB}} x[y\leftarrow u][x\leftarrow r]$. It is not a problem, as $\to_{\mathsf{lh}}$ has the diamond property—this is standard, see (Accattoli, 2012).

Example 3.4.1. *We provide here an example of LHE sequence. Consider the following 3 steps:*

$$\begin{array}{lll} (\lambda x.xx)(\lambda y.y) & \to_{\mathsf{dB}} & (xx)[x\leftarrow\lambda y.y] \to_{\mathsf{ls}} ((\lambda y.y)x)[x\leftarrow\lambda y.y] \\ & \to_{\mathsf{dB}} & y[y\leftarrow x][x\leftarrow\lambda y.y] \end{array}$$

that turn a β/multiplicative redex into a ES, substitute on the head variable occurrence, and continue with another multiplicative step. Two linear substitution steps on the head variable, followed by two steps of

Sub-tm	Context	Log	T		Sub-tm	Context	Log	T
$\underline{t[x\leftarrow u]}$	C	L	T	$\to_{\mathsf{es}}$	$\underline{t}$	$C\langle\langle\cdot\rangle[x\leftarrow u]\rangle$	L	T
$\underline{x}$	$C\langle D_n[x\leftarrow u]\rangle$	$L_n \cdot L$	T	$\to_{\mathsf{var2}}$	$\underline{u}$	$C\langle D_n\langle x\rangle[x\leftarrow\langle\cdot\rangle]\rangle$	$l_n \cdot L$	T
t	$\underline{C\langle\langle\cdot\rangle[x\leftarrow u]\rangle}$	L	T	$\to_{\mathsf{es2}}$	$t[x\leftarrow u]$	$\underline{C}$	L	T
u	$\underline{C\langle D_n\langle x\rangle[x\leftarrow\langle\cdot\rangle]\rangle}$	$l_n \cdot L$	T	$\to_{\mathsf{var3}}$	x	$\underline{C\langle D_n[x\leftarrow u]\rangle}$	$L_n \cdot L$	T

where $l_n := (x, D_n[x\leftarrow u], L_n)$

FIGURE 3.4: Transitions for LSC-terms.

garbage collection complete the evaluation:

$$\begin{array}{lll} y[y\leftarrow x][x\leftarrow\lambda y.y] & \to_{\mathsf{ls}} & x[y\leftarrow x][x\leftarrow\lambda y.y] \\ & \to_{\mathsf{ls}} & (\lambda y.y)[y\leftarrow x][x\leftarrow\lambda y.y] \to^{2}_{\mathsf{gc}} \lambda z.z \end{array}$$

Additional λIAM Transitions. The λIAM presented in the previous sections is easily adapted to the LSC, by simply considering (logged) positions with respect to the extended syntax, and adding the four transitions for ES in Fig. 3.4. Transitions $\to_{\mathsf{es}}$ and $\to_{\mathsf{es2}}$ simply skip ES during search—now search is *up to β-redexes and ES*. Transition $\to_{\mathsf{var2}}$ shortcuts the search of the term u to substitute for x, given that u is already available in $[x\leftarrow u]$. Therefore, the machine stays in the $\downarrow$ phase and moves to evaluate u. Note that the logged position for x is directly added to the log and not to the tape. This is because we have avoided the search of the argument. We have reached it directly: note that when a $\uparrow$-search ends with the $\to_{\mathsf{arg}}$ transition, the logged position indeed goes from the tape to the log. Transition $\to_{\mathsf{var3}}$ is dual to $\to_{\mathsf{var2}}$, and it is used to keep looking for arguments when the current subterm u has none left.

All results and considerations of Sect. 3.2 and 3.3 still hold in this more general setting, *mutatis mutandis*.

Relationship with (Weak) Head Evaluation, and Normal Forms. We shall prove that the λIAM is sound and adequate with respect to weak linear head evaluation $\to_{\mathsf{lh}}$, by showing that it approximates the spine structure of the $\to_{\mathsf{lh}}$-normal form of t, when it exists. Let us explain why the same holds also for weak head evaluation. The relationship between (weak) linear head and (weak) head evaluation is studied in detail by Accattoli and Dal Lago (2012), who prove that the two notions are termination equivalent and produce the same normal forms up to unfolding. The definition of unfolding $t\downarrow$ of a term with ES follows.

$$\begin{array}{rclcrcl} x\downarrow & := & x & & (tu)\downarrow & := & t\downarrow u\downarrow \\ (\lambda x.t)\downarrow & := & \lambda x.t\downarrow & & (t[x\leftarrow u])\downarrow & := & t\downarrow\{x\leftarrow u\downarrow\} \end{array}$$

Proposition 3.4.2 (Accattoli and Dal Lago, 2012)**.** *Let t be a λ-term. There exists a head evaluation $t \to^{*}_{\mathsf{h}} \mathit{hnf}(t)$ to head normal form if and only if there exists a linear head evaluation $t \to^{*}_{\mathsf{lh}} \mathit{lhnf}(t)$ to linear head normal form. Moreover, $\mathit{lhnf}(t)\downarrow = \mathit{hnf}(t)$*[9]*.*

The same result holds also in the weak case present here.

3.5 The Exhaustible State Invariant

The previous sections introduced all the ingredients for the formal study of the λIAM. From now on, we turn to the development of the proofs of soundness and adequacy. The first step, taken here, is to formalize the exhaustible state invariant mentioned in Sect. 3.3.

The intuition behind the invariant is that whenever a logged position l occurs in a reachable state, it is there *for a reason*: no logged positions occur in initial states, and transitions only add

[9] The same theorem applies also in the case of *weak* head linear reduction.

logged positions to which the machine may come back. In particular, if the state is set in the right way (to be explained), the λIAM can reach l, *exhausting* it.

Why It Is Needed. The exhaustible state invariant is meant to show that some undesirable configurations never arise, to characterize the final states of the λIAM. On states such as $(\underline{\lambda x.D\langle x\rangle}, C, L, l \cdot T)$ the λIAM requires the logged position l to have shape $(x, \lambda x.D, L')$, that is, to be associated to a position isolating an occurrence of x in $\lambda x.D\langle x\rangle$, otherwise the machine is stuck. Similarly, on states such as $(t, \underline{C\langle D\langle x\rangle[x \leftarrow \langle\cdot\rangle]\rangle}, l \cdot L, T)$ the position of l is expected to isolate an occurrence of x in $D\langle x\rangle$, or the machine is stuck. Luckily, the machine is never stuck for these reasons, and exhaustible states are the technical tool to prove it[10]. The invariant also allows to prove that the backtracking to a logged position l always ends on a state of position l, as expected—that is, backtracking always succeeds, and to characterize the structure of the tape in final states.

Preliminaries. Exhaustible states rest on some *tests* for their logged positions. More specifically, each logged position l in a state s has an associated test state s_l that tunes the data structures of s as to test for the reachability of l. Actually, there shall be *two* classes of test states, one accounting for the logged positions in the tape of s, and one for those in the log of s.

Tape Tests. Tape tests are easy to define. They focus on one of the logged positions in the tape, discarding everything that follows.

Definition 3.5.1 (Tape tests). *Let $s = (t, C, L, T' \cdot l \cdot T'', d)$ be a state. Then the* tape test of s of focus l *is the state $s_l = (t, C, L, T' \cdot l, \uparrow^{|T' \cdot l|_l})$.*

Note that the direction of tape tests is reversed with respect to what stated by the *tape and direction* invariant (Lemma 3.3.2), and so, in general, they are not reachable states. Such a counterintuitive fact is needed for the invariant to go through, no more no less. When proving that backtracking always succeeds (Lemma 3.5.6 below), we shall extend their tape via the tape lifting property (Lemma 3.3.6) as to satisfy the invariant and be reachable.

Log Tests. The idea is the same underlying tape tests: they focus on a given logged position in the log. Their definition however requires more than simply stripping down the log, as the new log and the state position still have to form a logged position, which requires to change the state position—said differently, the *position and the log* invariant (Lemma 3.3.2) has to be preserved.

In the applications of the exhaustible invariant given below, we need only log tests of a very simple form. Namely, given a state $s = (t, \underline{C}, l \cdot L, T)$, we shall consider the log test $s_l := (t, \underline{C}, l \cdot L, \epsilon)$, obtained by emptying the tape and (in this case) without changing the position. The more general form of log tests needing the position change is technical and defined at the end of the section—it is unavoidable for proving the invariant, but we fear that giving it here would obfuscate the use of the exhaustible invariant, whose idea is instead quite simple.

The Exhaustibility Invariant. Exhausting a logged position l means backtracking to it. We then decorate the backtracking transition $\to_{\mathsf{bt1}}$ and $\to_{\mathsf{bt2}}$ as $\to_{\mathsf{bt1},l}$ and $\to_{\mathsf{bt2},l}$ to specify the involved logged position l. We also need a notion of state positioned in l, which is the target state of $\to_{\mathsf{bt2},l}$.

Definition 3.5.2 (State surrounding a position). *Let $l = (t, D, L')$ be a logged position. A state s surrounds l if $s = (t, \underline{C_n\langle D\rangle}, L' \cdot L_n, \epsilon)$ for some context C_n and log L_n.*

After having introduced all the necessary preliminaries, we can define exhaustible states.

Definition 3.5.3 (Exhaustible States). *$\mathcal{E}$ is the smallest set of states s such that if s_l is a tape or a log test of s then there exists a run $\rho : s_l \to^*_{\lambda\mathrm{IAM}} \to_{\mathsf{bt2},l} s'$, where s' surrounds l and for the shortest of such runs ρ it holds that $s' \in \mathcal{E}$. States in $\mathcal{E}$ are called* exhaustible.

[10] One could redefine the transitions of the λIAM asking—for these states—to jump to whatever variable position is in the logged position l. Then the λIAM would not get stuck, and the invariant would not be needed for characterizing final states, but we would then need it for soundness—there is no easy way out.

Informally, exhaustible states are those for which every logged position can be successfully tested, that is, the λIAM can backtrack to (an exhaustible state surrounding) it, if properly initialized. Roughly, a state is exhaustible if the backtracking information encoded in its logged positions is coherent. The set $\mathcal{E}$ being the *smallest* set of such states implies that checking that a state is exhaustible can be finitely certified, *i.e.* there must be a finitary proof.

Proposition 3.5.4 (Exhaustible invariant). *Let s be a λIAM reachable state. Then s is exhaustible.*

The proof of Prop. 3.5.4 is long, but logically quite simple, being structured around a simple induction on the length of the run from the initial state to s, and can be found in the Appendix.

Consequences of the Exhaustible Invariant. First, the λIAM never gets stuck for a mismatch of logged positions.

Corollary 3.5.5 (Logged Positions Never Block the λIAM). *Let s be a reachable state.*
1. *If $s = (\underline{\lambda x.D\langle x\rangle}, C, L, l\cdot T)$ then s is not final.*
2. *If $s = (u, \underline{C\langle D\langle x\rangle[x \leftarrow \langle\cdot\rangle]\rangle}, l\cdot L, T)$ then s is not final.*

Proof. For point 1, by the exhaustible invariant (Prop. 3.5.4), s is exhaustible. Then, its tape test $(\underline{\lambda x.D\langle x\rangle}, C, L, l)$ does at least one transition towards a state $s' \neq s$ surrounding l. Point 2 is analogous, just consider the log test $s' = (u, \underline{C\langle D\langle x\rangle[x \leftarrow \langle\cdot\rangle]\rangle}, l\cdot L, \epsilon)$. □

The second consequence is that backtracking always succeeds.

Lemma 3.5.6 (Backtracking always succeeds). *Let s a reachable state. If $s \to_{\mathsf{bt1},l} s'$ then there is s'' such that $s' \to^*_{\lambda\mathrm{IAM}} \to_{\mathsf{bt2},l} s''$.*

Proof. Consider $s = (t, \underline{C\langle u\langle\cdot\rangle\rangle}, l\cdot L, T) \to_{\mathsf{bt1},l} (\underline{u}, C\langle\langle\cdot\rangle t\rangle, L, l\cdot T) = s'$. Since s' is reachable then it is exhaustible, and so its tape test ${s'}_l := (\underline{u}, C\langle\langle\cdot\rangle t\rangle, L, l)$ can be exhausted, that is, there is a run $\rho : {s'}_l \to^*_{\lambda\mathrm{IAM}} \to_{\mathsf{bt2},l} q$ for a state q surrounding l. Note that ${s'}_l$ is s' with empty tape. Now, we lift ρ to a run $\rho^T : s' \to^*_{\lambda\mathrm{IAM}} \to_{\mathsf{bt2},l} s''$ by using the tape lifting lemma (Lemma 3.3.6). □

Finally, we characterize final states, as anticipated in Section 3.3.

Lemma 3.5.7 (Final states). *Let $s_t \to^*_{\lambda\mathrm{IAM}} s$ be a run ending on a final state. Then $s = (\underline{\lambda x.u}, C, L, \epsilon)$ for some C and L.*

Proof. The λIAM is never stuck on $\to_{\bullet 1}$, $\to_{\bullet 4}$, $\to_{\mathsf{es}}$, and $\to_{\mathsf{es2}}$. By the balance invariant (Lemma 3.3.2), it is also never stuck on $\to_{\mathsf{var}}$, $\to_{\mathsf{var2}}$, and $\to_{\mathsf{bt1}}$ because the log has not enough entries. Note also that if the position is $(t, C\langle\langle\cdot\rangle u\rangle)$ and the direction is $\uparrow$, one among $\to_{\bullet 3}$ and $\to_{\mathsf{arg}}$ always applies, as the tape cannot be empty, by the balance invariant. By Corollary 3.5.5, the λIAM cannot be stuck on $\to_{\mathsf{bt2}}$ and $\to_{\mathsf{var3}}$. Then, the machine is stuck only on $\to_{\bullet 2}$ if the tape is empty—the stuck state is $(\underline{\lambda x.u}, C, L, \epsilon)$. □

Log Tests and Position Changes. To define the log test focussing on the m-th logged position l_m in the log of a state $(t, C_n, l_n \cdots l_2 \cdot l_1, T, d)$, we remove the prefix $l_n \cdots l_{m+1}$ (if any), and move the current position up by $n - m$ levels. Moreover, the tape is emptied and the direction is set to $\uparrow$. Let us define the position change.

Definition 3.5.8 (Outer Contexts and Positions). *Let (u, C_{n+1}) be a position. Then, for every decomposition of n into two natural numbers m, k with $m + k = n$, we can find contexts C_m and C_k, and a term r satisfying exactly one of the two following conditions (levels can be incremented in two ways).*
- *Case $t = C_m\langle rC_k\langle u\rangle\rangle$. Then, the $m+1$-*outer context *of the position (u, C_{n+1}) is the context $O_{m+1} := C_m\langle r\langle\cdot\rangle\rangle$ of level $m+1$ and the $m+1$-*outer position *is $(C_k\langle u\rangle, O_{m+1})$.*
- *Case $t = C_m\langle r[x \leftarrow C_k\langle u\rangle]\rangle$. Then, the $m+1$-*outer context *of the position (u, C_{n+1}) is the context $O_{m+1} := C_m\langle r[x \leftarrow \langle\cdot\rangle]\rangle$ of level $m+1$ and the $m+1$-*outer position *is $(C_k\langle u\rangle, O_{m+1})$.*

Note that the m-outer context and the m-outer position (of a given position) have level m. It is easy to realize that any position having level n has *unique* m-outer context and m-outer position, for every $1 \leq m \leq n+1$, and that, moreover, outer positions are hereditary, in the following sense: the i-outer position of the m-outer position of (u, C_{n+1}) is exactly the i-outer position of (u, C_{n+1}).

Definition 3.5.9 (Log tests). *Let $s = (t, C_n, l_n \cdots l_2 \cdot l_1, T, d)$ be a state with $1 \leq m \leq n$, and (u, O_m) be the m-outer position of (t, C_n). The m-*log test of s of focus l_m *is the state $s_{l_m} := (u, O_m, l_m \cdots l_2 \cdot l_1, \epsilon, \uparrow)$.*

By definition, log tests for s do not depend on the direction of s, nor on the underlying tape, and they are stable by head translations of the position (t, C_n) of s, in the sense that if $t = H\langle r \rangle$ then $s = (t, C_n, L, T, d)$ and its head translation $(r, C_n\langle H \rangle, L, T', d)$ induce the same log tests (because the two positions have the same outer positions and the two states have the same logs).

Lemma 3.5.10 (Invariance properties of log tests). *Let $s = (t, C_n, L_n, T, d)$ be a state. Then:*

1. Direction*: the dual (t, C_n, L_n, T, d^1) of s induces the same log tests;*
2. Tape*: the state (t, C_n, L_n, T', d) obtained from s replacing T with an arbitrary tape T' induces the same log tests;*
3. Head translation*: if $t = H\langle r \rangle$ then the head translation $(r, C_n\langle H \rangle, L_n, T', d)$ of s induces the same log tests.*
4. Inclusion*: if $C_n = C_m\langle C_i \rangle$ and $L_n = L_i \cdot L_m$ then log tests of $(C_i\langle t \rangle, C_m, L_m, T', d)$ are log tests of s.*

Proof. The first three points are immediate consequences of the definition of log test. We prove the fourth point. Let $s' = (C_j\langle t \rangle, C_i, L_i, T, d)$. By induction on j. If $j = 0$ then $i = n$ and $s = s'$, therefore the statement is simply says that the log test of s is itself, that is obviously true. Let $j > 0$. By *i.h.*, the log test s'' of $(C_{j-1}\langle t \rangle, C_{i+1}, l \cdot L_i, T, d)$ is s_{l_i}. Let us spell out s''. If $C_{i+1} = C_i\langle u C_0 \rangle$ then $s'' = (u, C_i\langle\langle\cdot\rangle C_{j-1}\langle t \rangle\rangle, L_i, l, \uparrow)$. Note that $C_j = C_i\langle u\langle\cdot\rangle\rangle$. Since $s'' = s_{l_i}$, we have $s_{l_{i-1}}$ is a log test of s''. Now, since log tests are stable by head translation (Point 3), we have that $s_{l_{i-1}}$ is also the log test of the translation of s'' with respect to $C_{j-1}\langle t \rangle$, that is, of the state $(u C_{j-1}\langle t \rangle, C_i, L_i, l, \uparrow) = (C_j\langle t \rangle, C_i, L_i, l, \uparrow)$. □

3.6 Improvements

Implementation Theorem(s) for the λIAM. The λIAM, and more generally GoI machines, do implement strategies, but in a different way w.r.t. environment machines. The key point is that these machines do not trace how the strategy modifies the term. If $t \to_{wh} u$, a λIAM run of code t never passes through a representation of u, as *implementing* $\to_{wh}$ here denotes something else. There are actually *two* implementation theorems, called *soundness* and *adequacy*. Soundness is the fact that the run of code t is bisimilar to the run of code u. Note the difference with environment machines: there, the bisimulation is between *steps* on terms and transitions on states. For the λIAM, it is between *transitions* on states (of code t) and transitions on states (of code u). The important consequence of soundness is that the runs from t and u produce bisimilar final states (if they terminate). The idea is that these final states are different but encode the same description of the normal form of t and u, which is the semantics $[\![t]\!]$ induced by the λIAM. Adequacy guarantees that the interpretation $[\![t]\!]$ reflects some observable aspects of t. For a weak head evaluation[11], one usually observes termination only. This is exactly what $[\![t]\!]$ reflects, or it is adequate for. Environment machines implementing a strategy $\to$ stop on final states s_f decoding to $\to$-normal forms.

[11] In the general case, the IAM implements (linear) head reduction, as we prove in (Accattoli et al., 2020a). However, for the sake of uniformity, here we present it in a slightly simplified way: we choose to not evaluate under λ-abstractions *i.e.* following the *weak* (linear) head reduction, dubbed also Closed CbN. Indeed, the other machines presented in this thesis evaluate according to the same Closed CbN strategy.

3.6.1 Improvements, Abstractly

To prove the soundness and adequacy theorems we shall use *improvements*, a refinement of the classical notion of bisimulation inspired by Sands (1996), and explained here. The concrete improvements at work in the proofs of the theorems are defined in Sect. 3.6.

An improvement is a weak bisimulation between two transition systems preserving termination and guaranteeing that, whenever s and q are related and terminating, then q terminates in no more steps than s—the *no-more-steps* part implies that the definition is asymmetric in the way it treats the two transition systems, and it shall play a key role in the proof of adequacy.

Preliminaries. A deterministic transition system (DTS) is a pair $\mathcal{S} = (S, \mathcal{T})$, where S is a set of *states* and $\mathcal{T} : S \rightharpoonup S$ a partial function. If $\mathcal{T}(s) = s'$, then we write $s \to s'$, and if s rewrites in s' in n steps then we write $s \to^n s'$. We note with $\mathcal{F}_S$ the set of final states, *i.e.* the subset of $\mathcal{S}$ containing all $s \in \mathcal{S}$ such that $\mathcal{T}(s)$ is undefined. A state s is terminating if there exists $n \geq 0$ and $s' \in \mathcal{F}_S$ such that $s \to^n s'$. We call $S_\downarrow$ the set of terminating states of S and $S_\uparrow$ stands for $S \setminus S_\downarrow$. The *evaluation length map* $|\cdot| : S \to \mathbb{N} \cup \{\infty\}$ is defined as $|s| := n$ if $s \to^n s'$ and $s' \in \mathcal{F}_\mathcal{S}$, and $|s| := \infty$ if $s \in \mathcal{S}_\uparrow$.

Definition 3.6.1 (Improvements)**.** *Given two DTS $\mathcal{S}$ and $\mathcal{Q}$, a relation $\mathcal{R} \subseteq S \times Q$ is an* improvement *if given $(s, q) \in \mathcal{R}$ the following conditions hold (for 2 and 3 see Fig. 3.5).*

1. Final state right*: if $q \in \mathcal{F}_\mathcal{Q}$, then $s \to^n s'$, for some $s' \in \mathcal{F}_\mathcal{S}$ and $n \geq 0$.*
2. Transition left*: if $s \to s'$, then there exists s'', q', n, m such that $s' \to^m s''$, $q \to^n q'$, $s'' \mathcal{R} q'$ and $n \leq m + 1$.*
3. Transition right*: if $q \to q'$, then there exists s', q'', n, m such that $s \to^m s'$, $q' \to^n q''$, $s' \mathcal{R} q''$ and $m \geq n + 1$.*

What improves along an improvement is the number of transitions required to reach a final state, if any.

Proposition 3.6.2. *Let $\mathcal{R}$ be an improvement on two DTS $\mathcal{S}$ and $\mathcal{Q}$, and $s \mathcal{R} q$.*

1. Termination equivalence*: $s \in \mathcal{S}_\downarrow$ if and only if $q \in \mathcal{Q}_\downarrow$.*
2. Improvement*: $|s| \geq |q|$.*

Proof.

1. $\Rightarrow$. Let us suppose $s \in \mathcal{S}_\downarrow$ and let n be the number of steps that s needs to terminate. We proceed by induction on n. If $n = 0$, $s \in \mathcal{F}_\mathcal{S}$ and since $s \mathcal{R} q$, $q \in \mathcal{F}_\mathcal{Q}$ and thus $q \in \mathcal{Q}_\downarrow$. If $n = h > 0$, then $s \to s'$, and thus there exists s'', q', k, j such that $q \to^k q'$, $s' \to^j s''$, $s'' \mathcal{R} q'$ and $k \leq j + 1$. Since s'' terminates in less than $h - 1$ steps, by induction hypothesis $q' \in \mathcal{Q}_\downarrow$ and thus also $q \in \mathcal{Q}_\downarrow$.

   $\Leftarrow$. Let us suppose $q \in \mathcal{Q}_\downarrow$ and let n be the number of steps that q needs to terminate. We proceed by induction on n. If $n = 0$, $q \in \mathcal{F}_\mathcal{Q}$ and since $s \mathcal{R} q$, $s \in \mathcal{S}_\downarrow$. If $n = h > 0$, then $q \to q'$, and thus there exists s', q'', k, j such that $s \to^k s'$, $q' \to^j q''$, $s' \mathcal{R} q''$ and $k \geq j + 1$. Since q'' terminates in less than h steps, by induction hypothesis $s' \in \mathcal{S}_\downarrow$ and thus also $s \in \mathcal{S}_\downarrow$.

2. If $s \in \mathcal{S}_\uparrow$ and $q \in \mathcal{Q}_\uparrow$, then $|s| = |q| = \infty$. Let us consider the other case, *i.e.* when $s \in \mathcal{S}_\downarrow$ and $q \in \mathcal{Q}_\downarrow$. We proceed by induction on $|s|$. If $|s| = 0$, then $q \in \mathcal{F}_\mathcal{Q}$ and thus also $|q| = 0$. If $|s| = n > 0$, then $s \to s'$ and there exists s'', q', m, l such that $q \to^m q'$, $s' \to^l s''$, $s'' \mathcal{R} q'$ and $m \leq l + 1$. By *i.h.*, $|s''| \geq |q'|$. Thus, since $m \leq l + 1$, then $|s| = |s''| + l + 1 \geq |q'| + m = |q|$.

□

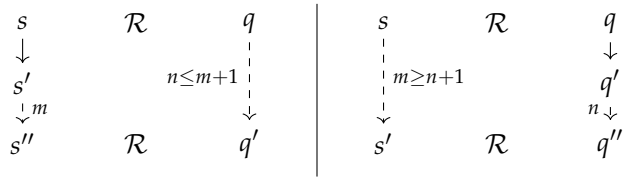


FIGURE 3.5: Diagrammatic definition of improvements.

3.6.2 Improvements, Concretely

In this section we define an improvement ▶ relation for the λIAM, to be used in the sequel to prove soundness and adequacy.

Given a $\to_{\mathsf{lh}}$-step $t \to_{\mathsf{lh}} u$, the improvement ▶ has to relate states of code t with states of code u. Since $\to_{\mathsf{lh}}$ is the union of the three rewriting rules $\to_{\mathsf{dB}}$, $\to_{\mathsf{ls}}$ and $\to_{\mathsf{gc}}$, we are going to define ▶ as the union of three improvements $\blacktriangleright_{\mathsf{dB}}$, $\blacktriangleright_{\mathsf{ls}}$, and $\blacktriangleright_{\mathsf{gc}}$.

Improvement for $\to_{\mathsf{dB}}$. Lifting a step $t \to_{\mathsf{lh}} u$ to a relation between a λIAM state s of code t and a state q of code u requires changing all positions relative to t in s to positions relative to u in q. Note that all the positions in the token have to be changed, so that ▶ has to relate positions, logged positions, tape, log, and states.

Explaining the Need of Context Rewriting. A second more technical aspect is that one needs to extend weak linear head evaluation to contexts. Consider a step $t \to_{\mathsf{dB}} u$ where—for simplicity—the redex is at top level and the associated state $(\underline{\langle \lambda x.r \rangle Sw}, \langle\cdot\rangle, \epsilon, \epsilon)$ has an empty token. This should be $\blacktriangleright_{\mathsf{dB}}$-related to a state $(\underline{\langle r[x{\leftarrow}w] \rangle S}, \langle\cdot\rangle, \epsilon, \epsilon)$. Let's have a look at how the two states evolve:

$$\begin{array}{ccc}
(\underline{\langle \lambda x.r \rangle Sw}, \langle\cdot\rangle, \epsilon, \epsilon) & \blacktriangleright_{\mathsf{dB}} & (\underline{\langle r[x{\leftarrow}w] \rangle S}, \langle\cdot\rangle, \epsilon, \epsilon) \\
\downarrow & & \downarrow{\scriptstyle |S|} \\
(\underline{\langle \lambda x.r \rangle S}, \langle\cdot\rangle w, \epsilon, \bullet) & & (\underline{r[x{\leftarrow}w]}, S, \epsilon, \epsilon) \\
\downarrow{\scriptstyle |S|} & & \downarrow \\
(\underline{\lambda x.r}, Sw, \epsilon, \bullet) & & \\
\downarrow & & \\
(\underline{r}, \langle \lambda x.\langle\cdot\rangle \rangle Sw, \epsilon, \epsilon) & & (\underline{r}, \langle \langle\cdot\rangle [x{\leftarrow}w] \rangle S, \epsilon, \epsilon)
\end{array}$$

To close the diagram, we need $\blacktriangleright_{\mathsf{dB}}$ to relate the two bottom states. Note that their relation can be seen as a $\to_{\mathsf{dB}}$ step involving the contexts of the two positions. Therefore we extend the definition of $\to_{\mathsf{dB}}$ to contexts adding the following top level clause (then included in $\to_{\mathsf{dB}}$ via a closure by head contexts): $\langle \lambda x.C \rangle St \mapsto_{\mathsf{dB}} \langle C[x{\leftarrow}t] \rangle S$. The new clause, in turn, requires a further extension of $\to_{\mathsf{dB}}$ (again closed by head contexts): $\langle \lambda x.t \rangle SC \mapsto_{\mathsf{dB}} \langle t[x{\leftarrow}C] \rangle S$.

Note that in the shown local bisimulation diagram the right side is shorter. This is typical of when the machine travels through the redex. Outside of the redex, however, the two sides have the same length, as the next example shows—example that also motivates a further extension of $\to_{\mathsf{dB}}$ to contexts. Consider the case where $t \to_{\mathsf{dB}} u$ and the diagram is (the states do a $\to_{\mathsf{arg}}$ transition):

$$\begin{array}{ccc}
(t, \underline{\langle\cdot\rangle r}, \epsilon, l) & \blacktriangleright_{\mathsf{dB}} & (u, \underline{\langle\cdot\rangle r}, \epsilon, l) \\
\downarrow & & \downarrow \\
(\underline{r}, t\langle\cdot\rangle, l, \epsilon) & & (\underline{r}, u\langle\cdot\rangle, l, \epsilon)
\end{array}$$

We then need to extend $\to_{\mathsf{dB}}$ so that $t\langle\cdot\rangle \to_{\mathsf{dB}} u\langle\cdot\rangle$. A similar situation happens also when entering an ES with transition $\to_{\mathsf{var2}}$. To close these diagrams, we add two further cases of reduction on contexts. Note that this time they have to be expressed via steps on terms (then included in $\to_{\mathsf{dB}}$ via a closure by head contexts), as their direct definition would require contexts with two holes. Of course, the same situation arises with ls and gc steps.

$$\frac{t \to_{\mathsf{a}} u}{tC \to_{\mathsf{a}} uC} \qquad \frac{t \to_{\mathsf{a}} u}{t[x{\leftarrow}C] \to_{\mathsf{a}} u[x{\leftarrow}C]} \qquad \mathsf{a} \in \{\mathsf{dB}, \mathsf{ls}, \mathsf{gc}\}$$

Definition 3.6.3. *The (overloaded) binary relation $\blacktriangleright_{\mathsf{dB}}$ between positions, stacks, and states is defined by the following rules*[12].

$$\frac{t \to_{\mathsf{dB}} u}{(t,H)\blacktriangleright_{\mathsf{dB}}(u,H)}\ \mathsf{rdx}_{\mathsf{dB}} \qquad \frac{C \to_{\mathsf{dB}} D}{(t,C)\blacktriangleright_{\mathsf{dB}}(t,D)}\ \mathsf{ctx}_{\mathsf{dB}}$$

$$\frac{}{\epsilon \blacktriangleright_{\mathsf{dB}} \epsilon}\ \mathsf{tok1}_{\mathsf{dB}} \qquad \frac{T \blacktriangleright_{\mathsf{dB}} T'}{\bullet \cdot T \blacktriangleright_{\mathsf{dB}} \bullet \cdot T'}\ \mathsf{tok2}_{\mathsf{dB}}$$

$$\frac{(x,C)\blacktriangleright_{\mathsf{dB}}(x,D) \qquad L \blacktriangleright_{\mathsf{dB}} L'}{(x,C,L)\blacktriangleright_{\mathsf{dB}}(x,D,L')}\ \mathsf{pos}_{\mathsf{dB}} \qquad \frac{l \blacktriangleright_{\mathsf{dB}} l' \qquad \Gamma \blacktriangleright_{\mathsf{dB}} \Gamma'}{l \cdot \Gamma \blacktriangleright_{\mathsf{dB}} l' \cdot \Gamma'}\ \mathsf{tok3}_{\mathsf{dB}}$$

$$\frac{(t,C)\blacktriangleright_{\mathsf{dB}}(u,D) \qquad T \blacktriangleright_{\mathsf{dB}} T' \qquad L \blacktriangleright_{\mathsf{dB}} L' \qquad d = d'}{(t,C,L,T,d)\blacktriangleright_{\mathsf{dB}}(u,D,L',T',d')}\ \mathsf{state}_{\mathsf{dB}}$$

Note that $\blacktriangleright_{\mathsf{dB}}$ contains all pairs $((\underline{t}, \langle\cdot\rangle, \epsilon, \epsilon), (\underline{u}, \langle\cdot\rangle, \epsilon, \epsilon))$, where $t \to_{\mathsf{dB}} u$, *i.e.* all the pairs of initial states involving a dB-redex.

Improvement for $\to_{\mathsf{ls}}$. As for $\to_{\mathsf{dB}}$, the improvement for $\to_{\mathsf{ls}}$ requires extending the rewriting relation to contexts. There are, however, some new subtleties. Given $t \to_{\mathsf{dB}} u$ and a position (r, C) for t, for $\blacktriangleright_{\mathsf{dB}}$ the redex in t falls always entirely either in t or C. If $t \to_{\mathsf{ls}} u$, instead, the redex can be split between the two. Consider the following diagram (where to simplify we assume the step to be at top level and the token to be empty).

$$\begin{array}{ccc} (\underline{H\langle x\rangle[x{\leftarrow}r]}, \langle\cdot\rangle, \epsilon, \epsilon) & \blacktriangleright_{\mathsf{ls}} & (\underline{H\langle r\rangle[x{\leftarrow}r]}, \langle\cdot\rangle, \epsilon, \epsilon) \\ \downarrow & & \downarrow \\ (\underline{H\langle x\rangle}, \langle\cdot\rangle[x{\leftarrow}r], \epsilon, \epsilon) & & (\underline{H\langle r\rangle}, \langle\cdot\rangle[x{\leftarrow}r], \epsilon, \epsilon) \end{array}$$

To close it, we have to $\blacktriangleright_{\mathsf{ls}}$-relate the two bottom states, where the pattern of the redex/reduct is split between the two parts of the position. This motivates clause rdx2 in the definition of $\blacktriangleright_{\mathsf{ls}}$ below.

The new rule comes with consequences. Consider the following diagram involving the new clause for $\blacktriangleright_{\mathsf{ls}}$:

$$\begin{array}{ccc} (\underline{x}, H[x{\leftarrow}t], \epsilon, \epsilon) & \blacktriangleright_{\mathsf{ls}} & (\underline{t}, H[x{\leftarrow}t], \epsilon, \epsilon) := q \\ \downarrow & & \\ s =: (\underline{t}, H\langle x\rangle[x{\leftarrow}\langle\cdot\rangle], (x, H[x{\leftarrow}t], \epsilon), \epsilon) & & \end{array} \tag{1}$$

To close the diagram, we have to $\blacktriangleright_{\mathsf{ls}}$-relate s and q. There are, however, two delicate points. First, we cannot see the context $H\langle x\rangle[x{\leftarrow}\langle\cdot\rangle]$ as making a $\to_{\mathsf{ls}}$ step towards $H[x{\leftarrow}t]$, because t does not occur in $H\langle x\rangle[x{\leftarrow}\langle\cdot\rangle]$. For that, we have to introduce a variant of $\to_{\mathsf{ls}}$ on contexts that is parametric in t (and more general than the one to deal with the showed simplified diagram):

$$H\langle x\rangle[x{\leftarrow}C] \quad \mapsto_{\mathsf{ls},t} \quad H\langle C\rangle[x{\leftarrow}C\langle t\rangle].$$

The second delicate point of diagram (1) is that the extension of $\blacktriangleright_{\mathsf{ls}}$ has to also $\blacktriangleright_{\mathsf{ls}}$-relate logs of different length, namely ϵ and $(x, H[x{\leftarrow}t], \epsilon)$. This happens because positions of the two states do isolate the same term, but at different depths, as one is in the ES. Then the definition of $\blacktriangleright_{\mathsf{ls}}$ has two clauses, one for logs ($\mathsf{pos2}_{\mathsf{ls}}$) and one for states ($\mathsf{state2}_{\mathsf{ls}}$), to handle such a case. The mismatch in logs lengths is at most 1.

Definition 3.6.4. *The binary relation $\blacktriangleright_{\mathsf{ls}}$ is defined by:*

[12] Γ is a meta-variable that stands either for a log L or for a tape T.

$$\frac{t \to_{\mathsf{ls}} u}{(t,H) \blacktriangleright_{\mathsf{ls}} (u,H)}\ \mathsf{rdx}_{\mathsf{ls}} \qquad \frac{C \to_{\mathsf{ls}} D}{(t,C) \blacktriangleright_{\mathsf{ls}} (t,D)}\ \mathsf{ctx}_{\mathsf{ls}}$$

$$\frac{K = K'\langle G[x \leftarrow t]\rangle}{(H\langle x\rangle, K) \blacktriangleright_{\mathsf{ls}} (H\langle t\rangle, K)}\ \mathsf{rdx2} \qquad \frac{}{\epsilon \blacktriangleright_{\mathsf{ls}} \epsilon}\ \mathsf{tok1}_{\mathsf{ls}}$$

$$\frac{T \blacktriangleright_{\mathsf{ls}} T'}{\bullet \cdot T \blacktriangleright_{\mathsf{ls}} \bullet \cdot T'}\ \mathsf{tok2}_{\mathsf{ls}} \qquad \frac{l \blacktriangleright_{\mathsf{ls}} l' \qquad \Gamma \blacktriangleright_{\mathsf{ls}} \Gamma'}{l \cdot \Gamma \blacktriangleright_{\mathsf{ls}} l' \cdot \Gamma'}\ \mathsf{tok3}_{\mathsf{ls}}$$

$$\frac{(x,C) \blacktriangleright_{\mathsf{ls}} (x,D) \qquad L \blacktriangleright_{\mathsf{ls}} L'}{(x,C,L \blacktriangleright_{\mathsf{ls}} (x,D,L')}\ \mathsf{pos}_{\mathsf{ls}} \qquad \frac{C \to_{\mathsf{ls},x} D \qquad L \blacktriangleright_{\mathsf{ls}} L'}{(x,C,L \cdot l) \blacktriangleright_{\mathsf{ls}} (x,D,L')}\ \mathsf{pos2}_{\mathsf{ls}}$$

$$\frac{(t,C) \blacktriangleright_{\mathsf{ls}} (u,D) \qquad T \blacktriangleright_{\mathsf{ls}} T' \qquad L \blacktriangleright_{\mathsf{ls}} L' \qquad d = d'}{(t,C,L,T,d) \blacktriangleright_{\mathsf{ls}} (u,D,L',T',d')}\ \mathsf{state}_{\mathsf{ls}}$$

$$\frac{C \to_{\mathsf{ls},t} D \qquad T \blacktriangleright_{\mathsf{ls}} T' \qquad L \blacktriangleright_{\mathsf{ls}} L' \qquad d = d'}{(t,C,L \cdot l,T,d) \blacktriangleright_{\mathsf{ls}} (t,D,L',T',d')}\ \mathsf{state2}_{\mathsf{ls}}$$

Note that $\blacktriangleright_{\mathsf{ls}}$ contains all pairs $((\underline{t}, \langle\cdot\rangle, \epsilon, \epsilon), (\underline{u}, \langle\cdot\rangle, \epsilon, \epsilon))$, where $t \to_{\mathsf{ls}} u$, *i.e.* all the initial states containing a ls-redex and its reduct.

Improvement for $\to_{\mathsf{gc}}$ The candidate improvement $\blacktriangleright_{\mathsf{gc}}$ induced by $\to_{\mathsf{gc}}$ requires an extension of $\to_{\mathsf{gc}}$ with a rule on contexts similar to the parametric one for $\to_{\mathsf{ls}}$. Let $t[x \leftarrow u] \to_{\mathsf{gc}} t$ and consider:

$$\begin{array}{ccc} (\underline{t[x \leftarrow u]}, \langle\cdot\rangle, \epsilon, \epsilon) & \blacktriangleright_{\mathsf{gc}} & (\underline{t}, \langle\cdot\rangle, \epsilon, \epsilon) \\ \downarrow & & \\ (\underline{t}, \langle\cdot\rangle[x \leftarrow u], \epsilon, \epsilon) & & \end{array}$$

To close the diagram, we extend the definition of $\to_{\mathsf{gc}}$ to context with the following parametric rule (closed by head contexts):

$$C[x \leftarrow u] \ \mapsto_{\mathsf{gc},t}\ C \quad \text{if } x \notin \mathsf{fv}(t) \cup \mathsf{fv}(C).$$

Definition 3.6.5. *The binary relation $\blacktriangleright_{\mathsf{gc}}$ is defined as follows.*

$$\frac{t \to_{\mathsf{gc}} u}{(t,H) \blacktriangleright_{\mathsf{gc}} (u,H)}\ \mathsf{rdx}_{\mathsf{gc}} \qquad \frac{C \to_{\mathsf{gc}} D}{(t,C) \blacktriangleright_{\mathsf{gc}} (t,D)}\ \mathsf{ctx}_{\mathsf{gc}}$$

$$\frac{}{\epsilon \blacktriangleright_{\mathsf{gc}} \epsilon}\ \mathsf{tok1}_{\mathsf{gc}} \qquad \frac{C \to_{\mathsf{gc},t} D}{(t,C) \blacktriangleright_{\mathsf{gc}} (t,D)}\ \mathsf{ctx2}_{\mathsf{gc}}$$

$$\frac{T \blacktriangleright_{\mathsf{gc}} T'}{\bullet \cdot T \blacktriangleright_{\mathsf{gc}} \bullet \cdot T'}\ \mathsf{tok2}_{\mathsf{gc}} \qquad \frac{l \blacktriangleright_{\mathsf{gc}} l' \qquad \Gamma \blacktriangleright_{\mathsf{gc}} \Gamma'}{l \cdot \Gamma \blacktriangleright_{\mathsf{gc}} l' \cdot \Gamma'}\ \mathsf{tok3}_{\mathsf{gc}}$$

$$\frac{(x,C) \blacktriangleright_{\mathsf{gc}} (x,D) \qquad L \blacktriangleright_{\mathsf{gc}} L'}{(x,C,L) \blacktriangleright_{\mathsf{gc}} (x,D,L')}\ \mathsf{pos}_{\mathsf{gc}}$$

$$\frac{(t,C) \blacktriangleright_{\mathsf{gc}} (u,D) \qquad T \blacktriangleright_{\mathsf{gc}} T' \qquad L \blacktriangleright_{\mathsf{gc}} L' \qquad d = d'}{(t,C,L,T,d) \blacktriangleright_{\mathsf{gc}} (u,D,L',T',d')}\ \mathsf{state}_{\mathsf{gc}}$$

The proof of the next theorem is a tedious easy check of diagrams.

Theorem 3.6.6. *$\blacktriangleright_{\mathsf{ls}}$, $\blacktriangleright_{\mathsf{dB}}$ and $\blacktriangleright_{\mathsf{gc}}$ are improvements.*

3.7 Soundness and Adequacy

Here, we use the improvements of the previous sections to prove soundness and adequacy. Consider $\blacktriangleright = \blacktriangleright_{\mathsf{dB}} \cup \blacktriangleright_{\mathsf{ls}} \cup \blacktriangleright_{\mathsf{gc}}$, that is an improvement because its components are. Consequently, if $t \to_{\mathsf{lh}} u$, then the λIAM run on u improves the one on t, that is,

$$s_t = (\underline{t}, \langle\cdot\rangle, \epsilon, \epsilon) \ \blacktriangleright\ (\underline{u}, \langle\cdot\rangle, \epsilon, \epsilon) = s_u$$

Theorem 3.7.1 (Soundness). *If $t \to_{\mathsf{lh}} u$, then $[\![t]\!] = [\![u]\!]$.*

Proof. Since $t \to_{\mathsf{lh}} u$, then $s = (\underline{t}, \langle\cdot\rangle, \epsilon, \epsilon) \blacktriangleright (\underline{u}, \langle\cdot\rangle, \epsilon, \epsilon) = q$ by the results about improvements (Theorem 3.6.6). Since improvements transfer termination/divergence (Prop. 3.6.2.1), we have $[\![t]\!] = [\![u]\!]$. □

Adequacy. Adequacy is the fact that $[\![t]\!] = \Downarrow$ if and only if $\to_{\mathsf{lh}}$ terminates on t. We prove the two directions separately.

Direction λIAM to $\to_{\mathsf{lh}}$. The *only if* direction of the statement is immediate to prove. Since $[\![t]\!]$ is invariant by $\to_{\mathsf{lh}}$ (soundness) and $\to_{\mathsf{lh}}$ terminates on t, we have that $[\![t]\!] = [\![\mathsf{whnf}(t)]\!] = \Downarrow$.

Direction $\to_{\mathsf{lh}}$ to λIAM. The proof of the *if* direction of the adequacy theorem is by contraposition: if the $\to_{\mathsf{lh}}$ diverges on t then no run of the λIAM on t ends in a final state. The proof is obtained via a quantitative analysis of the improvements, showing that the length of runs *strictly* decreases along $\to_{\mathsf{lh}}$. Note indeed that improvements guarantee only that the length of runs does not increase. To prove that it actually decreases one needs an additional *global* analysis of runs—improvements only deal with *local* bisimulation diagrams. On proof nets, this decreasing property correspond to the standard fact that IAM paths passing through a cut have shorter residuals after that cut.

We recall that we write $|t|$ for the length of the run $(\underline{t}, \langle\cdot\rangle, \epsilon, \epsilon)$, with the convention that $|t| = \infty$ if the machine diverges.

Lemma 3.7.2 (The length of terminating runs strictly decreases along $\to_{\mathsf{lh}}$). *Let $t \to_{\mathsf{lh}} u$ and $|t| \neq \infty$. Then, $|t| > |u|$.*

Proof. We treat the case of $t \to_{\mathsf{dB}} u$, the others are obtained via similar diagrams. If t has a $\to_{\mathsf{dB}}$-redex then it has the shape $t = H\langle\langle\lambda x.r\rangle Sw\rangle$ and u is in the form $u = H\langle\langle r[x\leftarrow w]\rangle S\rangle$. By induction on the structure of H one can prove that there exist $k, n \geq 0$ such that $(\underline{t}, \langle\cdot\rangle, \epsilon, \epsilon) \to^n_{\lambda\mathrm{IAM}} (\underline{\langle\lambda x.r\rangle Sw}, H, \epsilon, \bullet^k)$ and $(\underline{u}, \langle\cdot\rangle, \epsilon, \epsilon) \to^n_{\lambda\mathrm{IAM}} (\underline{\langle r[x\leftarrow w]\rangle S}, H, \epsilon, \bullet^k)$. By the definition of the improvement $\blacktriangleright_{\mathsf{dB}}$, we have the following diagram:

$$
\begin{array}{ccc}
(\underline{H\langle\langle\lambda x.r\rangle Sw\rangle}, \langle\cdot\rangle, \epsilon, \epsilon) & \blacktriangleright_{\mathsf{dB}} & (\underline{H\langle\langle r[x\leftarrow w]\rangle S\rangle}, \langle\cdot\rangle, \epsilon, \epsilon) \\
\downarrow^n & & \downarrow^n \\
(\underline{\langle\lambda x.r\rangle Sw}, H, \epsilon, \bullet^k) & \blacktriangleright_{\mathsf{dB}} & (\underline{\langle r[x\leftarrow w]\rangle S}, H, \epsilon, \bullet^k) \\
\downarrow & & \downarrow{\scriptstyle |S|} \\
(\underline{\langle\lambda x.r\rangle S}, H\langle\langle\cdot\rangle w\rangle, \epsilon, \bullet\cdot\bullet^k) & & (\underline{r[x\leftarrow w]}, H\langle S\rangle, \epsilon, \bullet^k) \\
\downarrow{\scriptstyle |S|} & & \downarrow \\
(\underline{\lambda x.r}, H\langle Sw\rangle, \epsilon, \bullet\cdot\bullet^k) & & \\
\downarrow & & \\
s_1 = (\underline{r}, H\langle\langle\lambda x.\langle\cdot\rangle\rangle Sw\rangle, \epsilon, \bullet^k) & \blacktriangleright_{\mathsf{dB}} & (\underline{r}, H\langle\langle\langle\cdot\rangle[x\leftarrow w]\rangle S\rangle, \epsilon, \bullet^k) = s_2
\end{array}
$$

From $s_1 \blacktriangleright_{\mathsf{dB}} s_2$, the hypothesis $|t| \neq \infty$, and the properties of improvements (Lemma 2), we obtain $|s_1| \geq |s_2|$. Then, we have $|t| = n + 1 + |S| + 1 + |s_1| > n + |S| + 1 + |s_2| = |u|$. □

Using the lemma, we prove the *if* direction of adequacy, that then follows.

Proposition 3.7.3 ($\to_{\mathsf{lh}}$-divergence implies that the λIAM diverges). *If t is a $\to_{\mathsf{lh}}$-divergent LSC term, then $[\![t]\!] = \bot$.*

Proof. By contradiction, suppose that $[\![t]\!] = \Downarrow$. Then, by definition, $|t| = n \in \mathbb{N}$ and it ends on a final state. Since t $\to_{\mathsf{lh}}$-divergent, then there exists an infinite reduction sequence $\rho : t = t_0 \to_{\mathsf{lh}} t_1 \to_{\mathsf{lh}} t_2 \to_{\mathsf{lh}} \cdots t_k \to_{\mathsf{lh}} \cdots$. Since the length of terminating runs strictly decreases along $\to_{\mathsf{lh}}$ (Lemma 3.7.2), for each $i \in \mathbb{N}$ if $t_i \to_{\mathsf{lh}} t_{i+1}$ then $|t_i| > |t_{i+1}|$. Then $|t_0| \geq |t_{n+1}| + n + 1$. Since the length of runs is non-negative, we obtain that $|t_0| \geq n + 1$, which is absurd because $|t_0| = |t| = n$. □

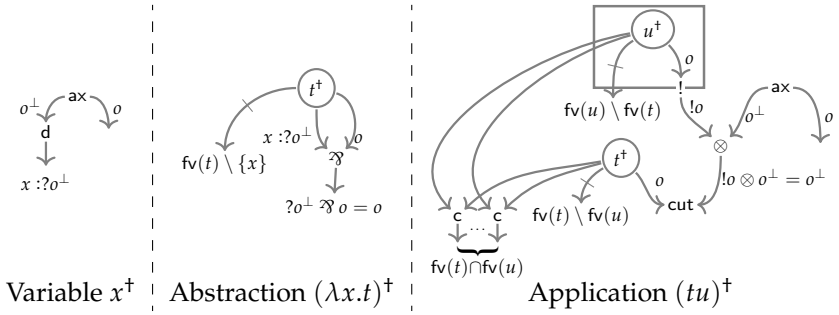


FIGURE 3.6: The call-by-name translation $(\cdot)^\dagger$ of the λ-calculus into linear logic proof nets.

Theorem 3.7.4 (Adequacy). *Let t be a LSC term. Then t has $\to_{\mathsf{lh}}$-normal form if and only if $[\![t]\!] = \Downarrow$.*

3.8 Comparison with Proof Nets

Here we sketch how the λIAM relates to the original presentation based on linear logic proof nets, due to Mackie (1995), Danos et al. (1996) and Danos and Regnier (1999), the IAM. Since this is not the main object of this thesis, we avoid defining proof nets and related concepts, and focus only on the key points.

The λIAM corresponds to the IAM on proof nets representing λ-terms according to the call-by-name translation $(\cdot)^\dagger$ in Fig. 3.6[13], and considering only paths from the distinguished conclusion of the net, as in (Danos et al., 1996) (while Mackie (1995) uses the call-by-value translation, and Danos and Regnier (1999) consider paths starting on whatever conclusions).

There is a bisimulation between the λIAM and such a restricted IAM, which is not strong because two λIAM transitions rather are *macros*, packing together whole sequences of transitions in their presentation. Namely, transition $\to_{\mathsf{var}}$ short-circuits the path between a variable x and its abstraction $\lambda x.C_n\langle x\rangle$. In proof nets, this path traverses a dereliction, exactly n auxiliary doors, possibly a contraction tree, and ends on the $\parr$ representing the abstraction. The dual transition $\to_{\mathsf{bt2}}$ does the reverse job, corresponding to the reversed path. Aside the different notations and the *macrification*, our transitions correspond exactly to the actions attached to proof net edges presented in (Danos and Regnier, 1999)[14], as we explain next.

In the proof nets presentation the token is given by two stacks, called *boxes stack* B and *balancing stack* S, corresponding exactly to our log L and tape T, respectively. They are formed by sequences of multiplicative constants p (corresponding to our $\bullet$) and by *exponential signatures* σ. They are defined by the following grammar[15].

$$\begin{array}{rrll} \textsc{Balancing stacks} & S & ::= \epsilon \mid \mathsf{p}\cdot S \mid \sigma\cdot S \\ \textsc{Boxes stacks} & B & ::= \epsilon \mid \sigma\cdot B \\ \textsc{Exp. signatures} & \sigma,\sigma' & ::= \square \mid \langle\sigma,\sigma'\rangle \mid \langle \mathsf{l},\sigma\rangle \mid \langle \mathsf{r},\sigma\rangle \end{array}$$

Intuitively, exponential signatures are binary trees with $\square$, l or r as leaves, where l and r denote the left/right premise of a contraction. Fig. 3.7 shows the IAM transitions concerning exponential signatures that are the relevant difference with respect to the λIAM.

To explain how $\to_{\mathsf{var}}$ is simulated by the IAM, let us recall it:

[13] The translation uses a recursive type $o = ?o^\perp \parr o$ in order to be able to represent untyped terms of the λ-calculus—this is standard. Every net has a unique conclusion labeled with o, which is the *output*, and all the other conclusions have type $?o^\perp$ and are labeled with a free variable of the term. In the abstraction case $\lambda x.t$, if $x \notin \mathsf{fv}(t)$ then a weakening is added to represent that variable.

[14] We refer to (Danos and Regnier, 1999) rather than (Danos et al., 1996) because in (Danos et al., 1996) the definition is only sketched, while (Danos and Regnier, 1999) is more accurate.

[15] With respect to (Danos and Regnier, 1999): for clarity, we use symbols l and r instead of p' and q', and we omit q, dual of p, as in the call-by-name translation it is always and only next to exponential signatures, which then subsume it.

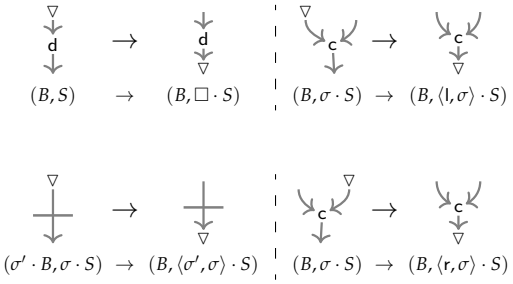


FIGURE 3.7: The transition rules of the proof nets presentation of the IAM related to exponential signatures.

$$(x, C\langle \lambda x.D_n \rangle, L_n \cdot L, T) \to_{\mathsf{var}} (\lambda x.D_n\langle x \rangle, C, L, (x, \lambda x.D_n, L_n) \cdot T).$$

The IAM does the same, just in more steps and with another syntax. Consider a token $(B_n \cdot B, S)$ approaching a variable x that is n boxes deeper than its binder $\lambda x.D_n\langle x \rangle$. Variables are translated as dereliction links and thus we have: $(B_n \cdot B, S) \to (B_n \cdot B, \square \cdot S)$.

Then, the token travels until the binder of x is found (a $\parr$ in the proof net translation of the term), *i.e.* it traverses exactly n boxes always exiting from the auxiliary doors. Moreover, for every such box a contraction could be encountered. Let's first suppose that x is used linearly, so that no contractions are encountered. Then the token rewrites in the following way, traversing n auxiliary doors.

$$\begin{aligned}(\sigma_1 \cdot B_{n-1} \cdot B, \square \cdot S) &\to (\sigma_2 \cdot B_{n-2} \cdot B, \langle \sigma_1, \square \rangle \cdot S) \\ &\to (\sigma_3 \cdot B_{n-3} \cdot B, \langle \sigma_2, \langle \sigma_1, \square \rangle \rangle \cdot S) \\ &\to \cdots \to (B, \langle \sigma_n, \langle \cdots \langle \sigma_1, \square \rangle \cdots \rangle \rangle \cdot S).\end{aligned}$$

Note the perfect matching between the two formulations: in both cases the first n logged positions/signatures in the log/boxes stack are removed from it and, once wrapped in a single logged position/signature, then put on the tape/balancing stack. In presence of contractions the exponential signature $\langle \sigma_n, \langle \cdots \langle \sigma_1, \square \rangle \cdots \rangle \rangle$ is interleaved by l and r leaves. These symbols represent nothing more than a binary code used to traverse the contraction tree of x. In the λIAM, the same information is represented, more compactly, by specifying the variable occurrence via its position inside its binder.

Chapter 4

Sequence Types Capture IAM Time

Here we introduce the type system that we shall use to measure the length of λIAM runs. We shall give an abstract type-theoretic notion of time, strongly linked to linear logic via non-idempotent intersection types. Moreover, we recover the λIAM as a machine acting directly on type derivations, this way showing its *canonicity*, and linking two apparently distinct worlds.

4.1 Sequence (Intersection) Types

Intersections, Multi Sets, and Sequences. The framework that we adopt is the one of intersection types. As in many recent works, we use the non-idempotent variant, where the intersection type $A \wedge A$ is not equivalent to A, and which has stronger ties to linear logic and time analyses, because it takes into account how many times a resource/type A is used, and not just whether A is used or not. We recall that non-idempotent intersections are multi sets, which is why these types are sometimes called *multi types* and an intersection $A \wedge B \wedge A$ is rather noted $[A, B, A]$. Here we add a further change, we also consider *non-commutative* multi types. Removing commutativity turns multi sets into lists, or sequences—thus, we call them *sequence types*. Adopting sequences is an inessential tweak. Our study does not really depend on their sequential structure, we only need to use bijections between multi sets, to describe the SIAM, and these bijections are just more easily managed using sequences rather than multi sets. This *rigid* approach has been already used fruitfully by Tsukada et al. (2017) and Mazza et al. (2018).

Basic Definitions. As for multi types, there are two layers of types, *linear types* and *sequence types*, mutually defined as follows.

$$\begin{array}{rrcl} \text{LINEAR TYPES} & A, B & ::= & \star \mid S \to A \\ \text{SEQUENCE TYPES} & S, S' & ::= & [A_1, \dots, A_n] \end{array}$$

Since commutativity is ruled out, we have, e.g., $[A, B] \neq [B, A]$. We shall use $[\cdot]$ as a generic list constructor not limited to types, thus writing $[2, 1, 12, 4]$ for a list of natural numbers, and also use it for lists of judgments or type derivations. Note that there is a ground type $\star$, which can be thought as the type of normal forms, that in Closed CbN are precisely abstractions. Note also that arrow (linear) types $S \to A$ can have a sequence only on the left. The empty sequence is noted $[\cdot]$, and the concatenation of two sequences S and S' is noted $S \uplus S'$.

Type judgments have the form $\Gamma \vdash t : A$, where Γ is a type environment, defined below. The typing rules are in Fig. 4.1, type derivations are noted π and we write $\pi \triangleright \Gamma \vdash t : A$ for a type derivation π of ending judgment $\Gamma \vdash t : A$. Type environments, ranged over by Γ, Δ are total maps from variables to sequence types such that only finitely many variables are mapped to non-empty sequence types, and we write $\Gamma = x_1 : S_1, \dots, x_n : S_n$ if $dom(\Gamma) = \{x_1, \dots, x_n\}$—note that type environments are commutative, what is non-commutative is only the sequence constructor $[\cdot]$. Given two type environments Γ, Δ, the expression $\Gamma \uplus \Delta$ stands for the type environment assigning to every variable x the list $\Gamma(x) \uplus \Delta(x)$.

In the following we use two basic properties of the type system, collected in the following straightforward lemma. One is the absence of weakening, and the other one is a correspondence between sequence types and axioms. We write $|S|$ for the length of S as a sequence.

$$\frac{}{x : [A] \vdash x : A}\ \text{T-Var} \qquad \frac{\Gamma, x : S \vdash t : A}{\Gamma \vdash \lambda x.t : S \to A}\ \text{T-}\lambda \qquad \frac{}{\vdash \lambda x.t : \star}\ \text{T-}\lambda_\star$$

$$\frac{\Gamma \vdash t : [B_1, \dots, B_n] \to A \quad [\Delta_i \vdash u : B_i]_{i \in [1,\dots,n]}}{\Gamma \uplus \biguplus_{i \in [1,\dots,n]} \Delta_i \vdash tu : A}\ \text{T-@}$$

FIGURE 4.1: The sequence type system.

Lemma 4.1.1 (Relevance and axiom sequences). *If $\pi \rhd \Gamma \vdash t : A$ then $dom(\Gamma) \subseteq \mathsf{fv}(t)$, thus if t is closed then Γ is empty. Moreover, there are exactly $|\Gamma(x)|$ axioms typing x in π, which appear from left to right as leaves of π (seen as an ordered tree) in the order given by $\Gamma(x) = [A_1, \dots, A_k]$ and that the i-th axiom types x with A_i.*

Characterization of Termination. It is well-known that intersection and multi types characterize Closed CbN termination, that is, they type *all* and only those λ-terms that terminate with respect to weak head reduction. If terms are closed, the same result smoothly holds for sequence types, as we now explain. The only point where non-commutativity is delicate for the characterization is in the proof of the typed substitution lemma for subject reduction (and the dual lemma for subject expansion), as substitution may change the order of concatenation in type environments. In our simple setting where terms are closed, however, the term to substitute is closed[1] and—by the relevance lemma—its type derivation comes with no type environment, so the order-of-concatenation problem disappears. Therefore, sequence types characterize termination in Closed CbN too. Thus from now on we essentially identify multi and sequence types.

Theorem 4.1.2. *A closed term t has weak head normal form if and only if $\vdash t : \star$.*

Sequence Types and λIAM Time. We would like to extend de Carvalho (2007) result, presented in Chapter 2, that links (the size of) multi type derivations to the length of KAM runs, to the case of the IAM. The way in which the IAM computes is completely different from how the KAM evaluation mechanism, and in particular not a priori linked to the normalization of multi type proofs. This way, while keeping the same type system, we have to change the weight assignment to typing rules. We anticipate that the intuition is that there is a bijection between occurrences of the ground type $\star$ in a derivation for a term t and states in the λIAM run for t. This is why we have to count the number of occurrences of $\star$ in a type derivation. First, we have to define a norm on types and sequence types, counting the number of occurrences of $\star$:

$$\|\star\| := 1 \qquad \|S \to A\| := \|S\| + \|A\| \qquad \big\|[A_1, \dots, A_n]\big\| := \sum_{1 \le i \le n} \|A_i\|$$

Then we define the weight system $\mathbf{W}_{\lambda\text{IAM}}(\cdot)$ in Fig. 4.2. Observe how this weight system is structurally very similar to $\mathbf{W}_{KAM}(\cdot)$ of Fig. 2.3, the only difference being the fact that whenever the latter adds 1 to the weight, thus counting the size of the derivation, the former adds the number of occurrences of $\star$ in the underlying type, thus counting the total number of $\star$ occurrences. The next section proves the following theorem, that is the λIAM analogous of de Carvalho's theorem.

Theorem 4.1.3. *There is a complete λIAM run of length n from t if and only if exists π such that $\pi \rhd \vdash t : \star$ and $\mathbf{W}_{\lambda IAM}(\pi) = n$.*

[1] It is well known that in Closed CbN the substitutions $t\{x \leftarrow u\}$ associated to reduced β-redexes are such that u is closed. The term t is of course (potentially) open, and its type derivation has a type environment Γ, but the important point here is that the type derivation of u has no type environment, so that the substitution does not concatenate sequence types.

$$\frac{}{x : [A] \overset{\|A\|}{\vdash} x : A}\ \text{T-VAR} \qquad \frac{\Gamma, x : S \overset{w}{\vdash} t : A}{\Gamma \overset{w+\|S\to A\|}{\vdash} \lambda x.t : S \to A}\ \text{T-}\lambda \qquad \frac{}{\overset{0}{\vdash} \lambda x.t : \star}\ \text{T-}\lambda_\star$$

$$\frac{\Gamma \overset{w}{\vdash} t : [B_1, \dots, B_n] \to A \quad [\Delta_i \overset{v_i}{\vdash} u : B_i]_{i\in[1,\dots,n]}}{\Gamma \uplus \biguplus_{i\in[1,\dots,n]} \Delta_i \overset{w+\sum v_i+\|A\|}{\vdash} tu : A}\ \text{T-@}$$

FIGURE 4.2: The weight assignment $\mathbf{W}_{\lambda\text{IAM}}(\cdot)$.

4.2 The Sequence IAM

This section introduces yet another machine, the *Sequence IAM*, or *SIAM*, that mimics the λIAM directly on top of a type derivation π. It is the key tool used in the next section to show that the λIAM weights on type derivations do measure the time cost of λIAM runs.

The SIAM. The idea behind the SIAM is simple but a formal definition is a technical nightmare. Let us explain the idea. The machine moves over a fixed type derivation $\pi \triangleright \vdash t : \star$, to be thought as the code. The position of the machine is expressed by an occurrence of a type judgment[2] J of π. As the λIAM, the SIAM has two possible directions, noted $\downarrow$ and $\uparrow$[3]. In direction $\uparrow$ the machine looks at the rule above the focused judgment, in direction $\downarrow$ at the rule below. The only "data structure" is a type context $\mathbb{A}$ isolating an occurrence of $\star$ in the type A of the focused judgment (occurrence) $\Gamma \vdash u : A$, defined as follows (careful to not confuse type contexts $\mathbb{A}$ with type environments Γ):

$$\begin{array}{rrcl} \text{TYPE CTXS} & \mathbb{A} & ::= & \langle\cdot\rangle \mid S \to \mathbb{A} \mid \mathbb{S} \to A \\ \text{SEQUENCE CTXS} & \mathbb{S} & ::= & [A_1, .., A_k, \mathbb{A}, A_{k+1}.., A_n] \end{array}$$

Summing up, a state s is a quadruple $(\pi, J, \mathbb{A}, d)$. If J is in the form $\Gamma \vdash u : A$, we often write s as $\vdash u : \mathbb{A}\langle\star_d\rangle$, where $\mathbb{A}\langle\star\rangle = A$. In fact we shall see that type environments play no role.

Transitions. The SIAM starts on the final judgment of π, with empty type context $\mathbb{A} = \langle\cdot\rangle$ and direction $\uparrow$. It moves from judgment to judgment, following occurrences of $\star$ around π. The transitions are in Fig. 4.3, their union noted $\to_{\text{SIAM}}$, as we now explain them—the transitions have the labels of λIAM transitions, because they correspond to each other, as we shall show.

Let's start with the simplest, $\to_{\bullet 2}$. The state focuses on the conclusion judgment J of a T-λ rule with direction $\uparrow$. The eventual type environment Γ is omitted because the transition does not depend on it—none of the transitions does, so type environments are omitted from all transitions. The judgment assigns type $S \to A$ to $\lambda x.t$, and the type context is $S \to \mathbb{A}$, that is, it selects an occurrence of $\star$ in the target type $A = \mathbb{A}\langle\star\rangle$. The transition then simply moves to the judgment above, stripping down the type context to $\mathbb{A}$, and keeping the same direction. Transition $\bullet 4$ does the opposite move, in direction $\downarrow$, and transitions $\bullet 1$ and $\bullet 3$ behave similarly on T-@ rules: $[\vdash]$ simply denotes the right premise that is left unspecified since not relevant to the transition.

Transitions $\to_{\mathsf{arg}}$: the focus is on the left premise of a T-@ rule, of type $S \to B$ isolating $\star$ inside the i-th type A_i in S. The transition then moves to the final judgment of the i-th derivation in the right premise, changing direction. Transition $\to_{\mathsf{bt1}}$ does the opposite move.

Transitions $\to_{\mathsf{var}}$ and $\to_{\mathsf{bt2}}$ are based on the axiom sequences property of Lemma 4.1.1. Consider a T-λ rule occurrence whose right-hand type of the conclusion is $S \to B$. The premise has shape $\Gamma, x : S \vdash t : B$, and by the lemma there is a bijection between the sequence of linear types in S and the axioms on x, respecting the order in S. The left side of $\to_{\mathsf{bt2}}$ focuses on the i-th type A_i in S

[2] A judgment may occur repeatedly in a derivation, which is why we talk about *occurrences* of judgments. To avoid too many technicalities, we usually just write the judgment, leaving implicit that we refer to an occurrence of that judgment.

[3] Type derivations are upside-down wrt to the term structure, then direction $\downarrow$ of the λIAM becomes here $\uparrow$, and $\uparrow$ is $\downarrow$. The color code remains the same, of course.

$$\frac{\vdash t : S \to A \quad [\vdash]}{\vdash tu : \mathbb{A}\langle\star_{\uparrow}\rangle} \;\to_{\bullet 1}\; \frac{\vdash t : S \to \mathbb{A}\langle\star_{\uparrow}\rangle \quad [\vdash]}{\vdash tu : A}$$

$$\frac{\vdash t : A}{\vdash \lambda x.t : S \to \mathbb{A}\langle\star_{\uparrow}\rangle} \;\to_{\bullet 2}\; \frac{\vdash t : \mathbb{A}\langle\star_{\uparrow}\rangle}{\vdash \lambda x.t : S \to A}$$

$$\frac{\vdash t : S \to \mathbb{A}\langle\star_{\downarrow}\rangle \quad [\vdash]}{\vdash tu : A} \;\to_{\bullet 3}\; \frac{\vdash t : S \to A \quad [\vdash]}{\vdash tu : \mathbb{A}\langle\star_{\downarrow}\rangle}$$

$$\frac{\vdash t : \mathbb{A}\langle\star_{\downarrow}\rangle}{\vdash \lambda x.t : S \to A} \;\to_{\bullet 4}\; \frac{\vdash t : A}{\vdash \lambda x.t : S \to \mathbb{A}\langle\star_{\downarrow}\rangle}$$

$$\frac{\dfrac{}{\vdash x : \mathbb{A}\langle\star_{\uparrow}\rangle_i}\,i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : [...A_i...] \to B}{} \;\to_{\mathsf{var}}\; \frac{\dfrac{}{\vdash x : A_i}\,i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : [...\mathbb{A}\langle\star_{\downarrow}\rangle_i...] \to B}{}$$

$$\frac{\dfrac{}{\vdash x : A_i}\,i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : [...\mathbb{A}\langle\star_{\uparrow}\rangle_i...] \to B}{} \;\to_{\mathsf{bt2}}\; \frac{\dfrac{}{\vdash x : \mathbb{A}\langle\star_{\downarrow}\rangle_i}\,i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : [...A_i...] \to B}{}$$

$$\frac{\vdash t : [...\mathbb{A}\langle\star_{\downarrow}\rangle_i...] \to B \quad \vdash_i u : A_i}{\vdash tu : B} \;\to_{\mathsf{arg}}\; \frac{\vdash t : [...A_i...] \to B \quad \vdash_i u : \mathbb{A}\langle\star_{\uparrow}\rangle_i}{\vdash tu : B}$$

$$\frac{\vdash t : [...A_i...] \to B \quad \vdash_i u : \mathbb{A}\langle\star_{\downarrow}\rangle_i}{\vdash tu : B} \;\to_{\mathsf{bt1}}\; \frac{\vdash t : [...\mathbb{A}\langle\star_{\uparrow}\rangle_i...] \to B \quad \vdash_i u : A_i}{\vdash tu : B}$$

FIGURE 4.3: The transitions of the Sequence IAM (SIAM).

and the SIAM moves to the judgment of the axiom corresponding to that type, which is exactly the i-th from left to right seeing the derivation as a tree where the children of nodes are ordered as in the typing rules. Transition $\rightarrow_{\mathsf{var}}$ does the opposite move, which can always happen because the code is the type derivation of a closed term.

The only typing rule not inducing a transition is T-$\lambda_\star$. Accordingly, when the SIAM reaches one of these rules it is in a final state. Exactly as the λIAM, the SIAM is bi-deterministic.

Proposition 4.2.1. *The SIAM is bi-deterministic for each type derivation $\pi \triangleright \vdash t : \star$.*

An Example. We present below the very same example analyzed in Section 3.1. We have reported its type derivation, with the occurrences of $\star$ on the right of $\vdash$ annotated with increasing integers and a direction. The occurrence of $\star$ marked with 1 represents the first state, and so on.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : [[\star] \to \star] \vdash x : [\star_{\downarrow 16}] \to \star_{\uparrow 6}} \quad \dfrac{}{y : [\star] \vdash y : \star_{\uparrow 17}}}{y : [\star], x : [[\star] \to \star] \vdash xy : \star_{\uparrow 5}}}{y : [\star] \vdash \lambda x.xy : [[\star_{\uparrow 15}] \to \star_{\downarrow 7}] \to \star_{\uparrow 4}}}{\vdash \lambda y.\lambda x.xy : [\star_{\downarrow 18}] \to [[\star_{\uparrow 14}] \to \star_{\downarrow 8}] \to \star_{\uparrow 3}} \quad \dfrac{}{\vdash \mathsf{I} : \star_{\uparrow 19}}}{\vdash (\lambda y.\lambda x.xy)\mathsf{I} : [[\star_{\uparrow 13}] \to \star_{\downarrow 9}] \to \star_{\uparrow 2}} \quad \dfrac{\dfrac{}{z : [\star] \vdash z : \star_{\uparrow 11}}}{\vdash \lambda z.z : [\star_{\downarrow 12}] \to \star_{\uparrow 10}}}{\vdash (\lambda y.\lambda x.xy)\mathsf{I}(\lambda z.z) : \star_{\uparrow 1}}$$

One can immediately notice that every occurrence of $\star$ is visited exactly once. Moreover, the sequence of the visited subterms is the same as the one obtained in the example of Section 3.1.

4.3 Measuring the Time of Interaction

The aim of this section is to explain the strong bisimulation between the SIAM and the λIAM, that, once again, is based on a variant on the exhaustible invariant. A striking point of the SIAM is that it does not have the log nor the tape. They are encoded in the judgment occurrence J and in the type context $\mathbb{A}$ of its states, as we shall show. But first, let's make a step back.

Handling Duplication. β-reducing a λ-term (potentially) duplicates arguments, whose different copies may be used differently, typically being applied to different further arguments. The machines in this thesis never duplicate arguments, but have nonetheless to distinguish different uses of a same piece of code. This is why the λIAM uses *logged* positions instead of simple positions: the log is a trace of (part of) the previous run that allows to distinguishing different uses of the position—the closures of the KAM or the history mechanism of the λPAM are alternatives.

The key point of multi/sequence type derivations is that duplication is explicitly accounted for, somewhat *in advance*, by multi-set/sequences: all arguments come with as many type derivations as the times they are duplicated during evaluation. Note indeed that the type derivation may be way bigger than the term itself, while this is not possible with, say, simple types. Therefore, there is no need to resort to logs, closures, or histories to distinguish copies, because all copies are already there: simple positions in the type derivation (not in the term!) are informative enough.

Relating Logs and Tapes with Typed Positions. In the λIAM, the log $L = l_1 \cdot \ldots \cdot l_n$ has a logged position for every argument $u_1, \ldots, u_n$ in which the position of the current state is contained. The argument u_i is the answer to the query of an argument for the variable in the logged position l_i. The SIAM does not keep a trace of the variables for which it completed a query, but the answers to those (forgotten) queries are simply given by the sub-derivations for $u_1, \ldots, u_n$ in which the current judgment occurrence J is contained—the way in which l_k identifies a copy of u_k in the λIAM corresponds on the type derivation π to the index i of the sub-derivation (in the sequence of sub-derivations) typing u_k in which J is located. Note that the λIAM manipulates the log only via transitions $\rightarrow_{\mathsf{arg}}$ and $\rightarrow_{\mathsf{bt1}}$, that on the SIAM correspond exactly to entering/exiting derivations for arguments. The tape, instead, contains logged positions for which the λIAM either has not yet

found the associated argument, or it is backtracking to. Note that the λIAM puts logged positions on the tape via transitions $\to_{\mathsf{var}}$ and $\to_{\mathsf{bt1}}$, and removes them using $\to_{\mathsf{arg}}$ and $\to_{\mathsf{bt2}}$. By looking at Fig. 4.2, it is evident that there is a logged position on the λIAM tape for every type sequence S in which it lies the hole $\langle\cdot\rangle$ of the current type context $\mathbb{A}$ of the SIAM.

These ideas are used to extract from every SIAM state s a λIAM state $\mathsf{ext}(s)$ in a quite technical way. A notable point is that the extraction procedure is formally defined by means of yet another reformulation on the SIAM of the exhaustible invariant, called *S-exhaustibility*, relying on tests induced by a SIAM state built following the explained correspondence. The extraction process induces a relation $s \simeq_{\mathsf{ext}} \mathsf{ext}(s)$ that is easily proved to be a strong bisimulation between the SIAM and the λIAM.

4.3.1 S-Exhaustible Invariant

In this section we prove the S-exhaustible state invariant for the SIAM, then use it to extract λIAM states from SIAM ones, and finally prove the strong bisimulation between the two machines.

We start by defining the notions of typed tests used to define S-exhaustible states.

Type Positions and Generalized States. To define tests, we have to consider a slightly more general notion of SIAM state. In Sect. 4.2, a state is a quadruple $(\pi, J, \mathbb{A}, d)$ where J is an occurrence of a judgment $\Gamma \vdash u : A$ in π, d is a direction, and $\mathbb{A}$ is a type context isolating an occurrence of $\star$ in A. The generalization simply is to consider type contexts $\mathbb{A}$ such that $\mathbb{A}\langle B\rangle = A$ for some B, that is, not necessarily isolating $\star$. A pair $(B, \mathbb{A})$ such that $\mathbb{A}\langle B\rangle = A$ is called a position in A.

Note that the SIAM can be naturally adapted to this more general notion of state, that follows an arbitrary formula B, not necessarily $\star$ —it can be found in Fig. 4.4, and it amounts to simply replacing $\star$ with B.

To easily manage SIAM states we also use a concise notations, writing $\vdash t : A, \mathbb{A}$ for a state $s = (\pi, J, (A, \mathbb{A}), d)$ where J is $\Gamma \vdash t : \mathbb{A}\langle A\rangle$ for some Γ, potentially specifying the direction via colors and under/over-lining.

SIAM Tests. Given a SIAM state $s = (\pi, J, (A, \mathbb{A}), d)$, the underlying idea is that the judgment occurrence J encodes the log of the λIAM, while the type context $\mathbb{A}$ encodes the tape. It is then natural to define two kinds of test, one for judgments and one for type contexts.

The intuition is that a test focuses on (the occurrence of) an element B of a sequence S related to s, and that these sequence elements play the role of logged positions in the λIAM. These sequence elements are of two kinds:

1. *Elements containing J*: those in which the focused judgment J itself is contained, corresponding to the logged positions in the log of the λIAM. Note that the positions on the log are those for which the λIAM has previously found the corresponding arguments. In the SIAM these arguments are exactly those in which the focused judgment is contained.

2. *Elements appearing in $\mathbb{A}$*: those in the right-hand type of s in which the focused type A is contained, corresponding to the logged positions on the tape of the λIAM. They correspond to λIAM queries for which the argument has not yet been found, or positions to which the λIAM is backtracking to.

Each one of these elements is then identified by a judgment occurrence π' and a position $(B, \mathbb{A}')$ in the right-hand type of π'.

Definition 4.3.1 (Focus). *A* focus f *in a derivation π is a pair $f = (J, (A, \mathbb{A}))$ of a judgment occurrence J and of a type position $(A, \mathbb{A})$ in the right-hand type $\mathbb{A}\langle A\rangle$ of J.*

The intuition is that exhausting a test $s_{J,(A,\mathbb{A})}$ in π shall amount to retrieving the axiom of π of type A that would be substituted by that sequence element of type A by reducing π via cut-elimination—the definition of exhaustible tests is given below, after the definition of tests.

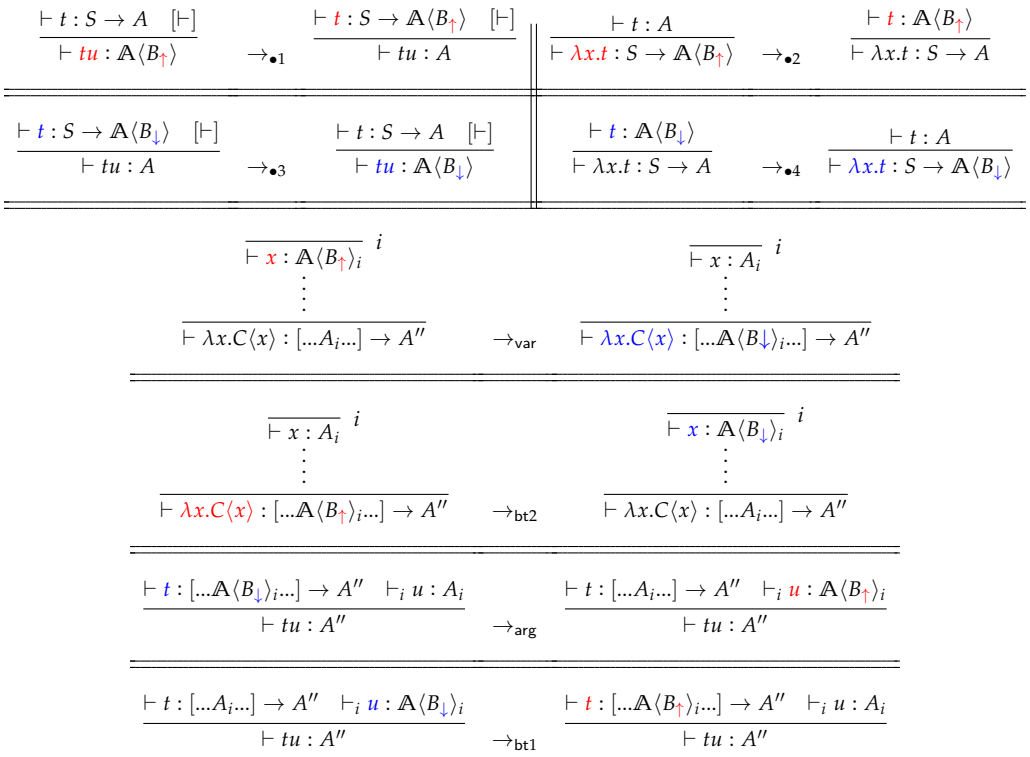


FIGURE 4.4: The transitions of the (Generalized) Sequence IAM (SIAM).

Definition 4.3.2 (judgment tests). *Let $s = (\pi, J, (A, \mathbb{A}), d)$ be a SIAM state. Let r_i be i-th* T-@ *rule found traversing π by descending from the focused judgment J towards the final judgment of π. Let J_i be the judgment of the sequence S_i in the right premise of r_i traversed in such a descent (careful: J_i is the j-th judgment of S_i for some j, that is, the index i denotes the connection with rule r_i, not the position in S_i). Let J_i be $\Gamma \vdash t : B$. Then $s^i_f = (\pi, J_i, (B, \langle\cdot\rangle), \downarrow)$ is the i-th judgment test of s, having as focus $f := (J_i, (B, \langle\cdot\rangle))$.*

We often omit the judgment from the focus, writing simply $s_{(B,\langle\cdot\rangle)}$, and even concisely note s_f as $\vdash t : B, \langle\cdot\rangle_{\downarrow}$. Note that judgment tests always have type context $\langle\cdot\rangle$. According to the intended correspondence judgment/ log and type context/tape between the SIAM and the λIAM, having type context $\langle\cdot\rangle$ corresponds to the fact that the log tests of the λIAM have an empty tape.

Example 4.3.3 (judgment test).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : [[\star] \to \star] \vdash x : [\star_{\downarrow 16}] \to \star_{\uparrow 6}} \quad \dfrac{}{y : [\star] \vdash y : \star_{\uparrow 17}}}{y : [\star], x : [[\star] \to \star] \vdash xy : \star_{\uparrow 5}}}{y : [\star] \vdash \lambda x.xy : [[\star_{\uparrow 15}] \to \star_{\downarrow 7}] \to \star_{\uparrow 4}}}{\vdash \lambda y.\lambda x.xy : [\star_{\downarrow 18}] \to [[\star_{\uparrow 14}] \to \star_{\downarrow 8}] \to \star_{\uparrow 3}} \quad \dfrac{}{\vdash \mathsf{I} : \star_{\uparrow 19}}}{\vdash (\lambda y.\lambda x.xy)\mathsf{I} : [[\star_{\uparrow 13}] \to \star_{\downarrow 9}] \to \star_{\uparrow 2}} \quad \dfrac{}{z : [\star] \vdash z : \star_{\uparrow 11}}}{\vdash (\lambda y.\lambda x.xy)\mathsf{I}(\lambda z.z) : \star_{\uparrow 1}}$$

Let us give an example of judgment test in the context of the given example of SIAM run (which we have reported here, for readability reasons). If we consider the state $\uparrow 11$, we find its log tests going down in the type derivation for each T-@ *rule traversed from the right hand side. In this case we immediately find the judgment $\vdash \lambda z.z : [\star] \to \star$. Then, $\vdash \lambda z.z : \langle[\star] \to \star\rangle_{\downarrow}$ is a log test for $\uparrow 11$. Since between $\vdash \lambda z.z : [\star] \to \star$ and the root of the derivation we do not cross any other suitable* T-@ *rule, there are no other log tests for $\uparrow 11$.*

Type (Context) Tests. While judgment tests depend only on the judgment occurrence J of a state $s = (\pi, J, (A, \mathbb{A}), d)$, type context tests—dually—fix J and depend only on the type context $\mathbb{A}$ of s, that is, they all focus on sequence elements of the form $(J, (B, \mathbb{A}'))$ where $\mathbb{A}'\langle B\rangle = \mathbb{A}\langle A\rangle$ and $\mathbb{A} = \mathbb{A}'\langle\mathbb{A}''\rangle$ for some type context $\mathbb{A}''$. Namely, there is one type context test (shortened to *type test*) for every sequence in which the hole of $\mathbb{A}$ is contained. We need some notions about type contexts, in particular a notion of level analogous to the one for term contexts.

Terminology About Type Contexts. Define type contexts $\mathbb{A}_n$ of level $n \in \mathbb{N}$ as follows:

$$\begin{array}{lcl} \mathbb{A}_0 & := & \langle\cdot\rangle \mid S \to \mathbb{A}_0 \\ \mathbb{A}_{n+1} & := & [...\mathbb{A}_n...] \to A \mid S \to \mathbb{A}_{n+1} \end{array}$$

Clearly, every type context $\mathbb{A}$ can be seen as a type context $\mathbb{A}_n$ for a unique n, and viceversa a type context of level n is also simply a type context—the level is then sometimes omitted. A *prefix* of a context $\mathbb{A}$ is a context $\mathbb{A}'$ such that $\mathbb{A}'\langle\mathbb{A}''\rangle = \mathbb{A}$ for some $\mathbb{A}''$. Given $\mathbb{A}$ of level $n > 0$, there is a smallest prefix context $\mathbb{A}|_i$ of level $0 < i \leq n$, and it has the form $\mathbb{A}'\langle[...\langle\cdot\rangle...] \to A\rangle$ for a type context $\mathbb{A}'$ of level $i - 1$.

Definition 4.3.4 (Type tests). *Let $s = (\pi, J, (A, \mathbb{A}), d)$ be a SIAM state and n be the level of $\mathbb{A}$. The sequence of directed prefixes $\mathsf{DiPref}(\mathbb{A})$ of $\mathbb{A}$ is the sequence of pairs $(\mathbb{A}', d')$, where $\mathbb{A}'$ is a prefix of $\mathbb{A}$, defined as follows:*

$$\begin{array}{lcll} \mathsf{DiPref}(\mathbb{A}) & := & [\cdot] & \text{if } n{=}0 \\ \mathsf{DiPref}(\mathbb{A}) & := & [(\mathbb{A}|_1, \uparrow), \ldots, (\mathbb{A}|_n, \uparrow^{n-1})] & \text{if } n{>}0 \end{array}$$

The i-th directed prefix (from left to right) $(\mathbb{A}', d')$ in $\mathsf{DiPref}(\mathbb{A})$ induces the type test $s^i_f := (\pi, J, (\mathbb{A}''\langle A\rangle, \mathbb{A}'), d')$ of s and focus $f := (J, (\mathbb{A}''\langle B\rangle, \mathbb{A}'))$, where $\mathbb{A}''$ is the unique type context such that $\mathbb{A} = \mathbb{A}'\langle\mathbb{A}''\rangle$.

According to the idea that type tests correspond to the tape tests of the λIAM, note that the first element (on the left) of the sequence $\mathsf{DiPref}(\mathbb{A})$ has $\uparrow$ direction, and that the direction alternates along the sequence. This is the analogous to the fact that the tape test associated to the first logged position on the tape (from left to right) has always direction $\downarrow$, and passing to the test of the next logged position on the tape switches the direction.

Example 4.3.5 (Type test). *Let us now give examples of type tests in the example of SIAM run that we provided. We compute the tape tests of $\uparrow_{13}$. Its type is*

$$[[\langle\star\rangle] \to \star] \to \star$$

with respect to the notation of the previous definition, we have $A = \star$ and $\mathbb{A} = [[\langle\cdot\rangle] \to \star] \to \star$. The level of $\mathbb{A}$ is 2. Tape tests are associated with the pairs in $\mathsf{DiPref}([[\langle\cdot\rangle] \to \star] \to \star)$, namely $[([\langle\cdot\rangle] \to \star, \uparrow), ([[\langle\cdot\rangle] \to \star] \to \star, \downarrow)]$.

Definition 4.3.6 (State respecting a focus). *Let $f = (J, (A, \mathbb{A}))$ be a focus. A SIAM state s respects f if it is an axiom $\vdash x : \langle A\rangle_\downarrow$ for some variable x (the typing context of s, which is omitted by convention, is $x : A$).*

Definition 4.3.7 (S-Exhaustible states). *The set $\mathcal{E}_S$ of S-exhaustible states is the smallest set such that if $s \in \mathcal{E}_S$, then for each type or judgment test of s_f of focus f there exists a run $\rho : s_f \to^*_{\text{SIAM}} \to_{\mathsf{bt2}} s'$ where s' respects f and for the shortest such run $s' \in \mathcal{E}_S$.*

Lemma 4.3.8 (S-exhaustible invariant). *Let t be a closed term, $\pi \triangleright \Gamma \vdash t : A$ a sequence type derivation for it, and $\rho : \vdash t : \langle A\rangle_\uparrow \to^k_{\text{SIAM}} s$ an initial SIAM run. Then s is S-exhaustible.*

The proof, long but conceptually simple, is in the Appendix.

4.3.2 The Extraction Process, and the λIAM/SIAM Strong Bisimulation

From S-exhaustible states one is able to *extract* λIAM states, as the following definition shows. Please note that the definition is well-founded, precisely because the objects are S-exhaustible states. Indeed, the induction principle used to define S-exhaustability allows recursive definition on S-exhaustible states to be well-behaved.

Definition 4.3.9 (Extraction of logged positions)**.** *Let s be an S-exhaustible SIAM state in a derivation π, t be the final term in π, and s_f be a judgment or type test of s. Since s is S-exhaustible, there is an exhausting run $s_f \to^{+}_{\mathrm{SIAM}} s' \in \mathcal{E}_S$. Let x be the variable of s'. Then the logged position extracted from s_f is $l_{\mathsf{ext}}(s_f) := (x, \lambda x.D_n, l_{\mathsf{ext}}(s'^1)..l_{\mathsf{ext}}(s'^n))$, where D_n is the context (of level n) retrieved traversing π from s' to the binder of λx of x in t and s'^i is the i-th judgment test of s'.*

Definition 4.3.10 (Extraction of logs, tapes, and states)**.** *Let $s = (\pi, J, (A, \mathbb{A}), d)$ be an S-exhaustible SIAM state where t is the final term in π, and J is $\Gamma \vdash u : \mathbb{A}\langle A\rangle$. The λIAM state extracted from s is $s_{\mathsf{ext}}(s) := (u, C_s, L_{\mathsf{ext}}(s), T_{\mathsf{ext}}(s), d)$ where*

- Context*: C_s is the only term context such that $t = C_s\langle u\rangle$;*
- Log*: $L_{\mathsf{ext}}(s) := l_1 \cdots l_i \cdots l_n$ where $l_i = l_{\mathsf{ext}}(s^i_f)$ where s^i_f is the i-th judgment test of s.*
- Tape*: $T_{\mathsf{ext}}(s) = T^s_{\mathsf{ext}}(\mathbb{A}, 0)$ where $T^s_{\mathsf{ext}}(\mathbb{A}, i)$ is the auxiliary function defined by induction on $\mathbb{A}$ as follows.*

$$\begin{array}{lcl} T^s_{\mathsf{ext}}(\langle\cdot\rangle, i) & := & \epsilon \\ T^s_{\mathsf{ext}}(S \to \mathbb{A}, i) & := & \bullet \cdot T^s_{\mathsf{ext}}(\mathbb{A}, i) \\ T^s_{\mathsf{ext}}([...\mathbb{A}...] \to B, i) & := & l_{\mathsf{ext}}(s^i_f) \cdot T^s_{\mathsf{ext}}(\mathbb{A}, i+1) \end{array}$$

where s^i_f is the i-th type test of s.

We use $\simeq_{\mathsf{ext}}$ for the extraction relation between S-exhaustible SIAM states and λIAM states defined as $(s, s_{\mathsf{ext}}(s)) \in \simeq_{\mathsf{ext}}$.

First of all, we show that the extracted state respects the λIAM invariant about the length of the log.

Lemma 4.3.11. *Let s be an S-exhaustible SIAM state and $s_{\mathsf{ext}}(s) = (t, C_s, L_{\mathsf{ext}}(s), T_{\mathsf{ext}}(s), d)$ the λIAM state extracted from it. Then the level of C_s is exactly the length of $L_{\mathsf{ext}}(s)$, that is, $(t, C_s, L_{\mathsf{ext}}(s))$ is a logged position.*

Proof. The length of $L_{\mathsf{ext}}(s)$ is the number of judgment tests of s, which is the number of T-@ rules traversed descending from the focused judgment J of s to the final judgment of π. The level of C_s is the number of arguments in which the hole of C_s is contained, which are exactly the number of T-@ rules traversed descending from J to the final judgment of π. □

Now we are able to state (and prove) that $\simeq_{\mathsf{ext}}$ is a strong bisimulation.

Proposition 4.3.12 (SIAM-λIAM bisimulation)**.** *Let t a closed and $\to_{wh}$-normalizable term, and $\pi \triangleright \vdash t : \star$ a type derivation. Then $\simeq_{\mathsf{ext}}$ is a strong bisimulation between S-exhaustible SIAM states on π and λIAM states on t. Moreover, if $s_\pi \simeq_{\mathsf{ext}} s_\lambda$ then s_π is SIAM reachable if and only if s_λ is λIAM reachable.*

The proof, long but conceptually simple, is in the Appendix.

4.3.3 Acyclicity and The Correctness of the Weighting System

Weights and the Length of SIAM Runs via Acyclicity. We now turn to the proof of the correctness of the weight assignment $\mathbf{W}_{\lambda\mathrm{IAM}}(\pi)$, that is, the fact that it correctly measures the length of λIAM complete runs. While the weight assignment for the λIAM is similar to de Carvalho's one for the KAM, the proof of its correctness is completely different, and it must be, as we know explain. The KAM performs an evaluation that essentially mimics cut-elimination and so the number of KAM transitions to normal form is obtained via a refined, quantitative form of subject reduction. One may say that it is obtained in a *step-by-step* manner. The λIAM, instead, does not mimic subject reduction. It walks over the type derivation *without ever changing it*, potentially passing many times over the same judgment (because of backtracking). Correctness of weights cannot then be obtained via a refined subject reduction property, because the reduced derivation gives rise to a different run, and not to a sub-run. It must instead follow from a *global* analysis of a fixed derivation, that we now develop.

Weights as in $\mathbf{W}_{\lambda\text{IAM}}(\pi)$ count the number of occurrences of $\star$ in π, and every such occurrence corresponds to a state of the SIAM. Proving the correctness of the weight system amounts to showing that every state of the SIAM is reachable, and reachable exactly once. In order to do so, we have to show that the SIAM never loops on typed derivations.

Note a subtlety: by the bisimulation with the λIAM (Prop. 4.3.12) we know that the run of the SIAM terminates, but we do not know whether it reaches all states (*i.e.* all $\star$ occurrences). What we have to prove, then, is that there are no unreachable loops, that is, loops that are not reachable from an initial state. The next easy lemma guarantees that this is enough.

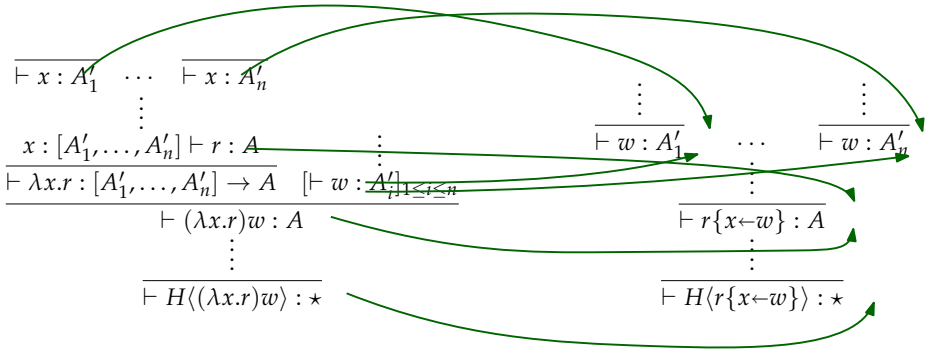
Lemma 4.3.13. *Let T be an acyclic bi-deterministic transition system on a finite set of states $\mathcal{S}$ and with only one initial state s_i. Then all states in $\mathcal{S}$ are reachable from s_i, and reachable only once.*

Proof. Let us consider a generic state $s \in \mathcal{S}$ and show that it is reachable from s_i. If $s = s_i$ we are done. Otherwise, since the system is bi-deterministic we can deterministically go backwards from s. Since the set of states is finite and there are no cycles, then the backward sequence must end on an initial state, that is, on s_i. Thus s is reachable from s_i. If a state is reachable twice, then clearly there is a cycle, absurd. □

We show the absence of loops using a sort of subject reduction property. We first show that if the SIAM loops on $\pi \triangleright \vdash t : \star$ and $t \to_{wh} u$, then there is a type derivation $\pi' \triangleright \vdash u : \star$ on which the SIAM loops—that is, SIAM looping is preserved by reduction of the underlying term. This is done by defining a relation $\blacktriangleright$ between the SIAM states on π and on π'. This idea has been already exploited in Section 3.6, indeed we have to prove that $\blacktriangleright$ is an improvement.

Explaining the Bisimulation via a Diagram. Let us give an intuitive explanation of the improvement $\blacktriangleright$ that we are going to build next. Given two type derivations $\pi \triangleright \vdash H\langle(\lambda x.r)w\rangle : \star$ and $\pi' \triangleright \vdash H\langle r\{x\leftarrow w\}\rangle : \star$, it is possible to define a relation $\blacktriangleright$ between states of the former and of the latter as depicted in the figure below. The key points are:

1. each axiom for x in π is $\blacktriangleright$-related with the judgment for the argument w that replaces it in π'.
2. Both the judgment for r and the one for $(\lambda x.r)w$ are $\blacktriangleright$-related to $r\{x\leftarrow w\}$.
3. The judgment for $\lambda x.r$ is not $\blacktriangleright$-related to any judgment of π'.



Defining $\blacktriangleright$. In order to define $\blacktriangleright$ formally, we enrich each type judgment (occurrence) $\vdash t : \mathbb{A}\langle\star\rangle$ with a context C such that $C\langle t\rangle$ is the term in the final judgment of the derivation π, obtaining $\vdash (t, C) : \mathbb{A}\langle\star\rangle$.

Definition 4.3.14 (Bisimulation $\blacktriangleright$). *The definition of $\blacktriangleright$ for $\vdash (t, C) : \mathbb{A}\langle\star\rangle$ has 4 clauses:*

- rdx*: the redex is in t, that is, $t = H\langle(\lambda x.u)r\rangle$, and so C is a head context K:*

$$\vdash (H\langle(\lambda x.u)r\rangle, K) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\mathsf{rdx}} \vdash (H\langle u\{x\leftarrow r\}\rangle, K) : \mathbb{A}\langle\star\rangle$$

- body: *the term t is part of the body of the abstraction involved in the redex:*

$$\vdash (t, H\langle(\lambda x.D)u\rangle) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\mathsf{body}} \vdash (t\{x\leftarrow u\}, H\langle D\{x\leftarrow u\}\rangle) : \mathbb{A}\langle\star\rangle$$

- arg: *the term t is part of the argument of the redex:*

$$\vdash (t, H\langle(\lambda x.D\langle x\rangle)E\rangle) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\mathsf{arg}} \vdash (t, H\langle D\{x\leftarrow E\langle t\rangle\}\langle E\rangle\rangle) : \mathbb{A}\langle\star\rangle$$

- ext: *The term t is disjoint form the redex, that then takes place only in C:*

$$\vdash (t, K\langle H\langle(\lambda x.r)u\rangle D\rangle) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\mathsf{ext}} \vdash (t, K\langle H\langle r\{x\leftarrow u\}\rangle D\rangle) : \mathbb{A}\langle\star\rangle$$

Please note that the only states of π which are not mapped to any state of π' are those relative to the judgment $\vdash \lambda x.r : [B_1 ... B_n] \to A$.

Proposition 4.3.15. *$\blacktriangleright$ is a loop-preserving improvement between SIAM states.*

The proof, again quite long but conceptually simple, is carried out in the Appendix.

Corollary 4.3.16. *If $\pi \triangleright \vdash H\langle(\lambda x.r)w\rangle : \star$ contains a cycle, the also $\pi' \triangleright \vdash H\langle r\{x\leftarrow w\}\rangle : \star$ contains a cycle.*

Proof. If the run of the SIAM on $\pi \triangleright \vdash H\langle(\lambda x.r)w\rangle : \star$ loops then there exists a state s_π such that a computation starting from s_π diverges. Every state but $\vdash (\lambda x.r, H\langle\langle\cdot\rangle w\rangle) : \mathbb{A}\langle\star\rangle$, which however is not final, is related by $\blacktriangleright$ to a state $s_{\pi'}$ of $\mathsf{T}_{\pi'}$. Since improvements preserve non-termination (Prop. 3.6.2.1), also $s_{\pi'}$ diverges. Since $s_{\pi'}$ has a finite number of states, there must be a cycle. □

Other works dealing with the GoI also prove the absence of unreachable loops, for instance (Baillot, 1999; Baillot et al., 2011). Then, by the trivial fact that the SIAM does not loop on $\to_{wh}$-normal terms (as they are typed using just one rule, namely T-$\lambda_\star$), we obtain that it never loops.

Corollary 4.3.17. *Let $\pi \triangleright \vdash t : \star$ be a type derivation. Then the SIAM does not loop on π.*

Proof. Since t is typable, then it has normal form, call it u. Clearly the type derivation for u has no cycles. By the previous corollary, also π cannot have any of them. □

Quantitative Correctness. The correctness of the weights for the length of SIAM runs immediately follows, and, via the strong bisimulation in Prop. 4.3.12, it transfers to the λIAM.

Theorem 4.3.18 (λIAM time via sequence types)**.** *Let t be a closed term that is $\to_{wh}$-normalizing, σ the complete λIAM run from s_t, and $\pi \triangleright \vdash t : \star$ a type derivation for t. Then $|\sigma| = \mathbf{W}_{\lambda IAM}(\pi)$.*

Proof. By Lemma 4.3.13 every state of T_π is traversed exactly once, during a computation that starts from the initial state. Thus the length of the computation is the cardinality of the states of T_π. Since a state in a type judgment $\Gamma \vdash t : A$ occurring in π is given by an occurrence of $\star$ in A, then for every judgment the number of associated states is $\|A\|$. Then, it is immediate to note that the number of states in a type derivation ending in $\pi \triangleright \overset{n}{\vdash} t : \star$ is exactly n. □

Chapter 5

Tree Types Capture IAM Space

On the same line of the previous chapter, we would like to measure also the λIAM *space* consumption by the way of a type system. This is particularly interesting because of the already cited results about the space efficiency of the λIAM. Indeed, a type-theoretic characterization could be a very useful tool to study the space behaviour of the λIAM in the general case of the pure λ-calculus.

λIAM Space Consumption. The space needed to represent a λIAM state is given by the following definition (the meta-variable Γ to denote either a tape T or a log L):

$$|(t, C, L, T, d)|_{\mathsf{sp}} := |L|_{\mathsf{sp}} + |T|_{\mathsf{sp}}$$

$$\begin{array}{rclcrcl} |(x, D, L')|_{\mathsf{sp}} & := & \mathsf{X} + |L'|_{\mathsf{sp}} & \qquad & |\epsilon|_{\mathsf{sp}} & := & 0 \\ |l \cdot \Gamma|_{\mathsf{sp}} & := & |l|_{\mathsf{sp}} + |\Gamma|_{\mathsf{sp}} & & |\bullet \cdot T|_{\mathsf{sp}} & := & 1 + |T|_{\mathsf{sp}} \end{array}$$

The value of the unknown X is simply the size of a pointer to a subterm of the term under evaluation, *i.e.* $\mathsf{X} = \log |C\langle t \rangle|$. Then, we are able to define the space of a λIAM run by taking the maximum size of the states reached during the run.

Definition 5.0.1. *Let $\rho : s_0 \to^*_{\lambda\mathrm{IAM}} s$ be a λIAM run. Then,*

$$|\rho|_{\mathsf{sp}} := \max_{s' \in \rho} |s'|_{\mathsf{sp}}$$

First of all, it is worth noticing what happens in the case of diverging computations. In principle, two cases could occur: either the space consumption is finite, or it is infinite. Actually, it is easy to prove that the first case is not possible.

From bi-determinism it is immediate to prove acyclicity for reachable states.

Proposition 5.0.2. *Let s be a reachable λIAM state. Then s is reached exactly once.*

Proof. We proceed by induction on the length of the run $\rho : s_0 \to^*_{\lambda\mathrm{IAM}} s$. If $|\rho| = 0$, the result is trivial. Otherwise, if $|\rho| > 0$, we have $\rho : s_0 \to^*_{\lambda\mathrm{IAM}} s' \to_{\lambda\mathrm{IAM}} s$. Let us call $\sigma : s_0 \to^*_{\lambda\mathrm{IAM}} s'$. By *i.h.*, every state in σ is reached exactly once. Then s cannot be part of σ, because otherwise it would have two different predecessors, contradicting bi-determinism. Thus s is reached exactly once in ρ. □

Then, since there are no cycles, an infinite run goes through infinite different states and thus consumes unbounded space.

Proposition 5.0.3. *Let ρ be an infinite λIAM run. Then $|\rho|_{\mathsf{sp}} = \infty$.*

Proof. The result comes from the fact that in finite amount of memory, only a finite amount of configurations can be encoded. Since in an infinite run, an infinite number of *different states* are reached, then the λIAM needs an unbounded space to perform the computation. □

5.1 Tree (Intersection) Types

Here we introduce the type system that we shall use to measure the space used by λIAM runs.

$$\frac{}{x : [A] \vdash x : A}\ \text{T-VAR} \qquad \frac{\Gamma, x : T \vdash t : A}{\Gamma \vdash \lambda x.t : T \to A}\ \text{T-}\lambda$$

$$\frac{\Gamma \vdash t : T \to A \quad \Delta \vdash u : T}{\Gamma \uplus \Delta \vdash tu : A}\ \text{T-@} \qquad \frac{}{\vdash \lambda x.t : \star}\ \text{T-}\lambda_\star$$

$$\frac{\Gamma_i \vdash t : G_i \quad 1 \leq i \leq n}{[\uplus_{i=1}^n \Gamma_i] \vdash t : [G_1, .., G_n]}\ \text{T-MANY} \qquad \frac{}{\vdash t : [\cdot]}\ \text{T-NONE}$$

FIGURE 5.1: The tree type system.

We adapt sequence types of Chapter 4 in order to capture the size of λIAM states. In particular, we need to store more information into types. Indeed, multi/sequence types are usually defined by two mutually dependent layers, a linear one containing ground types and (linear) arrow types, and the multiset level containing linear types. Here we also have two layers, except that we allow multisets to also contain multisets, thus we can have $[A, [[B, B], A, [A]], A, B]$. A nested multiset is a *tree* whose leaves are linear types and whose internal nodes are nested multisets.

Basic Definitions. As for multi types, there are two mutually defined layers of types, *linear types* and *tree types*.

$$\begin{array}{rrcll} \text{LINEAR TYPES} & A, B & ::= & \star \mid T \to A & \\ \text{TREE TYPES} & T, T' & ::= & [G_1, \ldots, G_n] & n \geq 0 \\ \text{(GENERIC) TYPES} & G, G' & ::= & A \mid T & \end{array}$$

As the reader can note, the grammar for tree types is almost identical to that of sequence/multi types, except for the fact that now trees can contain other trees, and not only linear types. About trees, since commutativity is ruled out, we have, for instance, that $[A, B] \neq [B, A]$. As always, note that the empty tree type/sequence is a valid type, which is noted $[\cdot]$. The concatenation of two sequences T and T' is noted $T \uplus T'$. We use $|T|$ for the length of T as a sequence, that is, $|[G_1, .., G_n]| := n$. Notations and conventions about type judgments and derivations are the same as those introduced in previous chapters.

The typing rules are in Fig. 5.1. With respect to the literature, the differences are in rule T-MANY. There are two differences. The first one is the already mentioned fact that premises may assign both linear types and tree types, while the literature usually only allows linear types. The second one is that the rule surrounds $\Gamma := \uplus_{i=1}^n \Gamma_i$ with an additional nesting level—the notation $[\Gamma]$ standing for the type environment $x_1 : [T_1], \ldots, x_n : [T_n]$ if $\Gamma = x_1 : T_1, \ldots, x_n : T_n$.

A Small Example. We show an instance of the rule T-MANY in the delicate case in which the premises contain the same free variable x.

$$\frac{x : [A_1] \vdash t : G_1 \quad x : [A_2] \vdash t : G_2}{x : [[A_1, A_2]] \vdash t : [G_1, G_2]}\ \text{T-MANY}$$

In particular, please note that first $[A_1]$ and $[A_2]$ are joined, and then they are surrounded by an additional nesting level. The other option would have been $x : [[A_1], [A_2]]$, but it is not what T-MANY does.

Leaves Extraction and Leaf Contexts. Every tree type T induces the sequence $\underline{T}$— equivalently, the flat tree type—of its leaves, defined by the following *leaves extraction* operation.

$$[\cdot]^\ell := [\cdot] \quad ([A] \uplus T)^\ell := [A] \uplus T^\ell \quad ([T'] \uplus T)^\ell := T'^\ell \uplus T^\ell$$

We shall describe the leaves of a tree type also via a notion of leaf context.

$$\text{LEAF CTXS} \quad \mathbb{L} ::= [G_1, .., \langle\cdot\rangle, .., G_n] \;\Big|\; [G_1, .., \mathbb{L}, .., G_n]$$

If $T^\ell = [A_1, .., A_n]$ then for every A_i there is a leaf context $\mathbb{L}^i$ such that $T = \mathbb{L}^i\langle A_i\rangle$. Therefore, we shall use the notation $T = \mathbb{L}^i\langle A\rangle$, or even simply $T^i = A$, to say that the linear type A is the i-th leaf of T.

In the following we use two basic properties of the type system (the same already given for the sequence type system), collected in the following straightforward lemma. One is the absence of weakening, and the other one is a correspondence between sequence types and axioms.

Lemma 5.1.1 (Relevance and axiom sequences). *If $\pi \triangleright \Gamma \vdash t : A$ then $dom(\Gamma) \subseteq \mathsf{fv}(t)$, thus if t is closed then Γ is empty. Moreover, there are exactly $|\Gamma(x)^\ell|$ axioms typing x in π, which appear from left to right as leaves of π (seen as an ordered tree) in the order given by $\Gamma(x)^\ell = [A_1, \ldots, A_k]$ and that the i-th axiom types x with A_i.*

Characterization of Termination. It is well-known that intersection and multi types characterize Closed CbN termination, that is, they type *all* and only those λ-terms that terminate with respect to Closed CbN. Moreover, every term that is Closed CbN normalizable can be typed with $\star$. The same characterization holds with tree types, following the standard recipe[1] for multi types, without surprises. See the Appendix for details.

Theorem 5.1.2 (Correctness and completeness of tree types for Closed CbN). *A closed term t is Closed CbN normalizable if and only if there exists a tree type derivation $\pi \triangleright \vdash t : \star$.*

Relationship with Multi Types. The leaves extraction operation can easily be extended to a flattening function turning a tree type into a multi type. Flattening can also be extended to derivations, by collapsing trees of T-MANY rules into the more traditional rule for multi sets that does not *nodify* the type context. In this way, one obtains a forgetful transformation, easily defined by induction on derivations. A converse *lifting* transformation, however, cannot be defined by induction on derivations—it is unclear how to define it on applications. This fact is evidence that tree types are strictly richer than multi types, because the tree structure cannot be inferred from the multiset one.

5.2 The Tree IAM

This section introduces a machine evaluating type derivations, the *Tree IAM*, or *TIAM*, that mimics the λIAM directly on top of a type derivation π, as we have already done with sequence types. It is the key tool that we shall use to measure the space cost of λIAM runs. Indeed, the TIAM is a very minor variant over the similar SIAM machine evaluating type derivations for sequence types of the previous chapter.

The TIAM. The TIAM moves over a fixed type derivation $\pi \triangleright \vdash t : \star$, to be thought as the code, following the occurrence of $\star$ in the final judgment through π, according to the transitions in Fig. 5.2. It is defined in the very same ways of the SIAM. Although identical to the previous chapter, we shall now explain every involved concept.

The position of the machine is given by an occurrence of a type judgment[2] J of π. As the λIAM, the TIAM has two possible directions, noted $\downarrow$ and $\uparrow$[3]. In direction $\uparrow$ the machine looks at the rule above the focused judgment, in direction $\downarrow$ at the rule below. The only "data structure"—encoding

[1] Namely, substitution lemma plus subject reduction for correctness, and anti-substitution lemma, subject expansion, and typability of all normal forms for completeness (here trivial, because all normal forms are typed by T-$\lambda_\star$).

[2] A judgment may occur repeatedly in a derivation, which is why we talk about *occurrences* of judgments. To avoid too many technicalities, however, we usually just write the judgment, leaving implicit that we refer to an occurrence of that judgment.

[3] Type derivations are upside-down wrt to the term structure, then direction $\downarrow$ of the λIAM becomes here $\uparrow$, and $\uparrow$ is $\downarrow$.

$$\frac{\vdash t : T \to A \quad \vdash}{\vdash tu : \mathbb{A}\langle\star_{\uparrow}\rangle} \quad \to_{\bullet 1} \quad \frac{\vdash t : T \to \mathbb{A}\langle\star_{\uparrow}\rangle \quad \vdash}{\vdash tu : A}$$

$$\frac{\vdash t : A}{\vdash \lambda x.t : T \to \mathbb{A}\langle\star_{\uparrow}\rangle} \quad \to_{\bullet 2} \quad \frac{\vdash t : \mathbb{A}\langle\star_{\uparrow}\rangle}{\vdash \lambda x.t : T \to A}$$

$$\frac{\vdash t : T \to \mathbb{A}\langle\star_{\downarrow}\rangle \quad \vdash}{\vdash tu : A(= \mathbb{A}\langle\star\rangle)} \quad \to_{\bullet 3} \quad \frac{\vdash t : T \to A \quad \vdash}{\vdash tu : \mathbb{A}\langle\star_{\downarrow}\rangle}$$

$$\frac{\vdash t : \mathbb{A}\langle\star_{\downarrow}\rangle}{\vdash \lambda x.t : T \to A} \quad \to_{\bullet 4} \quad \frac{\vdash t : A}{\vdash \lambda x.t : T \to \mathbb{A}\langle\star_{\downarrow}\rangle}$$

$$\frac{\dfrac{}{\vdash x : \mathbb{A}\langle\star_{\uparrow}\rangle_i}\, i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle A_i\rangle \to B} \quad \to_{\mathsf{var}} \quad \frac{\dfrac{}{\vdash x : A_i}\, i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle\mathbb{A}\langle\star_{\downarrow}\rangle_i\rangle \to B}$$

$$\frac{\dfrac{}{\vdash x : A_i}\, i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle\mathbb{A}\langle\star_{\uparrow}\rangle_i\rangle \to B} \quad \to_{\mathsf{bt2}} \quad \frac{\dfrac{}{\vdash x : \mathbb{A}\langle\star_{\downarrow}\rangle_i}\, i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle A_i\rangle \to B}$$

$$\frac{\vdash t : \mathbb{L}\langle\mathbb{A}\langle\star_{\downarrow}\rangle_i\rangle \to A \qquad \dfrac{.. \vdash u : \mathbb{A}\langle\star\rangle_i ..}{\vdash u : T}}{\vdash tu : A} \quad \to_{\mathsf{arg}} \quad \frac{\vdash t : T \to A \qquad \dfrac{.. \vdash u : \mathbb{A}\langle\star_{\uparrow}\rangle_i ..}{\vdash u : \mathbb{L}\langle\mathbb{A}\langle\star\rangle_i\rangle}}{\vdash tu : A}$$

$$\frac{\vdash t : T \to A \qquad \dfrac{.. \vdash u : \mathbb{A}\langle\star_{\downarrow}\rangle_i ..}{\vdash u : \mathbb{L}\langle\mathbb{A}\langle\star\rangle_i\rangle}}{\vdash tu : A} \quad \to_{\mathsf{bt1}} \quad \frac{\vdash t : \mathbb{L}\langle\mathbb{A}\langle\star_{\uparrow}\rangle_i\rangle \to A \qquad \dfrac{.. \vdash u : \mathbb{A}\langle\star\rangle_i ..}{\vdash u : T}}{\vdash tu : A}$$

FIGURE 5.2: The transitions of the Tree IAM (TIAM).

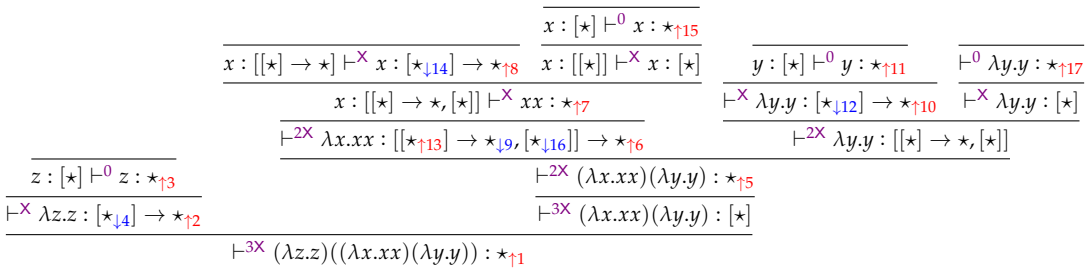


FIGURE 5.3: An example of TIAM execution.

the tape of the λIAM, as we shall explain—is a type context $\mathbb{A}$ isolating an occurrence of $\star$ in the type A of the focused judgment (occurrence) $\Gamma \vdash u : A$, defined as follows (careful to not confuse type contexts $\mathbb{A}$ and $\mathbb{T}$ with type environments Γ):

$$\begin{array}{rrl} \text{LINEAR CTXS} & \mathbb{A} ::= \langle\cdot\rangle \mid T \to \mathbb{A} \mid \mathbb{T} \to A \\ \text{TREE CTXS} & \mathbb{T} ::= [G_1, .., \mathbb{G}, .., G_n] \\ \text{TYPE CTXS} & \mathbb{G} ::= \mathbb{A} \mid \mathbb{T} \end{array}$$

Summing up, a state s is a quadruple $(\pi, J, \mathbb{A}, d)$. If J is in the form $\Gamma \vdash u : A$, we often write s as $\vdash u : \mathbb{A}\langle\star_d\rangle$, where $\mathbb{A}\langle\star\rangle = A$. We shall see that type environments play no role.

Intuitions about Positions (and Logs). The intuition is that the active position (t, C_n) of a λIAM state corresponds to the judgment occurrence J in the TIAM, or, more precisely, to its position in the type derivation π. The sub-term t is exactly the term typed by J. The context C_n is exactly the context giving the term $C_n\langle t\rangle$ typed by the final judgment of π. The level n of the context C_n of the active position counts the number of arguments in which the hole of C_n is contained. On the type derivation, each such argument is associated to a T-@ rule having the current judgment J in its right sub-derivation. Last, note that in the λIAM the current log L_n has length equal to n. We shall see in the next section that one can recover the log L_n applying an extraction process to the T-@ rules found descending from J towards the final judgment.

Transitions. The TIAM starts on the final judgment of π, with empty type context $\mathbb{A} = \langle\cdot\rangle$ and direction $\uparrow$. It moves from judgment to judgment, following occurrences of $\star$ around π. To specify the transitions, we use the leaf contexts defined in the previous section and we mark by the double horizontal line trees of T-MANY rules. Exactly as the λIAM, the TIAM is bi-deterministic.

Proposition 5.2.1. *The TIAM is bi-deterministic for each type derivation $\pi \triangleright \vdash t : \star$.*

An Example. In Fig. 5.3 we present the an example of (weighted) type derivation. We have reported the occurrences of $\star$ on the right of $\vdash$ annotated with increasing integers and a direction. The occurrence of $\star$ marked with 1 represents the first state, and so on. One can immediately notice that every occurrence of $\star$ is visited exactly once. Please note that judgments are decorated with weights (such as X), which shall be introduced only in Sect. 5.3, in order to later provide an example of decoration—they can be safely ignored for now.

5.2.1 The λIAM and TIAM Bisimulation

The aim of this section is to explain the strong bisimulation between the TIAM and the λIAM, that is essentially the same between the SIAM and the λIAM studied in the previous chapter. A striking point of the TIAM is that it does not have the log nor the tape. They are encoded in the position of the judgment occurrence J and in the type context $\mathbb{A}$ of its states, as we shall show.

$$\frac{}{\underset{x:[A]}{\overset{\|A\|}{\vdash}}\ x:A}\ \text{T-VAR} \qquad \frac{\Gamma, x:T \overset{w}{\vdash} t:A}{\underset{\Gamma}{\overset{\max\{w,\|T\to A\|\}}{\vdash}}\ \lambda x.t : T\to A}\ \text{T-}\lambda \qquad \frac{\Gamma \overset{w}{\vdash} t : T\to A \quad \Delta \overset{v}{\vdash} u : T}{\underset{\Gamma\uplus\Delta}{\overset{\max\{w,v\}}{\vdash}}\ tu : A}\ \text{T-@}$$

$$\frac{}{\underset{\vdash \lambda x.t : \star}{\overset{0}{\vdash}}}\ \text{T-}\lambda_\star \qquad \frac{\Gamma_i \overset{w_i}{\vdash} t : G_i \quad 1\le i\le n}{\underset{[\uplus_{i=1}^n \Gamma_i]}{\overset{X+\max_i\{w_i\}}{\vdash}}\ t : [G_1,..,G_n]}\ \text{T-M} \qquad \frac{}{\underset{\vdash t : [\cdot]}{\overset{0}{\vdash}}}\ \text{T-NONE}$$

FIGURE 5.4: The tree type system with weights.

Relating Logs and Tapes with Typed Positions. In the λIAM, the log $L = l_1 \cdot \ldots \cdot l_n$ has a logged position for every argument $u_1, \ldots, u_n$ in which the position of the current state is contained. The argument u_i is the answer to the query of an argument for the variable in the logged position l_i. The TIAM does not keep a trace of the variables for which it completed a query, but the answers to those (forgotten) queries are simply given by the sub-derivations for $u_1, \ldots, u_n$ in which the current judgment occurrence J is contained—the way in which l_k identifies a copy of u_k in the λIAM corresponds on the type derivation π to the index i of the leaf (in the tree of sub-derivations) typing u_k in which J is located. Note that the λIAM manipulates the log only via transitions $\to_{\mathsf{arg}}$ and $\to_{\mathsf{bt1}}$, that on the TIAM correspond exactly to entering/exiting derivations for arguments. The tape, instead, contains logged positions for which the λIAM either has not yet found the associated argument, or it is backtracking to. Note that the λIAM puts logged positions on the tape via transitions $\to_{\mathsf{var}}$ and $\to_{\mathsf{bt1}}$, and removes them using $\to_{\mathsf{arg}}$ and $\to_{\mathsf{bt2}}$. By looking at Fig. 5.2, it is clear that there is a logged position on the λIAM tape for every type sequence of the flattening of T in which it lies the hole $\langle\cdot\rangle$ of the current type context $\mathbb{A}$ of the TIAM.

Extracting λIAM States. These ideas are used to extract from every TIAM state s a λIAM state $\mathsf{ext}(s)$ in a quite technical way. In particular, the extraction process retrieves a log $L_{\mathsf{ext}}(s)$ from the judgment J of s and a tape $T_{\mathsf{ext}}(s)$ from the type context $\mathbb{A}$ of s, using a sophisticated *T-exhaustible invariant* of the TIAM to retrieve the exact shape of the logged positions in $L_{\mathsf{ext}}(s)$ and $T_{\mathsf{ext}}(s)$.

Let us give a high-level description of how extraction works. The invariant is based on the pairing of every TIAM state s with a set of *test states*, some coming from the judgment J of s, called *judgment tests*, and some coming from the type context $\mathbb{A}$, called *type (context) tests*. The invariant guarantees a certain recursive property of each test state. The extraction process uses this property to extract a logged position $l_{s'}$ from each test state s' of s.

Given a TIAM state $s = (\pi, J, \mathbb{A}, d)$, its judgment tests are associated to the T-@ rules having J in their right sub-derivation. Their extractions give logged positions $l_1 \cdot \ldots \cdot l_n$ forming the extracted log $L_{\mathsf{ext}}(s)$, following the correspondence described above.

Type tests are associated to the leaf contexts surrounding the hole of $\mathbb{A}$. The extraction of the tape $T_{\mathsf{ext}}(s)$ from $\mathbb{A}$ is done according to the following schema:

$$\begin{array}{lcl} T_{\mathsf{ext}}(\langle\cdot\rangle) & := & \epsilon \\ T_{\mathsf{ext}}(T \to \mathbb{A}) & := & \bullet \cdot T_{\mathsf{ext}}(\mathbb{A}) \\ T_{\mathsf{ext}}(\mathbb{L}\langle\mathbb{A}\rangle \to B) & := & l_{\mathsf{ext}}(s_{\mathbb{L}}) \cdot T_{\mathsf{ext}}(\mathbb{A}) \end{array}$$

where $s_{\mathbb{L}}$ is the state test associated to the leaf context $\mathbb{L}$.

Since this is essentially identical to what have been done in the previous chapter for the SIAM, the technical development is in the Appendix. The extraction process induces a relation $s \simeq_{\mathsf{ext}} \mathsf{ext}(s)$ that is easily proved to be a strong bisimulation between the TIAM and the λIAM.

Proposition 5.2.2 (TIAM and λIAM bisimulation)**.** *Let t a closed term and $\pi \triangleright \vdash t : \star$ a tree type derivation. Then $\simeq_{\mathsf{ext}}$ is a strong bisimulation between TIAM states on π and λIAM states on t. Moreover, if $s_\pi \simeq_{\mathsf{ext}} s_\lambda$ then s_π is TIAM reachable if and only if s_λ is λIAM reachable.*

The *moreover* part of the above statement hints at a bijection between *all* the states in π and reachable λIAM states. However, there still could be the possibility that some of the states in π are not reachable. This is actually *not* the case, as for the SIAM.

Proposition 5.2.3. *Let t a closed term and $\pi \triangleright \vdash t : \star$ a tree type derivation. Then every state of π is reached exactly once.*

5.3 Measuring the Space of Interaction

This section contains the main contribution of the chapter: it gives a way of measuring the space consumed by the complete λIAM run on the term t via a quantitative analysis of the tree type derivation for t. We proceed in two steps.

1. *The space of single extracted states*: given a TIAM state $s = (\pi, J, \mathbb{A}, d)$, we show how to measure the space of the extracted λIAM state $\mathsf{ext}(s)$ from π, J, and $\mathbb{A}$.
2. *The space of the whole execution*: we refine tree type derivations adding *weights* on judgments, and show that the weight of the final judgment coincides with the maximum space consumption over all extracted states, that is, along the whole λIAM execution.

The Undetermined Pointers Size X. A technical point common to both parts is that the quantitative study of tree types derivations is relative to an undetermined value X. The reason for using X is that our space analyses have both local and global components. Locally, we count how many occurrences of $\bullet$ and how many logged positions are involved in a state (for step 1) or in all states in and above that judgment (for step 2). The global component comes from the fact that all logged positions of the λIAM, independently of where they arise, are implemented via pointers to the *global* code. Essentially, X is meant to be replaced, at the very end of both analyses, by the size of pointers to the λIAM global code, that is, by $\log |t_0|$, where t_0 is the term typed in the final judgment of the type derivation π. Therefore, locally our measures shall include X, which shall substituted at the end with $\log |t_0|$.

5.3.1 The Space of Single Extracted States

Trees and the Size of Extracted Logged Positions. Basically, given a TIAM state $s = (\pi, J, \mathbb{A}, d)$, the size of logged positions in $\mathsf{ext}(s)$ is obtained by counting X for

- *Extracted tape*: every sequence constructor $[\cdot]$ surrounding the hole $\langle\cdot\rangle$ in $\mathbb{A}$;
- *Extracted log*: every T-MANY rule on the path from J to the final judgment of π.

Clearly, it is the newly introduced tree structure that allows to measure the size of extracted logged positions, as expected. First, we define a size of type contexts meant to measure the size of the extracted tapes.

Definition 5.3.1 (Branch size of type contexts/extracted tapes)**.** *Let $s = (\pi, J, \mathbb{A}, d)$ be a TIAM state. The branch size $|\cdot|_{\mathsf{b}}$ for type contexts is defined as follows:*

$$\begin{array}{rclcrcl} |\langle\cdot\rangle|_{\mathsf{b}} & := & 0 & & |T \to \mathbb{A}|_{\mathsf{b}} & := & 1 + |\mathbb{A}|_{\mathsf{b}} \\ |\mathbb{T} \to A|_{\mathsf{b}} & := & |\mathbb{T}|_{\mathsf{b}} & |[G_1, .., \mathbb{G}, .., G_n]|_{\mathsf{b}} & := & \mathsf{X} + |\mathbb{G}|_{\mathsf{b}} \end{array}$$

Let us interpret the branch size with respect to the tape extraction schema of Section 5.2.1. The $+1$ in the clause for $T \to \mathbb{A}$ is there to count $\bullet$. The clause for sequences instead gives $|\mathbb{L}| = n \cdot \mathsf{X}$ if the hole has height n in the leaf context $\mathbb{L}$ seen as a tree—whence the name *branch size*.

Then, we define a branch size for judgments, meant to measure the size of extracted logs, and a branch size for states.

Definition 5.3.2. *Let $s = (\pi, J, \mathbb{A}, d)$ be a TIAM state.*

- Branch size of judgments/extracted logs: *let n be the number of* T-MANY *rules encountered descending from J to the root of π. Then* $|J|_{\mathsf{b}} := n \cdot \mathsf{X}$.
- Branch size of TIAM states: $|s|_{\mathsf{b}} := |\mathbb{A}|_{\mathsf{b}} + |J|_{\mathsf{b}}$.

We prove that the defined branch sizes do correspond to their intended meanings, that is, the branch sizes of extracted logs and tapes, showing that the size of TIAM states captures the space size of the extracted λIAM state.

Proposition 5.3.3 (Space of Single Extracted States)**.** *Let $s = (\pi, J, \mathbb{A}, d)$ be a reachable TIAM state. Then $|\mathbb{A}|_{\mathsf{b}} = |T_{\mathsf{ext}}(s)|_{\mathsf{sp}}$ and $|J|_{\mathsf{b}} = |L_{\mathsf{ext}}(s)|_{\mathsf{sp}}$, and thus $|s|_{\mathsf{b}} = |\mathsf{ext}(s)|_{\mathsf{sp}}$. Moreover,*

1. *if $L_{\mathsf{ext}}(s) = l_1..l_n$, and let h_i be the number of* T-MANY *rules of the i^{th}* T-MANY *rule tree found descending from J to the root of π, then $|l_i|_{\mathsf{sp}} = h_i$;*
2. *for each extracted tape position l, i.e. for each $\mathbb{G}$ such that $\mathbb{A} = \mathbb{G}\langle \mathbb{L}\langle \mathbb{A}'\langle\star\rangle\rangle \to A\rangle$, then $|l|_{\mathsf{sp}} = |\mathbb{L}|\cdot\mathsf{X}$.*

Proof. We proceed by induction on the length of the run $\sigma : s_0 \to^*_{\text{TIAM}} s$. If the length is 0, then $s = s_0 = (\pi, J, \langle\cdot\rangle, \uparrow)$, $J = \vdash t : \star$ and is the root of ρ. Then $|\langle\cdot\rangle|_{\mathsf{b}} = 0 = |T_{\mathsf{ext}}(s)|_{\mathsf{sp}}$ and $|J|_{\mathsf{b}} = 0 = |L_{\mathsf{ext}}(s)|_{\mathsf{sp}}$. Otherwise, $\sigma : s_0 \to^n_{\text{TIAM}} s' \to_{\text{TIAM}} s$. We analyze the different cases of the last transition.

- Case $\to_{\bullet 1}$.

$$s' = \dfrac{\dfrac{}{\vdash t : T \to A}\quad \vdash}{\vdash tu : \mathbb{A}\langle\star_{\uparrow}\rangle (= A)} \to_{\bullet 1} \dfrac{\dfrac{}{\vdash t : T \to \mathbb{A}\langle\star_{\uparrow}\rangle}\quad \vdash}{\vdash tu : A} = s$$

  The log is unchanged. $T_{\mathsf{ext}}(s) = \bullet \cdot T_{\mathsf{ext}}(s')$. Thus $|T \to \mathbb{A}|_{\mathsf{b}} = |\mathbb{A}|_{\mathsf{b}} + 1 =_{i.h.} |T_{\mathsf{ext}}(s')|_{\mathsf{sp}} + 1 = |T_{\mathsf{ext}}(s)|_{\mathsf{sp}}$.

- Case $\to_{\bullet 2}$. Equivalent to the previous one.
- Case $\to_{\mathsf{var}}$.

$$s' = \dfrac{\dfrac{\dfrac{}{\vdash x : \mathbb{A}\langle\star_{\uparrow}\rangle_i (= A_i)}\, i}{\vdots}}{\vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle A_i\rangle \to B} \to_{\mathsf{var}} \dfrac{\dfrac{\dfrac{}{\vdash x : A_i}\, i}{\vdots}}{\vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle \mathbb{A}\langle\star_{\downarrow}\rangle_i\rangle \to B} = s$$

  Let us set $k := |\mathbb{L}| - 1$. We observe that k is exactly the number of rules T-MANY which the judgment i lies in until the judgment J corresponding to the binder. We have $s_{\mathsf{ext}}(s') = (\underline{x}, D\langle\lambda x.C\rangle, L_{\mathsf{ext}}(i) \cdot L_{\mathsf{ext}}(J), T_{\mathsf{ext}}(\mathbb{A}_i))$ and $s_{\mathsf{ext}}(s) = (\lambda x.C\langle x\rangle, \underline{D}, L_{\mathsf{ext}}(J), (x, \lambda x.C, L_{\mathsf{ext}}(i)) \cdot T_{\mathsf{ext}}(\mathbb{A}_i))$. By *i.h.* we have $k \cdot \mathsf{X} = |L_{\mathsf{ext}}(i)|_{\mathsf{sp}}$. Then $|\mathbb{L}\langle\mathbb{A}_i\rangle \to B|_{\mathsf{b}} = \mathsf{X} + k \cdot \mathsf{X} + |\mathbb{A}_i|_{\mathsf{b}} = \mathsf{X} + |L_{\mathsf{ext}}(i)|_{\mathsf{sp}} + |T_{\mathsf{ext}}(s')|_{\mathsf{sp}} = |(x, \lambda x.C, L_{\mathsf{ext}}(i))|_{\mathsf{sp}} + |T_{\mathsf{ext}}(s')|_{\mathsf{sp}} = |T_{\mathsf{ext}}(s)|_{\mathsf{sp}}$. About the log, it suffices to note that $|J|_{\mathsf{b}} =_{i.h.} |L_{\mathsf{ext}}(s)|_{\mathsf{sp}}$.

- Case $\to_{\mathsf{bt2}}$.

$$s' = \dfrac{\dfrac{\dfrac{}{\vdash x : A_i (= \mathbb{A}\langle\star\rangle_i)}\, i}{\vdots}}{\vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle\mathbb{A}\langle\star_{\uparrow}\rangle_i\rangle \to B} \to_{\mathsf{bt2}} \dfrac{\dfrac{\dfrac{}{\vdash x : \mathbb{A}\langle\star_{\downarrow}\rangle_i}\, i}{\vdots}}{\vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle A_i\rangle \to B} = s$$

  Let us set $k := |\mathbb{L}| - 1$. We observe that k is exactly the number of rules T-MANY which the judgment i lies in until the judgment J corresponding to the binder. We have $s_{\mathsf{ext}}(s') = (\underline{\lambda x.C\langle x\rangle}, D, L_{\mathsf{ext}}(J), (x, \lambda x.C, L_{\mathsf{ext}}(i)) \cdot T_{\mathsf{ext}}(\mathbb{A}_i))$ and $s_{\mathsf{ext}}(s) = (x, \underline{D\langle\lambda x.C\rangle}, L_{\mathsf{ext}}(i) \cdot L_{\mathsf{ext}}(J), T_{\mathsf{ext}}(\mathbb{A}_i))$. $|\mathbb{A}_i|_{\mathsf{b}} = |\mathbb{L}\langle\mathbb{A}\langle\star_{\uparrow}\rangle_i\rangle \to B|_{\mathsf{b}} - |\mathbb{L}| \cdot \mathsf{X} =_{i.h.} |T_{\mathsf{ext}}(s')|_{\mathsf{sp}} - |(x, \lambda x.C, L_{\mathsf{ext}}(i))|_{\mathsf{sp}} = |(x, \lambda x.C, L_{\mathsf{ext}}(i)) \cdot T_{\mathsf{ext}}(\mathbb{A}_i)|_{\mathsf{sp}} - |(x, \lambda x.C, L_{\mathsf{ext}}(i))|_{\mathsf{sp}} = |T_{\mathsf{ext}}(\mathbb{A}_i)|_{\mathsf{sp}}$. About the log, since $|L_{\mathsf{ext}}(i)|_{\mathsf{sp}} = k \cdot \mathsf{X}$ by *i.h.* and $|J|_{\mathsf{b}} =_{i.h.} |L_{\mathsf{ext}}(J)|_{\mathsf{sp}}$, then $|J\langle i\rangle|_{\mathsf{b}} = |J|_{\mathsf{b}} + k \cdot \mathsf{X} = |L_{\mathsf{ext}}(J)|_{\mathsf{sp}} + |L_{\mathsf{ext}}(i)|_{\mathsf{sp}} = |L_{\mathsf{ext}}(s)|_{\mathsf{sp}}$.

- Cases $\to_{\bullet 3}$ and $\to_{\bullet 4}$. Equivalent to case $\to_{\bullet 1}$.
- Case $\to_{\mathsf{arg}}$.

$$s' = \dfrac{\vdash t : \mathbb{L}\langle \mathbb{A}\langle \star_{\downarrow}\rangle_i\rangle \to A \qquad \dfrac{.. \vdash u : \mathbb{A}\langle\star\rangle_i ..}{\vdash u : T}\ \text{T-M}}{\vdash tu : A} \to_{\mathsf{arg}} \dfrac{\vdash t : T \to A \qquad \dfrac{.. \vdash u : \mathbb{A}\langle \star_{\uparrow}\rangle_i ..}{\vdash u : \mathbb{L}\langle \mathbb{A}\langle\star\rangle_i\rangle}\ \text{T-M}}{\vdash tu : A} = s$$

  $s_{\mathsf{ext}}(s') = (t, \underline{C}, L_{\mathsf{ext}}(s'), T_{\mathsf{ext}}(\mathbb{L}\langle \mathbb{A}_i\rangle)) = (t, \underline{C}, L_{\mathsf{ext}}(s'), l \cdot T_{\mathsf{ext}}(\mathbb{A}_i))$ and $s_{\mathsf{ext}}(s) = (\underline{u}, D, L_{\mathsf{ext}}(s), T_{\mathsf{ext}}(\mathbb{A}_i)) = (\underline{u}, D, l \cdot L_{\mathsf{ext}}(J'), T_{\mathsf{ext}}(\mathbb{A}_i))$. We have by *i.h.* $|l|_{\mathsf{sp}} + |\mathbb{A}_i|_{\mathsf{b}} = |\mathbb{L}| \cdot \mathsf{X} + |\mathbb{A}_i|_{\mathsf{b}} = |\mathbb{L}\langle \mathbb{A}_i\rangle|_{\mathsf{b}} = |T_{\mathsf{ext}}(\mathbb{L}\langle \mathbb{A}_i\rangle)|_{\mathsf{sp}} = |l \cdot T_{\mathsf{ext}}(\mathbb{A}_i)|_{\mathsf{sp}} = |l|_{\mathsf{sp}} + |T_{\mathsf{ext}}(\mathbb{A}_i)|_{\mathsf{sp}}$. About the log, we have $|J|_{\mathsf{b}} = |J'|_{\mathsf{b}} + |\mathbb{L}| \cdot \mathsf{X} =_{i.h.} |L_{\mathsf{ext}}(J')|_{\mathsf{sp}} + |l|_{\mathsf{sp}} = |L_{\mathsf{ext}}(J)|_{\mathsf{sp}}$.
- Case $\to_{\mathsf{bt1}}$. Equivalent to the previous one.

□

The Need for Tree Types. A subtle point is that the tree structure of types is not needed in order to define the extraction process—indeed, λIAM states are extracted from *sequence* type derivations in the previous chapter. Extraction is an indirect process—a sort of logical relation—whose functioning is guaranteed by an invariant (the T-exhaustible invariant in the Appendix). The process does not describe explicitly the shape of the extracted logged positions, it only guarantees that adequate logged positions exist. Without tree types, the structure of multi types derivations somehow encodes enough information to retrieve $\mathsf{ext}(s)$, but how many logged positions are involved can be discovered only by unfolding the whole extraction process, the information is not encoded into the types themselves. There is a further subtlety. Tree types trace the *number* of pointers as precisely described by the *moreover* part of Prop. 5.3.3, but do not describe the internal structure of logged positions. Given a TIAM state s, we can easily know the length of $L_{\mathsf{ext}}(s)$ and $T_{\mathsf{ext}}(s)$, and know the number of pointers to implement each logged position l in them, which is enough to measure space. The internal structure of l, however, cannot be read from tree types. Again, it is determined only by unfolding the whole extraction process.

5.3.2 The Space of the Whole Execution

Type Weights. To obtain the space cost of the whole execution we endow tree types derivations with *weights*[4]. In turn, we first have to define a notion of weight for types. The intuition is that we are taking the max of the branch size for type contexts $\mathbb{G}$ used above, over all the ways of writing a type G as $\mathbb{G}\langle\star\rangle$, as confirmed by the associated lemma.

$$\|\star\| := 0 \qquad \|T \to A\| := \max\{\|T\|, \|A\| + 1\}$$
$$\big\|[G_1, .., G_n]\big\| := \mathsf{X} + \max_i\{\|G_i\|\}$$

Lemma 5.3.4. *Let G be a type. Then $\|G\| = \max_{\mathbb{G} | G = \mathbb{G}\langle\star\rangle} |\mathbb{G}|_{\mathsf{b}}$.*

Proof. We proceed by induction on the structure of G.

- Case $G = \star$. Then there is only one $\mathbb{G}$ such that $G = \mathbb{G}\langle\star\rangle$, *i.e.* $\mathbb{G} = \langle\cdot\rangle$.

[4] We introduce a different word for measuring space of the whole execution, because judgments are measured in two different ways: the *branch size* measures what is below the judgment, and corresponds to the size of the extracted log for a state, the *weight* measures what is above a judgment, and gives the maximum space over all states in the rooted sub-derivation.

- Case $G = T \to A$. By *i.h.* $\|T\| = \max_{\mathbb{T}|T=\mathbb{T}\langle\star\rangle} |\mathbb{T}|_{\mathsf{b}}$ and $\|A\| = \max_{\mathbb{A}|A=\mathbb{A}\langle\star\rangle} |\mathbb{A}|_{\mathsf{b}}$. We have that $\{\mathbb{G}|G = \mathbb{G}\langle\star\rangle\} = \{\mathbb{T} \to A|T = \mathbb{T}\langle\star\rangle\} \cup \{T \to \mathbb{A}|A = \mathbb{A}\langle\star\rangle\}$. Then

$$\begin{aligned}\|G\| &= \|T \to A\| = \max\{\|T\|, \|A\| + 1\} = \max\left\{\max_{\mathbb{T}|T=\mathbb{T}\langle\star\rangle} |\mathbb{T}|_{\mathsf{b}}, \max_{\mathbb{A}|A=\mathbb{A}\langle\star\rangle} |\mathbb{A}|_{\mathsf{b}} + 1\right\} \\ &= \max\{\{|\mathbb{T}|_{\mathsf{b}}|T = \mathbb{T}\langle\star\rangle\}, \{|\mathbb{A}|_{\mathsf{b}} + 1|A = \mathbb{A}\langle\star\rangle\}\} \\ &= \max\{\{|\mathbb{T} \to A|_{\mathsf{b}}|T = \mathbb{T}\langle\star\rangle\}, \{|T \to \mathbb{A}|_{\mathsf{b}}|A = \mathbb{A}\langle\star\rangle\}\} = \max_{\mathbb{G}|G=\mathbb{G}\langle\star\rangle} |\mathbb{G}|_{\mathsf{b}}\end{aligned}$$

- Case $G = [G_1, .., G_n]$. By *i.h.* for each $1 \leq i \leq n, \|G_i\| = \max_{\mathbb{G}|G_i=\mathbb{G}\langle\star\rangle} |\mathbb{G}|_{\mathsf{b}}$. Then

$$\begin{aligned}\|G\| &= \left\|[G_1, ..G_n]\right\| = \mathsf{X} + \max_i\{\|G_i\|\} = \mathsf{X} + \max_i \left\{\max_{\mathbb{G}|G_i=\mathbb{G}\langle\star\rangle} |\mathbb{G}|_{\mathsf{b}}\right\} \\ &= \max_i \left\{\mathsf{X} + \max_{\mathbb{G}|G_i=\mathbb{G}\langle\star\rangle} |\mathbb{G}|_{\mathsf{b}}\right\} = \max_i \left\{\{|[G_1..\mathbb{G}_i..G_n]|_{\mathsf{b}}|G_i = \mathbb{G}_i\langle\star\rangle\}\right\} = \max_{\mathbb{G}|G=\mathbb{G}\langle\star\rangle} |\mathbb{G}|_{\mathsf{b}}\end{aligned}$$

□

Note that, via the space of single extracted states (Prop. 5.3.3), the previous lemma states that the size of G is the maximum space of all the tapes extracted from TIAM states over the same judgment $\Gamma \vdash t : G$.

Judgments and Derivations Weights. Weights are extended to judgments in Fig. 5.4, and the weight of a derivation is the weight of its final judgment. The idea is that the weight w of a weighted judgment $J = \Gamma \overset{w}{\vdash} t : G$ gives the maximum space of all the states over J and—crucially—above J. Now, we prove that the weight of a judgment J is greater than the maximum size of the tape of the states in its derivation.

Lemma 5.3.5 (Judgment weights bound extracted tapes). *Let $\pi : \Gamma \overset{w}{\vdash} t : G$ be a weighted derivation and $\mathcal{J}$ be the set of all the judgments occurring in π. Then $w \geq \max_{\Delta \vdash u:G' \in \mathcal{J}} \left\|G'\right\|$.*

Proof. We proceed by induction on π.

- Case T-Var. This case is trivial.

$$\frac{}{x : [A] \overset{\|A\|}{\vdash} x : A}\ \text{T-Var}$$

- Case T-$\lambda_\star$. Also this case is trivial, since $\|\star\| = 0$.

$$\frac{}{\Gamma \overset{0}{\vdash} \lambda x.t : \star}\ \text{T-}\lambda_\star$$

- Case T-λ. The thesis follows by the *i.h.* applied to v.

$$\frac{\Gamma, x : T \overset{v}{\vdash} t : A}{\Gamma \overset{\max\{v, \|T \to A\|\}}{\vdash} \lambda x.t : T \to A}\ \text{T-}\lambda$$

- Case T-@. The thesis follows by the *i.h.* applied to u and v and the fact that $\|A\| \leq \|T \to A\|$.

$$\frac{\Gamma \overset{u}{\vdash} t : T \to A \quad \Delta \overset{v}{\vdash} u : T}{\Gamma \uplus \Delta \overset{\max\{u,v\}}{\vdash} tu : A}\ \text{T-@}$$

- Case T-MANY. The π has the following shape.

$$\dfrac{\Gamma_i \overset{v_i}{\vdash} t : G_i \quad 1 \leq i \leq n}{[\uplus_{i=1}^{n} \Gamma_i] \overset{\mathsf{X}+\max_i\{v_i\}}{\vdash} t : [G_1, .., G_n]}\ \text{T-MANY}$$

  By *i.h.*, $v_i \geq \|G_i\|$, so w is $\geq$ of the weight of the right-hand type of any internal judgment of π. We only have to show that w also bounds the weight of $[G_1, .., G_n]$. Note that $w = \mathsf{X} + \max_i \{v_i\} \geq_{i.h.} \mathsf{X} + \max_i \{\|G_i\|\} =: \big\|[G_1, .., G_n]\big\|$.

- Case T-NONE. Trivial since $\big\|[\cdot]\big\| = 0$.

$$\dfrac{}{\overset{0}{\vdash} t : [\cdot]}\ \text{T-NONE}$$

□

Judgment weights actually take also logs into account.

Lemma 5.3.6 (Weights bound also extracted logs). *Let $\pi \triangleright \Gamma \overset{w}{\vdash} t : G$ be a weighted derivation. Then $w \geq v + |J|_{\mathsf{b}}$ for every weighted judgment $J \vdash^{v}$ in π.*

Proof. By induction on the length n of path from J to the final judgment of π. If $n = 0$ then $|J|_{\mathsf{b}} = 0$ and $w = v$, giving $w = v = v + |J|_{\mathsf{b}}$. If $n > 0$ then we look at the rule of which J is a premise. Let $J' \vdash^{v'}$ be the concluding judgment of such a rule. By *i.h.*, $w \geq v' + |J'|_{\mathsf{b}}$. Now, for all rules but T-MANY we have that $v' \geq v$ and $|J'|_{\mathsf{b}} = |J|_{\mathsf{b}}$, so that $w \geq v' + |J'|_{\mathsf{b}} \geq v + |J|_{\mathsf{b}}$. For T-MANY, we have $v' \geq \mathsf{X} + v$ and $|J'|_{\mathsf{b}} = |J|_{\mathsf{b}} - \mathsf{X}$, so that

$$w \geq v' + |J'|_{\mathsf{b}} \geq \mathsf{X} + v + |J|_{\mathsf{b}} - \mathsf{X} = v + |J|_{\mathsf{b}}.$$

□

We then obtain that the weight of a derivation π for t bounds the space used by the TIAM execution of π, and so by the λIAM execution of t.

Theorem 5.3.7 (λIAM space bounds). *Let $\pi \triangleright \overset{w}{\vdash} t : \star$ be a weighted tree types derivation. Then $|\mathsf{ext}(s)|_{\mathsf{sp}} \leq w$ for every $s \in \mathsf{states}(\pi)$.*

Proof. We prove the bound using $|s|_{\mathsf{b}}$ instead of $|\mathsf{ext}(s)|_{\mathsf{sp}}$, and obtain the statement because $|s|_{\mathsf{b}}$ instead of $|\mathsf{ext}(s)|_{\mathsf{sp}}$ by Prop. 5.3.3. Let $s = (\pi, J, \mathbb{A}, d) \in \mathsf{states}(\pi)$ be a TIAM state and let v be its weight. By Lemma 5.3.6, $w \geq |J|_{\mathsf{b}} + v$. By Lemma 5.3.5, $v \geq \big\|\mathbb{A}\langle\star\rangle\big\|$, and by Lemma 5.3.4 $\big\|\mathbb{A}\langle\star\rangle\big\| \geq |\mathbb{A}|_{\mathsf{b}}$. Then $v \geq |\mathbb{A}|_{\mathsf{b}}$. Therefore, $w \geq |J|_{\mathsf{b}} + |\mathbb{A}|_{\mathsf{b}} = |s|_{\mathsf{b}}$. □

Last, we show that weights provide *exact* bounds, as there always is a witness state using as much space as in the weight.

Proposition 5.3.8 (Weight witness). *Let $\pi \triangleright \Gamma \overset{w}{\vdash} t : G$ be a weighted derivation and $G \neq [\cdot]$. Then there exists a TIAM state s over π such that $w = |s|_{\mathsf{b}}$.*

Proof. We proceed by induction on the structure of π.

- Case T-VAR:

$$\dfrac{}{x : [A] \overset{\|A\|}{\vdash} x : A}\ \text{T-VAR}$$

  There is no log and thus $s_L = 0$, and by Lemma 5.3.4, $\|A\| = \max_{\mathbb{A}|A=\mathbb{A}\langle\star\rangle} |\mathbb{A}|_{\mathsf{b}}$.

- Case T-$\lambda_\star$:

$$\frac{}{\Gamma \overset{0}{\vdash} \lambda x.t : \star}\ \text{T-}\lambda_\star$$

  There is no log and thus $s_L = 0$, and $\|\star\| = 0 = |\langle\cdot\rangle|_{\mathsf{b}}$.

- Case T-λ:

$$\frac{\Gamma, x : T \overset{v}{\vdash} t : A}{\Gamma \overset{\max\{v,\|T\to A\|\}}{\vdash} \lambda x.t : T \to A}\ \text{T-}\lambda$$

  There are two sub-cases:

  1. $w = v \geq \|T \to A\|$: then the statement follows by the *i.h.*
  2. $w = \|T \to A\| > v$, by Lemma 5.3.4, there is a state $s = (\pi, J, \mathbb{A}, d)$ over the concluding judgment $J = \Gamma \vdash \lambda x.t : T \to A$ for which $\|T \to A\| = |\mathbb{A}|_{\mathsf{b}}$. Since for the concluding judgment $|J|_{\mathsf{b}} = 0$, we obtain

$$w = \|T \to A\| = |\mathbb{A}|_{\mathsf{b}} = |\mathbb{A}|_{\mathsf{b}} + |J|_{\mathsf{b}} = |s|_{\mathsf{b}}.$$

- Case T-@:

$$\frac{\Gamma \overset{u}{\vdash} t : T \to A \quad \Delta \overset{v}{\vdash} u : T}{\Gamma \uplus \Delta \overset{\max\{u,v\}}{\vdash} tu : A}\ \text{T-@}$$

  The thesis follows by the *i.h.* applied to u if $u \geq v$ and to v otherwise.

- Case T-MANY:

$$\frac{\Gamma_i \overset{v_i}{\vdash} t : G_i \quad 1 \leq i \leq n}{[\uplus_{i=1}^{n}\Gamma_i] \overset{\mathsf{X}+\max_i\{v_i\}}{\vdash} t : [G_1, .., G_n]}\ \text{T-MANY}$$

  Let us set $m := \max_i \{v_i\}$. Then, we apply the *i.h.* to the sub-derivation π' with weight m. Let us call $s' = (\pi', J, \mathbb{G}, d)$ the state obtained through the *i.h.*. Then $|\mathbb{G}|_{\mathsf{b}} + |J|_{\mathsf{b}} = m$. Let us now consider the same state in the new type derivation π, which includes the T-MANY rule, $s = (\pi, J, \mathbb{G}, d)$. Now $|\mathbb{G}|_{\mathsf{b}} + |J|_{\mathsf{b}} = \mathsf{X} + m = \mathsf{X} + \max_i \{v_i\}$.

- Case T-NONE:

$$\frac{}{\overset{0}{\vdash} t : [\cdot]}\ \text{T-NONE}$$

  Impossible, because by hypothesis $G \neq [\cdot]$.

□

We can then conclude our complexity analysis.

Corollary 5.3.9 (λIAM exact bound via tree types derivations)**.** *Let $\pi \triangleright \overset{w}{\vdash} t : \star$ be a tree types derivation and ρ the complete λIAM run on t. Then $|\rho|_{\mathsf{sp}} = w$.*

Proof. By Theorem 5.3.7, $|\sigma|_{\mathsf{sp}} \leq w$. By Prop. 5.3.8, there exists a state s of the TIAM over π such that $|s|_{\mathsf{b}} = w$. By Prop. 5.3.3, $|\mathsf{ext}(s)|_{\mathsf{sp}} = |s|_{\mathsf{b}}$. Therefore, $|\sigma|_{\mathsf{sp}} = w$. □

Now, the reader can fully understand and appreciate the weights in the derivation of Fig. 5.3. Please note that we have considered $\max\{1, \mathsf{X}\} = \mathsf{X}$ when assigning the weights.

Chapter 6

The IAM Seems Unreasonable

In this chapter we exploit the type-theoretic characterization of the λIAM time and space consumption given in the previous chapters. Our application is the complexity analysis of the evaluation by the λIAM of the encoding of TMs into the λ-calculus. This is needed to prove the λIAM an (un)reasonable implementation of the λ-calculus, in time and/or space. Fortunately, we do not have to enter into all the details of the encoding. We just need to analyze how the (tail) recursion needed by TMs is implemented into the λ-calculus.

TMs can be seen as the iterative (*i.e.* tail recursive) application of a transition function to the state s. Given a TM with transition function M, we could write its main loop execution as

$$\texttt{while}\ (s \text{ is not final})\ \texttt{do}\ \{M(s)\}$$

This is very easy to implement in the λ-calculus. Let t_M be the encoding of the transition function M (which is typically very simple if states and tapes are encoded using, e.g., Scott's numerals (Wadsworth, 1980)). Then, the (recursively defined) function that iterates t_M, thus capturing the overall behavior of M, can be written as follows:

$$\mathsf{iter} = \underbrace{(\lambda f.\lambda s.\texttt{if}\ s \text{ is final}\ \texttt{then halt else}\ f(t_M s))}_{\mathsf{iteraux}}\,\mathsf{iter}$$

How can we build a solution to this equation in the form of a λ-term? Apart from an encoding of the conditional operator, itself very easy to write, we need a fixed-point combinator fix, such as Turing's:

$$\mathsf{fix} := \theta\theta \qquad \text{where} \quad \theta := \lambda x.\lambda y.y(xxy)$$

We highlight that $\mathsf{fix}\, t \to_{wh} (\lambda y.y(\mathsf{fix}\, y))t \to_{wh} t(\mathsf{fix}\, t)$. Then, we can set, as expected, $\mathsf{iter} := \mathsf{fix}\,\mathsf{iteraux}$.

Let us analyze how Θ implements recursion, independently on what the argument of Θ is. Please note that during the reduction of Θt, the variable y is substituted for the term t, which after two β-steps appears twice, once in head position (call this occurrence t_0), and once applied to Θ. The latter copy of t, together with Θ, can be copied potentially many times, depending on how t_0 uses its argument. Some of these copies, say m, will eventually appear in head position, and the same process starts again. In other words, the copies of t that the combinator Θ will eventually create can be organized in a tree, see Figure 6.1a. This is a faithful description of how recursion unfolds, independently on how t uses its argument.

If $t = \mathsf{iteraux}$, however, the situation is much simpler: t uses its argument at most once (note that f is used linearly), and the complicated tree in Figure 6.1a becomes the one in Figure 6.1b. Every copy t_{01^n} of t either brings $\Theta t_{1^{n+1}}$ in head position (without copying it), or discards it, depending on whether the current state is final or not. Saying it another way, the height of the tree in Figure 6.1b is nothing more than the number of reduction steps the Turing machine performs.

6.1 Time Unreasonableness via Sequence Types

Considering this simplification, we are able to devise a (sequence) type scheme for fix. The definition is quite involuted, starting from the fact that it is parametric in n, that morally represents

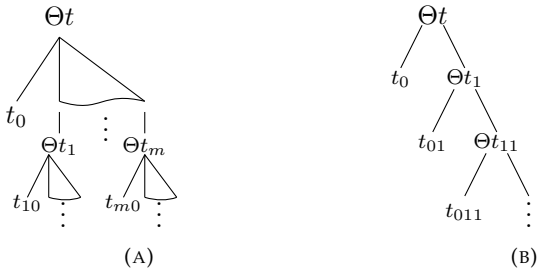


FIGURE 6.1: Different ways Θ can copy its argument.

the number of times the fix-point is unfolded. Moreover, the scheme is parameterized on a list of sequence types $\overline{A} := A_n, .., A_0$. Let us first suppose $n = 0$, *i.e.* that the recursion is never unfolded. This means that y never uses its argument xxy. Then, the type scheme is:

$$\begin{array}{rcl} \mathbb{F}_0^{\overline{A}} & := & [\mathbb{Y}_0^{\overline{A}}] \to A_0 \\ \mathbb{Y}_0^{\overline{A}} & := & [\cdot] \to A_0 \\ \mathbb{X}_0^{\overline{A}} & := & [\cdot] \to \mathbb{F}_0^{\overline{A}} \end{array}$$

Given $\mathsf{fix} := (\lambda x.\lambda y.y(xxy))\theta$, $\mathbb{F}_0$ is supposed to be the type of fix, $\mathbb{Y}_0$ the type of y, and $\mathbb{X}_0$ the type of x. If the recursion is unfolded $n+1$ times, then fix can be typed as follows:

$$\begin{array}{rcl} \mathbb{F}_{n+1}^{\overline{A}} & := & [\mathbb{Y}_{n+1}^{\overline{A}}, \ldots, \mathbb{Y}_0^{\overline{A}}] \to A_{n+1} \\ \mathbb{Y}_{n+1}^{\overline{A}} & := & [A_n] \to A_{n+1} \\ \mathbb{X}_{n+1}^{\overline{A}} & := & [\mathbb{X}_n^{\overline{A}}, \ldots, \mathbb{X}_0^{\overline{A}}] \to \mathbb{F}_{n+1}^{\overline{A}} \end{array}$$

Please note the hypothesis on the behaviour of y, that is supposed to use its argument only once.

Lemma 6.1.1. *For each $n \geq 0$ and type list $\overline{A}$ such that $|\overline{A}| \geq n+1$, $\vdash \theta : \mathbb{X}_n^{\overline{A}}$.*

Proof. Case $n = 0$.

$$\dfrac{\dfrac{\dfrac{}{y : [\mathbb{Y}_0^{\overline{A}}] \vdash y : \mathbb{Y}_0^{\overline{A}} := [\cdot] \to A_0}}{y : [\mathbb{Y}_0^{\overline{A}}] \vdash y(xxy) : A_0}}{\vdash \lambda y.y(xxy) : \mathbb{F}_0^{\overline{A}} := [\mathbb{Y}_0^{\overline{A}}] \to A_0}}{\vdash \lambda x.\lambda y.y(xxy) =: \theta : \mathbb{X}_0^{\overline{A}} := [\cdot] \to \mathbb{F}_0^{\overline{A}}}$$

Case $n+1$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : [\mathbb{X}_n^{\overline{A}}] \vdash x : \mathbb{X}_n^{\overline{A}} := [\mathbb{X}_{n-1}^{\overline{A}}, .., \mathbb{X}_0^{\overline{A}}] \to \mathbb{F}_n^{\overline{A}}} \quad \left[x : [\mathbb{X}_i^{\overline{A}}] \vdash x : \mathbb{X}_i^{\overline{A}}\right]_{i \in [0..n-1]}}{x : [\mathbb{X}_n^{\overline{A}}, .., \mathbb{X}_0^{\overline{A}}] \vdash xx : [\mathbb{Y}_n^{\overline{A}}, .., \mathbb{Y}_0^{\overline{A}}] \to A_n =: \mathbb{F}_n^{\overline{A}}} \quad \left[y : [\mathbb{Y}_i^{\overline{A}}] \vdash y : \mathbb{Y}_i^{\overline{A}}\right]_{i \in [0..n]}}}{y : [\mathbb{Y}_{n+1}^{\overline{A}}] \vdash y : \mathbb{Y}_{n+1}^{\overline{A}} := [A_n] \to A_{n+1} \qquad x : [\mathbb{X}_n^{\overline{A}}, .., \mathbb{X}_0^{\overline{A}}], y : [\mathbb{Y}_n^{\overline{A}}, .., \mathbb{Y}_0^{\overline{A}}] \vdash xxy : A_n}}{x : [\mathbb{X}_n^{\overline{A}}, .., \mathbb{X}_0^{\overline{A}}], y : [\mathbb{Y}_{n+1}^{\overline{A}}, .., \mathbb{Y}_0^{\overline{A}}] \vdash y(xxy) : A_{n+1}}}{x : [\mathbb{X}_n^{\overline{A}}, \ldots, \mathbb{X}_0^{\overline{A}}] \vdash \lambda y.y(xxy) : \mathbb{F}_{n+1}^{\overline{A}} := [\mathbb{Y}_{n+1}^{\overline{A}}, .., \mathbb{Y}_0^{\overline{A}}] \to A_{n+1}}}{\vdash \lambda x.\lambda y.y(xxy) =: \theta : \mathbb{X}_{n+1}^{\overline{A}} := [\mathbb{X}_n^{\overline{A}}, .., \mathbb{X}_0^{\overline{A}}] \to \mathbb{F}_{n+1}^{\overline{A}}}$$

□

Then, it is immediate to conclude the following proposition.

Proposition 6.1.2. *For each $n \geq 0$ and type list $\overline{A}$ such that $|\overline{A}| \geq n+1$, $\vdash \mathsf{fix} : \mathbb{F}_n^{\overline{A}}$.*

Proof.

$$\dfrac{\dfrac{}{\vdash \theta : \mathbb{X}_n^{\overline{A}} := [\mathbb{X}_{n-1}^{\overline{A}}, \ldots, \mathbb{X}_0^{\overline{A}}] \to \mathbb{F}_n^{\overline{A}}}\ \text{Lemma 6.1.1} \qquad \dfrac{\left[\vdash \theta : \mathbb{X}_i^{\overline{A}}\right]_{i\in[0..n-1]}}{}\ \text{Lemma 6.1.1}}{\vdash \mathsf{fix} := \theta\theta : \mathbb{F}_n^{\overline{A}}}$$

□

We can now return to our original problem, *i.e.* estimating the λIAM time consumption when executing the encoding of a Turing machine M which halts in n steps. We have already observed that the number of times the fix-point operator used in the encoding is unfolded is exactly n. This because the term encoding the transition function has to applied n times. This way, we have that fix is typed by $\mathbb{F}_n^{\overline{A}}$ for a suitable type list $\overline{A}$. The point now is to observe that the size of $\mathbb{F}_n^{\overline{A}}$, intended as the number of $\star$ occurrences inside it, is *exponential* in n.

Corollary 6.1.3. *For each $n+1 \geq 0$ and type list $\overline{A}$ such that $|\overline{A}| \geq n$, $\stackrel{\geq 2^n}{\vdash}\ \mathsf{fix} : \mathbb{F}_n^{\overline{A}}$.*

Proof. We partially weight the derivation of the previous proof.

$$\dfrac{\dfrac{}{\vdash^{w} \theta : \mathbb{X}_n^{\overline{A}} := [\mathbb{X}_{n-1}^{\overline{A}}, \ldots, \mathbb{X}_0^{\overline{A}}] \to \mathbb{F}_n^{\overline{A}}}\ \text{Lemma 6.1.1} \qquad \dfrac{\left[\vdash \theta : \mathbb{X}_i^{\overline{A}}\right]_{i\in[0..n-1]}}{}\ \text{Lemma 6.1.1}}{\vdash^{\geq w} \mathsf{fix} := \theta\theta : \mathbb{F}_n^{\overline{A}}}$$

Of course we have that $w \geq \left\|\mathbb{X}_n^{\overline{A}}\right\|$. We observe that $\left\|\mathbb{X}_n^{\overline{A}}\right\| \geq \sum_{0\leq i\leq n-1}\left\|\mathbb{X}_i^{\overline{A}}\right\|$. This is a Fibonacci-like recurrence and has a solution which is super-exponential. Thus $w \geq 2^n$. □

The fact that the λIAM simulates a TM M which halts in n steps in at least 2^n steps makes the number of λIAM transitions a *non* reasonable time measure. The required overhead is indeed polynomial. One could already deduce that the λIAM is not reasonable for space, too. In fact, the λIAM space consumption cannot be less that linear in n: it is well known that in order to have an exponential time, at least linear space is required. This fact contradicts the space invariance thesis, since one should have the space consumption of the λIAM linear in the *space* consumption of M, and not in *time*.

However, this last result can be proved also independently, by exploiting tree types.

6.2 Space Unreasonableness via Tree Types

In this section, we follow substantially the same path of the previous one: we give a tree type scheme to the fixed point combinator used in the encoding of TMs into the λ-calculus, and then analyze its complexity.

The definition is of course very similar to the previous one: again the scheme is parameterized by n, the number of times the fixed point is unfolded, and by a list of types $\overline{A}$. We start by defining the scheme for $n = 0$.

$$\begin{array}{rcl} \mathbb{F}_0^{\overline{A}} & := & \mathbb{T}_0^{\overline{A}} \to A_0 \\ \mathbb{T}_0^{\overline{A}} & := & [\mathbb{Y}_0^{\overline{A}}] \\ \mathbb{Y}_0^{\overline{A}} & := & [\cdot] \to A_0 \end{array}$$

We can observe that $\mathbb{Y}_0^{\overline{A}}$ has been *treefied*, and that $\mathbb{X}_0^{\overline{A}}$ has not been defined. Now, we proceed with the inductive case.

$$\begin{array}{rcl} \mathbb{F}_{n+1}^{\overline{A}} & := & \mathbb{T}_{n+1}^{\overline{A}} \to A_{n+1} \\ \mathbb{T}_{n+1}^{\overline{A}} & := & [\mathbb{Y}_{n+1}^{\overline{A}}, [\mathbb{T}_n^{\overline{A}}]] \\ \mathbb{Y}_{n+1}^{\overline{A}} & := & [A_n] \to A_{n+1} \end{array}$$

We now turn to the proof of the typability of fix by $\mathbb{F}_n^{\overline{A}}$. We need a preliminary lemma.

Lemma 6.2.1. *For each $n \geq 0$, and for each list of types $\overline{A}$ such that $|\overline{A}| \geq n+1$, $[\mathbb{T}_n^{\overline{A}}] \vdash y : \mathbb{T}_n^{\overline{A}}$.*

Proof. We proceed by induction on n. If $n = 0$, we can type y as follows:

$$\dfrac{\dfrac{}{y : [[\cdot] \to A_0] \vdash y : [\cdot] \to A_0}}{y : [[[\cdot] \to A_0]] \vdash y : [[\cdot] \to A_0]}$$

Case $n+1$:

$$\dfrac{\dfrac{}{y : [\mathbb{Y}_{n+1}^{\overline{A}}] \vdash y : \mathbb{Y}_{n+1}^{\overline{A}}} \quad \dfrac{\dfrac{\text{i.h.}}{y : [\mathbb{T}_n^{\overline{A}}] \vdash y : \mathbb{T}_n^{\overline{A}}}}{y : [[\mathbb{T}_n^{\overline{A}}]] \vdash y : [\mathbb{T}_n^{\overline{A}}]}}{y : [[\mathbb{Y}_{n+1}^{\overline{A}}, [\mathbb{T}_n^{\overline{A}}]]] \vdash y : [\mathbb{Y}_{n+1}^{\overline{A}}, [\mathbb{T}_n^{\overline{A}}]]}$$

□

Now, we able to prove the desired result.

Proposition 6.2.2. *For each $n \geq 0$, and for each list of types $\overline{A}$ such that $|\overline{A}| \geq n+1$, $\vdash \mathsf{fix} : \mathbb{F}_n^{\overline{A}}$.*

Proof. Since $\mathsf{fix} \to \lambda y.y(\mathsf{fix}\, y) =: \mathsf{fix}_2$ and types are preserved by reduction and expansion, we type fix_2. We proceed by induction on n. If $n = 0$, we can type $\mathsf{fix}_{(2)}$ as follows:

$$\dfrac{\dfrac{\dfrac{}{y : [[\cdot] \to A_0] \vdash y : [\cdot] \to A_0}}{y : [[\cdot] \to A_0] \vdash y(\mathsf{fix}\, y) : A_0}}{\vdash \mathsf{fix}_2 := \lambda y.y(\mathsf{fix}\, y) : [[\cdot] \to A_0] \to A_0}$$

Now, we prove that fix can be typed by $\mathbb{F}_{n+1}^{\overline{A}}$, knowing that by *i.h.* it can be typed with $\mathbb{F}_n^{\overline{A}}$.

$$\dfrac{\dfrac{}{y : [\mathbb{Y}_{n+1}^{\overline{A}}] \vdash y : \mathbb{Y}_{n+1}^{\overline{A}} := [A_n] \to A_{n+1}} \quad \dfrac{\dfrac{\dfrac{\text{i.h.}}{\vdash \mathsf{fix} : \mathbb{T}_n^{\overline{A}} \to A_n =: \mathbb{F}_n^{\overline{A}}} \quad \dfrac{\text{Lemma 6.2.1}}{y : [\mathbb{T}_n^{\overline{A}}] \vdash y : \mathbb{T}_n^{\overline{A}}}}{y : [\mathbb{T}_n^{\overline{A}}] \vdash \mathsf{fix}\, y : A_n}}{y : [[\mathbb{T}_n^{\overline{A}}]] \vdash \mathsf{fix}\, y : [A_n]}}{\dfrac{y : [\mathbb{Y}_{n+1}^{\overline{A}}, [\mathbb{T}_n^{\overline{A}}]] \vdash y(\mathsf{fix}\, y) : A_{n+1}}{\vdash \mathsf{fix}_2 := \lambda y.y(\mathsf{fix}\, y) : [\mathbb{Y}_{n+1}^{\overline{A}}, [\mathbb{T}_n^{\overline{A}}]] \to A_{n+1} =: \mathbb{F}_{n+1}^{\overline{A}}}}$$

□

We can now return to our original problem, *i.e.* estimating the λIAM space consumption when executing the encoding of a Turing machine M which halts in n steps and writes m cells of the tape. We have already observed that the number of times the fix-point operator used in the encoding is unfolded is exactly n. This because the term encoding the transition function has to applied n times. This way, we have that fix is typed by $\mathbb{F}_n^{\overline{A}}$ for a suitable type list $\overline{A}$. The point now is to observe that the size of $\mathbb{F}_n^{\overline{A}}$, intended as the nesting level, when it is seen as a tree, is linear in n. This means that the space consumption of the λIAM is linear in the *time* n of the encoded TM, and not in the *space* m. Of course this fact violates the space invariance thesis, and thus classifies the λIAM as an unreasonable model for both time *and* space.

Proposition 6.2.3. *For each $n \geq 0$, and for each list of types $\overline{A}$ such that $|\overline{A}| \geq n+1$, $\overset{\geq 2n\mathsf{X}}{\vdash}\ \mathsf{fix} : \mathbb{F}_n^{\overline{A}}$.*

Proof. We observe that $\left\|\mathbb{T}_n^{\overline{A}}\right\| \geq 2\mathsf{X} + \left\|\mathbb{T}_{n-1}^{\overline{A}}\right\|$, we have that $\left\|\mathbb{F}_n^{\overline{A}}\right\| \geq 2n\mathsf{X}$. Now, it suffices to observe that $\left\|\mathbb{F}_n^{\overline{A}}\right\| \geq \left\|\mathbb{T}_n^{\overline{A}}\right\| \geq 2n\mathsf{X}$. □

A last comment is about the fact that this is *not* a definitive result. In fact, our result is based on the fact that the fix-point combinator fix is used to implement a reasonable model, in our case TMs. While it is difficult to think about a simulation of Turing-complete mechanism without using a fix-point combinator[1], there may exist other exotic[2] fix-point combinators that behave better complexity-wise.

[1]There exists indeed a very simple trick to implement recursion in the λ-calculus without using a fix-point combinator, but it has the very same complexity when evaluated by the λIAM. It is actually an *inlining* of fix inside the function that one wants to implement in a recursive way.

[2]We say *exotic* because the other standard fix-point combinator, due to Curry, behaves the same. An idea, due to a private communication with Laurent Regnier, could be the use of the fix-point described by Girard (1988) in his second paper on the geometry of interaction.

Part II

The KAM: Space Complexity and Types

Chapter 7

Towards the KAM: the JAM and the PAM

In the last chapter of the previous part, we have seen that the λIAM appears to be unreasonable in both time and space. Moreover, we had already observed that the number of steps the λIAM needs to evaluate a λ-term t could be exponential in the number of steps t needs to normalize (through rewriting). This is because the λIAM recomputes many times the same paths inside the syntax tree of the term. Then, it is natural to ask if there is the possibility to optimize this mechanism as to not waste so much time. Of course, we have already observed in Chapter 2 that the Krivine Abstract Machine *is* a solution to this problem. However, we are interested in mechanisms which are directly derived from the λIAM[1].

7.1 The Jumping Abstract Machine, Revisited

The Jumping Abstract Machine (JAM) is introduced by Danos and Regnier (1999) as an optimization of the IAM obtained via a sophisticated analysis of IAM runs. Here we present the λJAM, the recasting of the JAM in the same syntactic framework of the λIAM. In particular, the λIAM and the λJAM rest on the same grammars and data structures, they only differ on some transitions.

Jumping Around the Log. The difference between the λIAM and the λJAM is in how they create logged positions, and consequently on how they backtrack. The λIAM has a *local* approach to logs, and backtracks via potentially long sequences of transitions, while the λJAM follows a *global* approach to logs, and it backtracks in just one *jump*. The transition system is presented in Fig. 7.1. The details of the two variants over the λIAM are:

- *Global logged position*: logged positions created by rule $\rightarrow_{\mathsf{var}}$ are now global, in that they record the global position of the variable, and not only the position relative to its binder. This way, also the log has to be entirely copied. Differently from the λIAM, there is some duplication of information.
- *Backtracking is short-circuited*: backtracking is a phase of a λIAM run which is contained between $\rightarrow_{\mathsf{bt1}}$ and $\rightarrow_{\mathsf{bt2}}$ transitions acting on the same logged position. It starts when the machine has to rebuild the history of a redex/substitution and ends when the substituted variable occurrence l is found. The optimization at the heart of the λJAM comes from the observation that the λIAM backtracks to the exact same state that created l. This way, one use l to jump directly to that state instead of doing the backtracking. Of course, this is possible only if positions are saved globally in logged positions: note that the λIAM saves in l only part of the log of the state creating l, while to jump back and avoid backtracking one needs to save the whole log.

[1] Actually, the KAM can be derived as an optimization of the λIAM, if one considers a variant of the λIAM based on the call-by-value translation of the λ-calculus into linear logic (or proof-nets). Interestingly, this "CbV λIAM" *still* evaluates terms under the call-by-name strategy. At this point the CbV λIAM could be optimized with jumps as we are doing in the next section for the λIAM and the resulting machine is the KAM. This was already observed by Danos and Regnier (1999)

Sub-term	Context	Log	Tape		Sub-term	Context	Log	Tape
$\underline{tu}$	C	L	T	$\to_{\bullet 1}$	$\underline{t}$	$C\langle\langle\cdot\rangle u\rangle$	L	$\bullet\cdot T$
$\underline{\lambda x.t}$	C	L	$\bullet\cdot T$	$\to_{\bullet 2}$	$\underline{t}$	$C\langle\lambda x.\langle\cdot\rangle\rangle$	L	T
$\underline{x}$	$C\langle\lambda x.D_n\rangle$	$L_n\cdot L$	T	$\to_{\mathsf{var}}$	$\lambda x.D_n\langle x\rangle$	$\underline{C}$	L	$(x, C\langle\lambda x.D_n\rangle, L_n\cdot L)\cdot T$
t	$\underline{C\langle\langle\cdot\rangle u\rangle}$	L	$\bullet\cdot T$	$\to_{\bullet 3}$	tu	$\underline{C}$	L	T
t	$\underline{C\langle\lambda x.\langle\cdot\rangle\rangle}$	L	T	$\to_{\bullet 4}$	$\lambda x.t$	$\underline{C}$	L	$\bullet\cdot T$
t	$\underline{C\langle\langle\cdot\rangle u\rangle}$	L	$l\cdot T$	$\to_{\mathsf{arg}}$	$\underline{u}$	$C\langle t\langle\cdot\rangle\rangle$	$l\cdot L$	T
t	$\underline{C\langle u\langle\cdot\rangle\rangle}$	$(x, D, L')\cdot L$	T	$\to_{\mathsf{jmp}}$	x	$\underline{D}$	L'	T

FIGURE 7.1: Transitions of the λ Jumping Abstract Machine (λJAM).

The absence of the backtracking phase makes the λJAM easier to understand than the λIAM. In particular, the $\downarrow$ and $\uparrow$ phases have now a precise meaning: the former being the quest for the head variable of the current subterm, and the latter being the search of the argument of the *only* variable occurrence in the tape. This is the second point of the following lemma.

Lemma 7.1.1 (λJAM basic invariants). *Let $s = (t, C_n, L, T, d)$ be a reachable state. Then*

1. Position and log: *(t, C_n, L) is a logged position, and*
2. Tape and direction: *if $d = \downarrow$, then T does not contain any logged position, otherwise, if $d = \uparrow$, then T contains exactly one logged position.*
3. Reversibility: *If $(t, C, L, T, d) \to_{\lambda\mathrm{IAM}} (u, D, L', T', d')$, then $(u, D, L', T', d'^1) \to_{\lambda\mathrm{IAM}} (t, C, L, T, d^1)$.*

Since the λJAM is an optimization of the λIAM, its final states have the same shape, namely $(\underline{\lambda x.u}, C, L, \epsilon)$. The fact that the log is always long enough to apply transition $\to_{\mathsf{var}}$ is given by the *position and log* invariant above. In the next section we shall prove that the λIAM and the λJAM are termination equivalent, obtaining as a corollary that the λJAM implements Closed CbN.

An Example. We present the λJAM execution trace of the same term considered for the λIAM. In particular, the first transitions are identical to the λIAM execution since no $\to_{\mathsf{var}}$ and $\to_{\mathsf{bt1}}$ rules are involved.

	Sub-term	Context	Log	Tape	Dir
	$\underline{(\lambda y.\lambda x.xy)\mathsf{II}}$	$\langle\cdot\rangle$	ϵ	ϵ	$\downarrow$
$\to_{\bullet 1}$	$\underline{(\lambda y.\lambda x.xy)\mathsf{I}}$	$\langle\cdot\rangle\mathsf{I}$	ϵ	$\bullet$	$\downarrow$
$\to_{\bullet 1}$	$\underline{\lambda y.\lambda x.xy}$	$\langle\cdot\rangle\mathsf{II}$	ϵ	$\bullet\cdot\bullet$	$\downarrow$
$\to_{\bullet 2}$	$\underline{\lambda x.xy}$	$(\lambda y.\langle\cdot\rangle)\mathsf{II}$	ϵ	$\bullet$	$\downarrow$
$\to_{\bullet 2}$	$\underline{xy}$	$(\lambda y.\lambda x.\langle\cdot\rangle)\mathsf{II}$	ϵ	ϵ	$\downarrow$
$\to_{\bullet 1}$	$\underline{x}$	$(\lambda y.\lambda x.\langle\cdot\rangle y)\mathsf{II}$	ϵ	$\bullet$	$\downarrow$

Instead, we observe that full context and log are saved at the occurrence of $\to_{\mathsf{var}}$ transitions. We set $l_x := (x, (\lambda y.\lambda x.\langle\cdot\rangle y)\mathsf{I}(\lambda z.z), \epsilon)$.

	Sub-term	Context	Log	Tape	Dir
	$\underline{x}$	$(\lambda y.\lambda x.\langle\cdot\rangle y)\mathsf{I}(\lambda z.z)$	ϵ	$\bullet$	$\downarrow$
$\to_{\mathsf{var}}$	$\lambda x.xy$	$\underline{(\lambda y.\langle\cdot\rangle)\mathsf{I}(\lambda z.z)}$	ϵ	$l_x\cdot\bullet$	$\uparrow$
$\to_{\bullet 4}$	$\lambda y.\lambda x.xy$	$\underline{\langle\cdot\rangle\mathsf{I}(\lambda z.z)}$	ϵ	$\bullet\cdot l_x\cdot\bullet$	$\uparrow$
$\to_{\bullet 3}$	$(\lambda y.\lambda x.xy)\mathsf{I}$	$\underline{\langle\cdot\rangle(\lambda z.z)}$	ϵ	$l_x\cdot\bullet$	$\uparrow$
$\to_{\mathsf{arg}}$	$\underline{(\lambda z.z)}$	$(\lambda y.\lambda x.xy)\mathsf{I}\langle\cdot\rangle$	l_x	$\bullet$	$\downarrow$
$\to_{\bullet 2}$	$\underline{z}$	$(\lambda y.\lambda x.xy)\mathsf{I}(\lambda z.\langle\cdot\rangle)$	l_x	ϵ	$\downarrow$
$\to_{\mathsf{var}}$	$\lambda z.z$	$\underline{(\lambda y.\lambda x.xy)\mathsf{I}\langle\cdot\rangle}$	l_x	$(z, (\lambda y.\lambda x.xy)\mathsf{I}(\lambda z.\langle\cdot\rangle), l_x)$	$\uparrow$

Finally, as already explained, backtracking is avoided by jumping: the λJAM restores the previously encountered state, saved in the logged position l_x, when exiting from the right-hand side of an application. We set $l_z := (z, (\lambda y.\lambda x.xy)\mathsf{I}(\lambda z.\langle\cdot\rangle), l_x)$.

	Sub-term	Context	Log	Tape	Dir
	$\lambda z.z$	$(\lambda y.\lambda x.xy)\mathsf{I}\langle\cdot\rangle$	l_x	l_z	$\uparrow$
$\to_{\mathsf{jmp}}$	x	$(\lambda y.\lambda x.\langle\cdot\rangle y)\mathsf{I}(\lambda z.z)$	ϵ	l_z	$\uparrow$
$\to_{\mathsf{arg}}$	y	$(\lambda y.\lambda x.x\langle\cdot\rangle)\mathsf{II}$	l_z	ϵ	$\downarrow$
$\to_{\mathsf{var}}$	$\lambda y.\lambda x.xy$	$\langle\cdot\rangle\mathsf{II}$	ϵ	$(y, (\lambda y.\lambda x.x\langle\cdot\rangle)\mathsf{II}, l_z)$	$\uparrow$
$\to_{\mathsf{arg}}$	I	$(\lambda y.\lambda x.xy)\langle\cdot\rangle\mathsf{I}$	$(y, (\lambda y.\lambda x.x\langle\cdot\rangle)\mathsf{II}, l_z)$	ϵ	$\downarrow$

Cost of λJAM Transitions. The cost of implementing λJAM transitions and runs on RAM is exactly the same as for the IAM: all transitions are atomic but for $\to_{\mathsf{var}}$, whose cost is given by the level n of the involved context D_n, itself bound by the size of the initial code t. Note that this means that in $\to_{\mathsf{var}}$ the duplication of the log L amounts to the duplication of the pointer to the concrete representation of L, and not of the whole of L (that would make the cost of $\to_{\mathsf{var}}$ much higher, namely depending on the length of the whole run that led to the transition).

7.2 Relating the λIAM and the λJAM: Jumping is Exhausting

In this section we prove that the λJAM is a time optimization of the λIAM via an adaptation of the exhaustible invariant. Our proof is based on the construction of a bisimulation which also provides, as a corollary, the implementation theorem for the λJAM. The basic idea is that the two machines are equivalent *modulo backtracking*. Indeed, the λJAM evaluates terms as the λIAM, but for the backtracking phase, which is short-circuited and done with just one *jump* transition. Then one has to show that the *jump* is actually simulated by the λIAM.

Log Tests. For simulating jumps we need log tests. We recall some notions from Section 3.5 The idea is that they focus on a given logged position in the log so that the *position and log* invariant (Lemma 7.1.1) is preserved. Roughly, the log test s_{l_m} focusing on the m-th logged position l_m in the log of a state $(t, C_n, l_n \cdots l_2 \cdot l_1, T, d)$ is obtained by removing the prefix $l_n \cdots l_{m+1}$ (if any), and moving the current position up by $n - m$ levels. Moreover, the tape is emptied and the direction is set to $\uparrow$.

In the argument for the simulation of jumps given below, we need only log tests of a very simple form. Namely, given a state $s = (t, C\langle u\langle\cdot\rangle\rangle, l \cdot L, T)$ from which the λJAM jumps, we shall consider the log test $s_l := (t, C\langle u\langle\cdot\rangle\rangle, l \cdot L, \epsilon)$, that is, the tape is emptied and (in this case) the position does not change. The more general form of log tests is the one already given for the λIAM, in Section 3.5.

I-Exhaustible Invariant. The λIAM exhaustible invariant proves that backtracking phases always succeed, and it is the key ingredient to relate the λIAM and the λJAM. While the underlying idea is clear, there is an important detail that has to be addressed: to establish the simulation, we have to prove that the λIAM can exhaust logged positions *of the λJAM*, rather than its own.

Since the two machines use logs differently, we have to use a function $I(\cdot)$ that maps the log-related notions of the λJAM to those of the λIAM (where Γ ranges over both logs and tapes):

$$\begin{array}{rlll} \textsc{Logged positions} & I(x, C\langle\lambda x.D_n\rangle, L_n{\cdot}L) := (x, \lambda x.D_n, I(L_n)) & & \\ \textsc{Tapes and Logs} & I(\epsilon) := \epsilon & I(l{\cdot}\Gamma) := I(l){\cdot}I(\Gamma) & I(\bullet{\cdot}T) := \bullet{\cdot}I(T) \\ \textsc{States} & I(t, C, L, T, d) := (t, C, I(L), I(T), d) & & \end{array}$$

Another point is that the state surrounding the exhausted position now is uniquely determined by the logged position. Given a logged position $l = (x, D, L)$, the *state induced by* l is $l^\circ := (x, D, L, \epsilon)$.

Definition 7.2.1 (I-Exhaustible States). *$\mathcal{E}_I$ is the smallest set of λJAM states s such that for any tape or log test s_l of s of focus l, there exists a run $\rho : I(s_l) \to^*_{\lambda\mathsf{IAM}} \to_{\mathsf{bt2},I(l)} I(l^\circ)$ such that $l^\circ \in \mathcal{E}_I$. States in $\mathcal{E}_I$ are called* I-exhaustible.

Lemma 7.2.2 (I-exhaustible invariant). *Let s be a λJAM reachable state. Then s is I-exhaustible.*

The proof, as always, is in the Appendix.

Jumping is Exhausting. From the invariant and the tape lifting property of the λIAM, it follows easily that jumps can be simulated via backtracking, from which the relationship between the λIAM and the λJAM immediately follows. We write $\to_{\mathsf{jmp},l}$ for a $\to_{\mathsf{jmp}}$ transition jumping to l.

Lemma 7.2.3 (Jumps simulation via backtracking). *Let s be a λJAM reachable state such that $s \to_{\mathsf{jmp},l} s'$. Then $I(s) \to_{\mathsf{bt1},I(l)} \to^*_{\lambda\mathrm{IAM}} \to_{\mathsf{bt2},I(l)} I(s')$.*

Proof. Let $l := (x, D, L')$ and consider $s = (t, C\langle u\langle\cdot\rangle\rangle, (x, D, L') \cdot L, T) \to_{\mathsf{jmp},l} (x, \underline{D}, L', T) = s'$. Since s is reachable then it is I-exhaustible, so its log test $s_l := (t, C\langle u\langle\cdot\rangle\rangle, l \cdot L, \epsilon)$ can be exhausted, that is, there is a λIAM run $\rho : I(s_l) \to^*_{\lambda\mathrm{IAM}} \to_{\mathsf{bt2},I(l)} I(x, \underline{D}, L', \epsilon) = s''$. Note that the first transition of ρ is necessarily $\to_{\mathsf{bt1},I(l)}$. Moreover, $I(s_l)$ and s'' are exactly $I(s)$ and $I(s')$ with empty tape. We lift ρ to a run $\rho^{I(T)} : I(s_l)^{I(T)} \to_{\mathsf{bt1},I(l)} \to^*_{\lambda\mathrm{IAM}} \to_{\mathsf{bt2},I(l)} s''^{I(T)}$ using Lemma 3.3.6. Now, $\rho^{I(T)}$ is exactly the λIAM simulation of the jump, because $I(s_l)^{I(T)} = I(s)$ and $s''^{I(T)} = I(s)$. □

From the lemma it easily follows a bisimulation between the λIAM and the λJAM, showing that the latter is faster.

Theorem 7.2.4 (λIAM and λJAM relationship).

1. λJAM to λIAM: *for every λJAM run $\rho_J : s_t^{\lambda JAM} \to^*_{\lambda\mathrm{JAM}} s$ there exists a λIAM run $I(\rho_J) : I(s_t^{\lambda JAM}) \to^*_{\lambda\mathrm{IAM}} I(s)$ such that $|\rho_I| \geq |\rho_J|$ and $|\rho_I|_{\mathsf{var}} \geq |\rho_J|_{\mathsf{var}}$.*
2. λIAM to λJAM: *for every λIAM run $\rho_I : s_t^{\lambda IAM} \to^*_{\lambda\mathrm{IAM}} s$ there exist a λJAM run $\rho_J : s_t^{\lambda JAM} \to^*_{\lambda\mathrm{JAM}} s'$ and a λIAM run $\sigma_I : s \to^*_{\lambda\mathrm{IAM}} I(s')$ such that $\rho_I\sigma_I = I(\rho_J)$.*
3. Termination and λJAM implementation: *$\lambda IAM(t)\Downarrow$ if and only if $\lambda JAM(t)\Downarrow$. Therefore, the λJAM implements Closed CbN.*

Proof.

1. We proceed by induction on the length of ρ_J. If $|\rho_J| = 0$ there is nothing to prove. Now, let us consider $\rho_J : s_t \to^*_{\lambda\mathrm{JAM}} s' \to_{\lambda\mathrm{JAM}} s$. Considering the property true for the reduction $\sigma_J : s_t \to^*_{\lambda\mathrm{JAM}} s'$, we prove that it is true for ρ_J. In particular, there exists a reduction $\sigma_I : I(s_t) \to^*_{\lambda\mathrm{IAM}} I(s')$ such that $|\sigma_I| \geq |\sigma_J|$ and $|\sigma_I|_{\mathsf{var}} \geq |\sigma_J|_{\mathsf{var}}$. We proceed considering all the possible transitions from s' to s.
   - Transitions $\to_{\bullet 1}, \to_{\bullet 2}, \to_{\bullet 3}, \to_{\bullet 4}, \to_{\mathsf{arg}}$. This group of transitions behaves identically, modulo $I(\cdot)$ in the two machines. Then $|\rho_J| = 1 + |\sigma_J| \leq_{i.h.} 1 + |\sigma_I| = |\rho_I|$ and $|\rho_J|_{\mathsf{var}} = |\sigma_J|_{\mathsf{var}} \leq_{i.h.} |\sigma_I|_{\mathsf{var}} = |\rho_I|_{\mathsf{var}}$.
   - Transition $\to_{\mathsf{var}}$. This transition behaves identically, modulo $I(\cdot)$, in the two machines. Therefore, $|\rho_J| = 1 + |\sigma_J| \leq_{i.h.} 1 + |\sigma_I| = |\rho_I|$, and exactly the same sequence of (in)equalites holds with respect to $|\cdot|_{\mathsf{var}}$.
   - Transition $\to_{\mathsf{jmp}}$. This is the only non trivial case. If $s \to_{\mathsf{jmp},l} s'$ then by the simulation of jumps via backtracking (Lemma 7.2.3) we have a run $\pi_I : I(s) \to_{\mathsf{bt1},I(l)} \to^*_{\lambda\mathrm{IAM}} \to_{\mathsf{bt2},I(l)} I(s')$. Then we define ρ_I as σ_I followed by π_I, so that $|\rho_J| = 1 + |\sigma_J| \leq_{i.h.} 1 + |\sigma_I| < |\pi_I| + |\sigma_I| = |\rho_I|$ and $|\rho_J|_{\mathsf{var}} = |\sigma_J|_{\mathsf{var}} \leq_{i.h.} |\sigma_I|_{\mathsf{var}} \leq |\pi_I|_{\mathsf{var}} + |\sigma_I|_{\mathsf{var}} = |\rho_I|_{\mathsf{var}}$.
2. By induction on the length of ρ_I. If $|\rho_I|_{\mathtt{t}} = 0$ there is nothing to prove. Now, let us consider $\rho_I : s_t \to^*_{\lambda\mathrm{IAM}} s_1 \to_{\lambda\mathrm{IAM}} s$. Considering the property true for the reduction $\rho'_I : s_t \to^*_{\lambda\mathrm{IAM}} s_1$, we prove that it is true for ρ_I. By *i.h.*, there are runs $\rho'_J : s_t \to^*_{\lambda\mathrm{JAM}} s_2$ and $\sigma'_I : s_1 \to^*_{\lambda\mathrm{IAM}} I(s_2)$ such that $\rho'_I\sigma'_I = I(\rho'_J)$. If σ'_I is non-empty then by determinism of the λIAM we are done, because σ'_I has to pass through s and the suffix σ_I of σ'_I starting on s proves the statement. If σ'_I is empty then $s_1 = I(s_2)$. Then consider the cases of transition $s_1 \to_{\lambda\mathrm{IAM}} s$:

- Transitions $\to_{\bullet 1}, \to_{\bullet 2}, \to_{\bullet 3}, \to_{\bullet 4}, \to_{\mathsf{arg}}, \to_{\mathsf{var}}$. The λJAM can do the same step and close the diagram, as these transitions behaves identically, modulo $I(\cdot)$, in the two machines.
- Transition $\to_{\mathsf{bt2}}$. Impossible because then the state $s_1 = I(s_2)$ would have direction $\downarrow$, have a logged position on the tape, and be the projection of a λJAM state—by the direction and tape invariant of the λJAM such states have no logged positions on the tape.
- Transition $\to_{\mathsf{bt1}}$. Then the λJAM can make a jump and we can close the diagram using the simulation of jumps via backtracking (Lemma 7.2.3), as in the previous point of the theorem.

3. The first two points of the theorem provide the proof that I is a bisimulation between the λIAM and the λJAM. Clearly, I preserves termination.

□

Corollary 7.2.5 (λIAM and λJAM relationship)**.** *There is a complete λJAM run ρ_J from t if and only if there is a complete λIAM run ρ_I from t. In particular, the λJAM implements Closed CbN. Moreover, $|\rho_J| \leq |\rho_I|$ and $|\rho_J|_{\mathsf{var}} \leq |\rho_I|_{\mathsf{var}}$.*

Exponential Gap. The time gap between the λIAM and the λJAM can be exponential, as it is shown by the family of terms t_n ($t_1 := \mathsf{I}$ and $t_{n+1} := t_n\mathsf{I}$) mentioned in the introduction. The results of this section provide a nice high-level proof. Next section shows that the time of the λJAM is polynomial in the time of the KAM, that takes time polynomial in the number of β-steps to evaluate t_n, that is, n. The study of multi types in Sect. 4.1 instead shows that the time of the λIAM depends on the size of the smallest type A_n of t_n, which is easily seen to be exponential in n. In fact, using the notation of Sect. 4.1, $A_1 := \star$, and $A_{n+1} := [A_n] \to A_n$.

7.3 Entangling the λJAM and the KAM: the HAM

Now we turn to the relationship between the λJAM and the KAM. We prove that KAM runs can be obtained from λJAM ones via *hops* that short-circuit the search for arguments realized by the blue transitions. It then follows that the KAM can be seen as a time improvement of the λJAM.

The HAM. To prove that the KAM is a time improvement of the λJAM, we introduce an intermediate machine, the *Hopping Abstract Machine* (HAM) in Fig. 7.2, that merges the two. The HAM is a technical tool addressing an inherent difficulty: the λJAM and the KAM use different data structures and it is impossible to turn a KAM state into a λJAM state without having to look at the whole run that led to that state, as it is instead possible for the λJAM and the λIAM.

The idea behind the HAM is to entangle the data structures of both machines so that their states get paired by construction, and to allow it to behave non-deterministically either as the λJAM or as the KAM. The HAM deals with two enriched objects, *logged closures* $\hat{c}$ and *closed (logged) positions* $\hat{l}$ (defined in Fig. 7.2, overloading some of the notations of the previous sections), obtained by adding a log to closures and an environment to logged positions. Of course, environments and logs have to be redefined as containing these enriched objects. There is also a *(closed) tape* T, that is, a data structure obtained by merging the roles of the stack and the tape and containing both logged closures and closed positions. In fact the closed tape is obtained from the λJAM tape by upgrading every $\bullet$ entry to a logged closure $\hat{c}$, and every logged position l to a closed one $\hat{l}$. Logged closures and closed positions contain the same information (a term, a context, a log, and an environment) but they play different roles.

The non-determinism of the machine amounts to the presence of *two* transitions $\to_{\mathsf{varJ}}$ and $\to_{\mathsf{hop/varK}}$ for the variable case, that are simply the var transitions of the λJAM and the KAM, lifted to the new data structures. In particular, transition $\to_{\mathsf{hop/varK}}$ short-circuits a whole $\uparrow$ phase of the λJAM *hopping* directly to the argument.

It is evident that by removing environments, turning every logged closure into $\bullet$, and removing $\to_{\mathsf{hop/varK}}$ we obtain the λJAM. Similarly, by removing logs, $\to_{\mathsf{varJ}}$, and the $\uparrow$ transitions, one

LOG. CLOS.	$\hat{c} ::= (t, C_{n+1}, E)^{L_n}$	ENV.	$E ::= \epsilon \mid [x \leftarrow \hat{c}] \cdot E$
LOGS	$L_0 ::= \epsilon \quad L_{n+1} ::= \hat{l} \cdot L_n$	CL. POSITIONS	$\hat{l} ::= (t, C_n, L_n)^E$
CL. TAPES	$T ::= \epsilon \mid \hat{c} \cdot T \mid \hat{l} \cdot T$	STATES	$s ::= (t, C, L, E, T, d)$

Term	Ctx	Log	Env	Cl. Tape		Term	Ctx	Log	Env	Cl. Tape
$\underline{tu}$	C	L	E	T	$\to_{\bullet 1/\mathsf{app}}$	$\underline{t}$	$C\langle\langle\cdot\rangle u\rangle$	L	E	$\hat{c}\cdot T$
$\underline{\lambda x.t}$	C	L	E	$\hat{c}\cdot T$	$\to_{\bullet 2/\mathsf{abs}}$	$\underline{t}$	$C\langle\lambda x.\langle\cdot\rangle\rangle$	L	$[x\leftarrow\hat{c}]\cdot E$	T
$\underline{x}$	$C\langle\lambda x.D_n\rangle$	$L_n\cdot L$	$E'\cdot[x\leftarrow\hat{c}]\cdot E$	T	$\to_{\mathsf{varJ}}$	$\lambda x.D_n\langle x\rangle$	$\underline{C}$	L	E	$\hat{l}\cdot T$
$\underline{x}$	C	L	E	T	$\to_{\mathsf{hop/varK}}$	$\underline{u}$	D	$\hat{l}\cdot L'$	F	T
where	$\hat{c} := (u, C\langle t\langle\cdot\rangle\rangle, E)^L$ in $\to_{\bullet 1/\mathsf{app}}$ $\hat{l} := (x, C\langle\lambda x.D\rangle, L_n\cdot L)^{E'\cdot[x\leftarrow\hat{c}]\cdot E}$ in $\to_{\mathsf{varJ}}$ $E = E'\cdot[x\leftarrow(u, D, F)^{L'}]\cdot E''$ and $\hat{l} = (x, C, L)^E$ in $\to_{\mathsf{hop/varK}}$.									
t	$\underline{C\langle\langle\cdot\rangle u\rangle}$	L	E	$\hat{c}\cdot T$	$\to_{\bullet 3}$	tu	$\underline{C}$	L	E	T
t	$\underline{C\langle\lambda x.\langle\cdot\rangle\rangle}$	L	$[x\leftarrow\hat{c}]\cdot E$	T	$\to_{\bullet 4}$	$\lambda x.t$	$\underline{C}$	L	E	$\hat{c}\cdot T$
t	$\underline{C\langle\langle\cdot\rangle u\rangle}$	L	E	$\hat{l}\cdot T$	$\to_{\mathsf{arg}}$	$\underline{u}$	$C\langle t\langle\cdot\rangle\rangle$	$\hat{l}\cdot L$	E	T
t	$\underline{C\langle u\langle\cdot\rangle\rangle}$	$\hat{l}\cdot L$	E	T	$\to_{\mathsf{jmp}}$	x	$\underline{D}$	L'	E'	T
where in the last transition $\hat{l} = (x, D, L')^{E'}$.										

FIGURE 7.2: Data structures and transitions of the Hopping Abstract Machine (HAM).

obtains the KAM. We avoid spelling out these immediate projections. Instead, we see KAM runs inside the HAM as given by the transition $\to_{\mathrm{HAM_K}} := \to_{\bullet 1/\mathsf{app}} \cup \to_{\bullet 2/\mathsf{abs}} \cup \to_{\mathsf{hop/varK}}$. Similarly, the λJAM is seen as transition $\to_{\mathrm{HAM_J}}$, defined as the union of all HAM transitions but $\to_{\mathsf{hop/varK}}$.

The HAM verifies the same basic properties of the λJAM, simply lifted to the enriched data structures. Moreover, it verifies a tape lifting property.

Lemma 7.3.1 (HAM tape lift). *Let $\rho : s = (t, C, L, E, T', d) \to^n_{\mathrm{HAM}} (u, D, L', E', T'', d') = s'$ be a run and T be a tape. Then there is a run $\rho^T : s^T = (t, C, L, E, T'\cdot T, d) \to^n_{\mathrm{HAM}} (u, D, L', E', T''\cdot T, d') = s'^T$.*

7.4 Hopping is Also Exhausting

Since jumping and hopping amount to a similar idea, the proof technique that we use to relate the λJAM and the KAM is obtained by another variant on the exhaustible invariant.

Testing Logged Closures. The main difference is that now we exhaust *logged closures* instead of logged positions. Via the $\uparrow$-exhaustible invariant below we shall show that the HAM can exhaust a logged closure—that is it can recover the argument in the closure—by using only λJAM $\uparrow$ transitions. This capability shall then be used to show that the λJAM can simulate hops.

Since logged closures are both in the environment and in the tape, we have two kinds of test. They are essential for the proof of the $\uparrow$-exhaustible invariant (in the Appendix), but they are not needed for the argument at work in the simulation, spelled out below.

Environment Tests. Given a HAM state (t, C, L, E, T, d) consider an entry $[x\leftarrow\hat{c}]$ in E. The idea is that one wants to exhaust $\hat{c}$ to return to the state saved in $\hat{c}$. Remember that the λJAM looks for the argument starting from the binder of x. Then, the test associated to $\hat{c}$ is obtained by positioning the machine on the binder λx for x, and modifying the log and the environment accordingly. Moreover, the tape is emptied.

Definition 7.4.1 (HAM environment tests). *Let $s = (t, C\langle\lambda x.D_n\rangle, L_n\cdot L, E'\cdot[x\leftarrow\hat{c}]\cdot E, T, d)$ be a state. Then, $s_{\hat{c}} := (\lambda x.D_n\langle t\rangle, \underline{C}, L, E, \epsilon)$ is an environment test for s of focus $\hat{c}$.*

As in the previous section, we need a notion of state induced by a logged closure $\hat{c}$, that is the state reached by a run exhausting $\hat{c}$. The definition may seem wrong, an explanation follows.

Definition 7.4.2 (HAM state induced by a logged closure). *Given a logged closure $\hat{c} = (u, D\langle t\langle\cdot\rangle\rangle, E)^L$, the state $\hat{c}^\circ$ induced by $\hat{c}$ is defined as $\hat{c}^\circ := (t, \underline{D\langle\langle\cdot\rangle u\rangle}, L, E, \epsilon)$.*

The previous definition is counter-intuitive, as one would expect $\hat{c}^\circ$ to rather be the state $s' := (\underline{u}, D\langle t\langle\cdot\rangle\rangle, L, E, \epsilon)$, but for technical reasons this is not possible. In the simulations of hops below, however, $\hat{c}^\circ$ is tape lifted to a state that makes a $\to_{\mathsf{arg}}$ transition to (a tape lifting of) s', as one would expect. We set $\to_\uparrow := \to_{\bullet 3, \bullet 4, \mathsf{arg}, \mathsf{jmp}}$.

Definition 7.4.3 (HAM $\uparrow$-Exhaustible states). *$\mathcal{E}_\uparrow$ is the smallest set of those states s such that for any tape or environment test $s_{\hat{c}}$ of s, there exists a run $\rho_\uparrow : s_{\hat{c}} \to_\uparrow^* \hat{c}^\circ$ and $\hat{c}^\circ \in \mathcal{E}_\uparrow$. States in $\mathcal{E}_\uparrow$ are called* $\uparrow$-exhaustible *(pronounced* up*-exhaustible).*

Lemma 7.4.4. *Let s be a HAM reachable state. Then s is $\uparrow$-exhaustible.*

Simulating Hops. From the invariant and the tape lifting property of the HAM, it follows easily that hops can be simulated via $\to_\uparrow$, as the next lemma shows.

Lemma 7.4.5 (Hops simulation via $\uparrow$). *Let s be a HAM reachable state and $s \to_{\mathsf{hop/varK}} s'$. Then $s \to_{\mathsf{varJ}} \to_\uparrow^* \to_{\mathsf{arg}} s'$.*

Proof. The hypothesis is: $s = (\underline{x}, C, L, E, T) \to_{\mathsf{hop/varK}} (\underline{u}, D\langle t\langle\cdot\rangle\rangle, \hat{l}\cdot L', F, T) = s'$ where $E = E'\cdot[x\leftarrow\hat{c}]\cdot E''$ with $\hat{c} = (u, D\langle t\langle\cdot\rangle\rangle, F)^{L'}$ and $\hat{l} := (x, C, L)^E$. From s the HAM can also do a $\to_{\mathsf{varJ}}$ transition: $s = (\underline{x}, C'\langle\lambda x.D'_n\rangle, L_n\cdot L'', E'[x\leftarrow\hat{c}]E'', T) \to_{\mathsf{varJ}} (\lambda x.D'_n\langle x\rangle, \underline{C'}, L'', E'', \hat{l}\cdot T) =: s''$ where $L = L_n\cdot L''$ and $C = C'\langle\lambda x.D'_n\rangle$. Now consider the environment test $s'_{\hat{c}} = (\lambda x.D_n\langle x\rangle, \underline{C}, L'', E'', \epsilon)$. By $\uparrow$-exhaustibility we obtain $\rho : s'_{\hat{c}} \to_\uparrow^* \hat{c}^\circ = (t, \underline{D\langle\langle\cdot\rangle u\rangle}, L', F, \epsilon)$ Then, lifting $\hat{c}^\circ$ with the tape $\hat{l}\cdot T$, one has $\hat{c}^\circ_{\hat{l}\cdot T} = (t, \underline{D\langle\langle\cdot\rangle u\rangle}, L', F, \hat{l}\cdot T) \to_{\mathsf{arg}} (\underline{u}, D\langle t\langle\cdot\rangle\rangle, \hat{l}\cdot L', F, T)$ Thus, $s \to_{\mathsf{varJ}} s'' \to_\uparrow^* \hat{c}^\circ_{\hat{l}\cdot T} \to_{\mathsf{arg}} s'$. □

From the lemma it easily follows a bisimulation between the λJAM and the KAM, showing that the latter is faster.

Theorem 7.4.6 (λJAM and KAM relationship via the HAM). *Let s_t be a HAM initial state.*

1. KAM to λJAM*: for every run $\rho_K : s_t \to^*_{\mathrm{HAM}_{\mathsf{K}}} s$ there exists a run $J(\rho_K) : s_t \to^*_{\mathrm{HAM}_{\mathsf{J}}} s$ such that $|J(\rho_K)| = |\rho_K| + |J(\rho_K)|_\uparrow$ and $|J(\rho_K)|_{\mathsf{varJ}} = |\rho_K|_{\mathsf{hop/varK}}$.*
2. λJAM to KAM*: for every run $\rho_J : s_t \to^*_{\mathrm{HAM}_{\mathsf{J}}} s$ there exist a run $\rho_K : s_t \to^*_{\mathrm{HAM}_{\mathsf{K}}} s'$ and a run $\sigma_J : s \to_\uparrow^* s'$ such that $\rho_J\sigma_J = J(\rho_K)$.*
3. Termination*: $\to_{\mathrm{HAM}_{\mathsf{K}}}$ terminates if and only if $\to_{\mathrm{HAM}_{\mathsf{J}}}$ terminates.*

Proof.

1. We proceed by induction on the length of ρ_K. If $|\rho_K| = 0$ there is nothing to prove. Now, let us consider $\rho_K : s_t \to^*_{\mathrm{HAM}_{\mathsf{K}}} s' \to_{\mathrm{HAM}_{\mathsf{K}}} s$. Considering the property true for the reduction $\sigma_K : s_t \to^*_{\mathrm{HAM}_{\mathsf{K}}} s'$, we prove that it is true for ρ_K. In particular, there exists a reduction $J(\sigma_K) : s_t \to^*_{\mathrm{HAM}_{\mathsf{J}}} s'$ such that $|J(\sigma_K)| = |\sigma_K| + |J(\sigma_K)|_\uparrow$ and $|J(\sigma_K)|_{\mathsf{varJ}} = |\sigma_K|_{\mathsf{hop/varK}}$. We proceed considering all the possible $\to_{\mathrm{HAM}_{\mathsf{K}}}$ transitions from s' to s.
   - Transitions $\to_{\bullet 1/\mathsf{app}}$ and $\to_{\bullet 2/\mathsf{abs}}$. These transitions belong also to $\to_{\mathrm{HAM}_{\mathsf{J}}}$, so the statement trivially holds. In particular, $|J(\rho_K)| = 1 + |J(\sigma_K)| =_{i.h.} 1 + |\sigma_K| + |J(\sigma_K)|_\uparrow = |\rho_K|$.

- Transition $\to_{\mathsf{hop/varK}}$. By Lemma 7.4.5, we have a run $\pi : s' \to_{\mathsf{varJ}} s'' \to_{\uparrow}^{+} s$. Then we define $J(\rho_k)$ as the concatenation of $J(\sigma_k)$ and π, for which

$$|J(\rho_k)| = |J(\sigma_k)| + 1 + |\pi| =_{i.h.} |\sigma_K| + |J(\sigma_k)|_{\uparrow} + 1 + |\pi_J| = |\sigma_K| + 1 + |J(\rho_k)|_{\uparrow} = \\ = |\rho_K| + |J(\rho_k)|_{\uparrow}$$

  and $|J(\rho_k)|_{\mathsf{varJ}} = 1 + |J(\sigma_k)|_{\mathsf{varJ}} =_{i.h.} 1 + |\sigma_K|_{\mathsf{hop/varK}} = |\rho_K|_{\mathsf{hop/varK}}$.

2. By induction on the length of ρ_J. If $|\rho_J| = 0$ there is nothing to prove. Now, let us consider $\rho_J : s_t \to_{\mathrm{HAM_J}}^{*} s_1 \to_{\mathrm{HAM_J}} s$. Considering the property true for the reduction $\rho_J' : s_t \to_{\mathrm{HAM_J}}^{*} s_1$, we prove that it is true for ρ_J. By *i.h.*, there are runs $\rho_K' : s_t \to_{\mathrm{HAM_K}}^{*} s_2$ and $\sigma_J' : s_1 \to_{\uparrow}^{*} s_2$ such that $\rho_J'\sigma_J' = J(\rho_K')$. If σ_J' is non-empty then by determinism of the λJAM we are done, because σ_J' has to pass through s and the suffix σ_J of σ_J' starting on s proves the statement. If σ_J' is empty then $s_1 = s_2$ and in particular s_2 has direction $\downarrow$, because it is reached by $\to_{\mathrm{HAM_K}}$. Then consider the cases of transition $s_1 \to_{\lambda\mathrm{JAM}} s$:

   - Transitions $\to_{\bullet 1/\mathsf{app}}$ and $\to_{\bullet 2/\mathsf{abs}}$. These transitions belong also to $\to_{\mathrm{HAM_K}}$, so $\to_{\mathrm{HAM_K}}$ can do the same step and close the diagram.
   - Transition $\to_{\mathsf{varJ}}$. Then we can close the diagram via the reasoning used at the previous point of the theorem, based on Lemma 7.4.5.

3. Two directions:

   - $\to_{\mathrm{HAM_K}}$ *termination implies* $\to_{\mathrm{HAM_J}}$ *termination*: an omitted standard invariant ensures that if terms are closed then, whenever the code is a variable x, the environment is defined on x. This fact forbids $\to_{\mathrm{HAM_K}}$ to get stuck on $\to_{\mathsf{hop/varK}}$ transitions. So $\to_{\mathrm{HAM_K}}$ final states have the shape $(\underline{\lambda x.t}, C, L, E, \epsilon)$, which are also $\to_{\mathrm{HAM_J}}$ final states. Then if $\to_{\mathrm{HAM_K}}$ terminates $\to_{\mathrm{HAM_J}}$ terminates.
   - $\to_{\mathrm{HAM_J}}$ *termination implies* $\to_{\mathrm{HAM_K}}$ *termination*: we prove the contrapositive statement. Suppose that $\to_{\mathrm{HAM_K}}$ diverges starting from s_t. Note that it has to make an infinity of $\to_{\mathsf{hop/varK}}$ transitions, because without them—that is considering only $\to_{\bullet 1/\mathsf{app}}$ and $\to_{\bullet 2/\mathsf{abs}}$—the size of the code strictly decreases. By the first point of the theorem, projecting the diverging $\to_{\mathrm{HAM_K}}$ run we obtain a diverging $\to_{\mathrm{HAM_J}}$ run, because the projection maps the infinity of $\to_{\mathsf{hop/varK}}$ transitions to an infinity of $\to_{\mathsf{varJ}}$ transitions.

□

Corollary 7.4.7 (λJAM and KAM relationship)**.** *There is a complete λJAM run ρ_J from t if and only if there is a complete KAM run ρ_K from t. Moreover, $|\rho_J| = |\rho_K| + |\rho_J|_{\uparrow}$ and $|\rho_J|_{\mathsf{var}} = |\rho_K|_{\mathsf{var}}$.*

Proof. It follows immediately from the previous theorem by the two obvious (and omitted) strong bisimulations between the KAM and the transition subrelation $\to_{\mathrm{HAM_K}}$ of the HAM, and between the λJAM and the transition subrelation $\to_{\mathrm{HAM_J}}$ of the HAM. □

7.5 The λJAM is Slowly Reasonable

In this section we provide bounds for the complexity of the λJAM. First, we show that it is quadratically slower than the KAM, and then, by using results from the literature about the KAM, we obtain bounds with respect to the two parameters for complexity analyses of abstract machines, namely, the size $|t|$ of the evaluated term and the number $\#\beta$ of $\to_{wh}$-steps to evaluate t.

Locating the λJAM. We have proved in the previous two sections that a run ρ_J of the λJAM from t is such that $|\rho_K| \leq |\rho_J| \leq |\rho_I|$, where ρ_K and ρ_I are the runs from t respectively of the KAM and of the λIAM. However, this tells nothing about the inherent complexity of evaluating a term with the λJAM. In fact, it is well known that $|\rho_K|$ is polynomial in $\#\beta$ and $|t|$ (namely quadratic in $\#\beta$ and linear in $|t|$), while $|\rho_I|$ can be exponential in both $\#\beta$ and $|t|$ (the typical example of

exponential behavior being the family of terms t_n defined as $t_1 := \mathsf{I}$ and $t_{n+1} := t_n\mathsf{I}$). What about the λJAM? Is it polynomial or exponential? It turns out that the λJAM is polynomial, and precisely at most quadratically slower than the KAM.

Bounding ↑ Phases. Since the KAM is the λJAM less the (blue) ↑ phases, and the complexity of the KAM is known, we only have to study the length of ↑ phases. The length of a ↑ phase extending a run ρ from t is bound by $|\rho|_{\mathsf{var}} \cdot |t|$, and the length of all ↑ phases together is bound by $|\rho|^2_{\mathsf{var}} \cdot |t|$. The proof is in three steps. First, we show that in absence of jumps a ↑ phase cannot be longer than $|t|$. An immediate induction on $|C|$ proves the following lemma.

Lemma 7.5.1. *Let* $\rho : (t, \underline{C}, L, T) \to^*_{\bullet 3, \bullet 4} s$. *Then* $|\rho| \leq |C| \leq |C\langle t\rangle|$.

Second, we need an invariant. To estimate the number of jumps in a ↑ phase, we need to link the structure of logs with the number of $\to_{\mathsf{var}}$ transitions encountered so far. We introduce the notion of *depth* of a tape/log Γ, defined in the following way:

$$\begin{array}{rclcrcl} \mathsf{depth}(\epsilon) & := & 0 & \qquad & \mathsf{depth}(\bullet \cdot T) & := & \mathsf{depth}(T) \\ \mathsf{depth}(l \cdot \Gamma) & := & \mathsf{depth}(l) & & \mathsf{depth}((x, C, L)) & := & 1 + \mathsf{depth}(L) \\ \mathsf{depth}(t, C, L, T, \uparrow) & := & \mathsf{depth}(T) & & \mathsf{depth}(t, C, L, T, \downarrow) & := & \mathsf{depth}(L) \end{array}$$

Proposition 7.5.2 (Depth invariant). *Let* $\rho : s_t \to^*_{\lambda\mathrm{JAM}} s$ *be an initial run of the λJAM. Then* $\mathsf{depth}(s) = |\rho|_{\mathsf{var}}$. *Moreover* $\mathsf{depth}(s) \geq \mathsf{depth}(l)$ *for every logged position* l *in* s.

Proof. We proceed by induction on the length of the run ρ. If $|\rho| = 0$, then $s = s_t$ and $\mathsf{depth}(s_t) = \mathsf{depth}(\underline{t}, C, \epsilon, \epsilon) = \mathsf{depth}(\epsilon) = 0 = |\rho|_{\mathsf{var}}$. If $|\rho| \geq 1$, let σ be the prefix of ρ such that $s_t \to^*_{\lambda\mathrm{JAM}} s'$, and let's consider the various cases of the last transition $s' \to_{\lambda\mathrm{JAM}} s$:

- Transitions $\to_{\bullet 1}$ or $\to_{\bullet 2}$: the result holds by *i.h.*, since the direction has not changed and neither the depth of the log.
- Transition $\to_{\mathsf{var}}$:

  $$s' = (\underline{x}, C\langle\lambda x.D_n\rangle, L_n \cdot L, T) \to_{\mathsf{var}} (\lambda x.D_n\langle x\rangle, \underline{C}, L, (x, C\langle\lambda x.D_n\rangle, L_n \cdot L) \cdot T) = s$$

  Then $\mathsf{depth}(s) = \mathsf{depth}(L_n \cdot L) + 1 = \mathsf{depth}(s') + 1 =_{i.h.} |\sigma|_{\mathsf{var}} + 1 = |\rho|_{\mathsf{var}}$. For the *moreover* part, let $l := (x, C\langle\lambda x.D_n\rangle, L_n \cdot L)$ and consider a logged position $l' \neq l$ in s. By *i.h.* $\mathsf{depth}(l') \leq \mathsf{depth}(s') < \mathsf{depth}(s)$. For l, instead, by definition of $\mathsf{depth}(\cdot)$ we have $\mathsf{depth}(l) = \mathsf{depth}(s)$.
- Transitions $\to_{\bullet 3}$, $\to_{\bullet 4}$, and $\to_{\mathsf{jmp}}$: the result holds by *i.h.*, since the direction has not changed and neither has the depth of the tape. For the *moreover* part, the every logged position of s is in s', and so it follows by the *i.h.*
- Transition $\to_{\mathsf{arg}}$: the result follows by *i.h.*, since the depth of the tape of s is the same of the depth of the log of s'.

  $$s = (u, \underline{C\langle\langle\cdot\rangle t\rangle}, L, l \cdot T) \to_{\mathsf{arg}} (\underline{t}, C\langle u\langle\cdot\rangle\rangle, l \cdot L, T) = s'$$

  For the *moreover* part, the every logged position of s is in s', and so it follows by the *i.h.*

□

Remember that $\to_{\uparrow} := \to_{\bullet 3, \bullet 4, \mathsf{arg}, \mathsf{jmp}}$. We also set $\to_{\downarrow} := \to_{\bullet 1, \bullet 2, \mathsf{var}}$.

Third, we bound ↑ phases. The number of jumps in a single phase $s \to^*_{\uparrow} s'$ of ↑ transitions is bound by $\mathsf{depth}(s)$, and pairing it up with Lemma 7.5.1 we obtain a bound on the phase. By the depth invariant the bound can be given relatively to $|\rho|_{\mathsf{var}}$, and a standard argument extends the bound to all ↑ phases in a run, adding a quadratic dependency. Let $|\rho|_{\uparrow}$ be the number of $\to_{\uparrow}$ transitions in ρ.

Lemma 7.5.3 (Bound on ↑ phases)**.**

1. One $\uparrow$ phase*: if* $s = (t, \underline{C}, L, T)$ *is a reachable state and* $\rho : s \to^*_{\uparrow} s'$ *then* $|\rho| \leq \mathsf{depth}(s) \cdot |C\langle t\rangle|$.
2. All $\uparrow$ phases*: if* $\rho : s_t \to^*_{\lambda\mathrm{JAM}} s$ *then* $|\rho|_{\uparrow} \leq |\rho|^2_{\mathsf{var}} \cdot |t|$.

Proof.

1. We can split ρ in many subruns $\rho_1 \dots \rho_n$ consisting only of $\to_{\bullet 3, \bullet 4}$ sequences and interleaved by $\to_{\mathsf{jmp}}$ transitions, *i.e.* $\rho = \rho_1 \cdot \to_{\mathsf{jmp}} \rho_2 \cdot \to_{\mathsf{jmp}} \cdots \rho_n$. By Lemma 7.5.1, each ρ_i is such that $|\rho_i| \leq |C\langle t\rangle|$. Moreover, note that the log is untouched by ρ_i and that the number of $\to_{\mathsf{jmp}}$ transitions is bound by the depth of the first logged position in L, itself bound by $\mathsf{depth}(s)$ by Lemma 7.5.2. Then $|\rho| \leq \mathsf{depth}(s) \cdot |C\langle t\rangle|$.
2. The run ρ has shape $\rho_1^{\downarrow}\rho_1^{\uparrow}\rho_2^{\downarrow}\rho_2^{\uparrow} \dots \rho_n^{\downarrow}\rho_n^{\uparrow}$ where $\rho_i^{\downarrow}$ is made out of $\to_{\downarrow}$ transitions and $\rho_i^{\uparrow}$ is made out of $\to_{\uparrow}$ transitions. By the previous point, we have $|\rho_i^{\uparrow}| \leq \mathsf{depth}(s_i^{\uparrow}) \cdot |t|$ where $s_i^{\uparrow}$ is the source state of $\rho_i^{\uparrow}$. By Lemma 7.5.2, $\mathsf{depth}(s_i^{\uparrow}) = \sum_{j=1}^{i} |\rho_j^{\downarrow}|_{\mathsf{var}}$. Now, $|\rho|_{\uparrow} = \sum_{i=1}^{n} |\rho_i^{\uparrow}| \leq \sum_{i=1}^{n} \mathsf{depth}(s_i^{\uparrow}) \cdot |t| = |t| \cdot \sum_{i=1}^{n} \sum_{j=1}^{i} |\rho_j^{\downarrow}|_{\mathsf{var}} \leq |t| \cdot \sum_{i=1}^{n} |\rho|_{\mathsf{var}} \leq |t| \cdot |\rho|^2_{\mathsf{var}}$.

□

The Complexity of the λJAM. From the complexity of the KAM, the fact that the λJAM and the KAM do exactly the same number of $\to_{\mathsf{var}}$ transitions, and that the number of $\uparrow$ transition of the λJAM are bound by $|\rho|^2_{\mathsf{var}} \cdot |t|$, we easily obtain the following results.

Theorem 7.5.4 (λJAM complexity)**.** *Let t be a closed term such that $t \to^n_{wh} u$, u be $\to_{wh}$ normal, and ρ_J and ρ_K be the complete λJAM and KAM runs from t. Then:*

1. *The λJAM is quadratically slower than the KAM: $|\rho_K| \leq |\rho_J| = \mathcal{O}(|\rho_K|^2 \cdot |t|)$.*
2. *The λJAM is (slowly) reasonable: $|\rho_J| = \mathcal{O}(n^4 \cdot |t|)$, and the cost of implementing ρ_J on a RAM is also $\mathcal{O}(n^4 \cdot |t|)$.*

Proof.

1. By Theorem 7.4.6.1, $|\rho_J| = |\rho_K| + |\rho_J|_{\uparrow}$ and $|\rho_K|_{\mathsf{var}} = |\rho_J|_{\mathsf{var}}$. By Lemma 7.5.3.2, $|\rho_J|_{\uparrow} = |\rho_J|^2_{\mathsf{var}} \cdot |t| = |\rho_k|^2_{\mathsf{var}} \cdot |t| \leq^* |\rho_k|^2 \cdot |t|$, from which the statement follows.
2. The previous point gives $|\rho_J| = \mathcal{O}(|\rho_K|^2 \cdot |t|)$ where ρ_K is the corresponding run on the KAM. As recalled in Sect. 7.5, $|\rho_K| = \mathcal{O}(n^2)$, from which we obtain $|\rho_J| = \mathcal{O}(n^4 \cdot |t|)$.

   To obtain the cost of implementing on a RAM, we need to consider the cost of implementing single transitions. They all have constant cost but for $\to_{\mathsf{var}}$ that costs $|t|$. Now note that in the length bound $|\rho_J| = \mathcal{O}(n^4 \cdot |t|)$ the component $|t|$ comes from the $\uparrow$ transitions, not $\to_{\mathsf{var}}$, so that the cost on RAM is not $\mathcal{O}(n^4 \cdot |t|^2)$ but simply $\mathcal{O}(n^4 \cdot |t|)$.

□

7.6 The Pointer Abstract Machine, Revisited

The Pointer Abstract Machine (PAM), due to Danos et al. (1996), gives an operational account of the interaction process at work in Hyland and Ong (2000) game semantics. The machine is always described rather informally via a pseudo-code algorithm. Here we define it according to our syntactic style, calling it λPAM, and provide its first formal and manageable presentation as an actual abstract machine.

Our result concerning the λPAM is that it is strongly bisimilar to the λJAM. Roughly, the two are the same machine, with exactly the same time behavior, they just use different data structures. In particular, the PAM can be seen as a way of implementing the JAM through pointers, *i.e.* sharing, which becomes explicit. The equivalence is mentioned by Danos and Regnier (1999), but it is not proved.

POSITIONS	$p ::= (t, C)$	TAPES	$T ::= \epsilon \mid \bullet \cdot T \mid p \cdot T$
HISTORIES	$H ::= \epsilon \mid (p,i) \cdot H$	STATES	$s ::= (t, C, H, i, T, d)$ $\quad i \in \mathbb{N}$

Sub-term	Context	Hist.	Index	Tape		Sub-term	Context	Hist.	Index	Tape
$\underline{tu}$	C	H	i	T	$\to_{\bullet 1}$	$\underline{t}$	$C\langle\langle\cdot\rangle u\rangle$	H	i	$\bullet \cdot T$
$\underline{\lambda x.t}$	C	H	i	$\bullet \cdot T$	$\to_{\bullet 2}$	$\underline{t}$	$C\langle\lambda x.\langle\cdot\rangle\rangle$	H	i	T
$\underline{x}$	$C\langle\lambda x.D_n\rangle$	H	i	T	$\to_{\mathsf{var}}$	$\lambda x.D_n\langle x\rangle$	$\underline{C}$	H	$\phi_H^n(i)$	$(x, C\langle\lambda x.D_n\rangle) \cdot T$
t	$\underline{C\langle\langle\cdot\rangle u\rangle}$	H	i	$\bullet \cdot T$	$\to_{\bullet 3}$	tu	$\underline{C}$	H	i	T
t	$\underline{C\langle\lambda x.\langle\cdot\rangle\rangle}$	H	i	T	$\to_{\bullet 4}$	$\lambda x.t$	$\underline{C}$	H	i	$\bullet \cdot T$
t	$\underline{C\langle\langle\cdot\rangle u\rangle}$	H	i	$p \cdot T$	$\to_{\mathsf{arg}}$	$\underline{u}$	$C\langle t\langle\cdot\rangle\rangle$	$(p,i) \cdot H$	$\lvert H\rvert + 1$	T
t	$\underline{C\langle u\langle\cdot\rangle\rangle}$	H	i	T	$\to_{\mathsf{jmp}}$	x_i^H	$\underline{D_i^H}$	H	$i-1$	T

FIGURE 7.3: Data structures and transitions of the λ Pointer Abstract Machine (λPAM).

Fragmented *vs* **Monolitic Run Traces.** Both machines jump and need to store information about the run, to jump to the right place. They differ on how they represent this information. The λJAM uses logged positions, that is, positions coming with the information to be restored after the jump. The approach can be seen as *fragmented*, as the trace of the run is distributed among all the logged positions in the state. The λPAM adopts a *monolitic* approach, storing all the information in a unique *history* H, a new data structure encoding the whole run in a minimalistic and sophisticated way. Roughly, the history H saves all the variable positions p for which an argument as been found, each one with a pointer (under the form of an index i) to a previous variable position p' in H. The index i intuitively realizes a mechanism to retrieve the log associated to p by the λJAM. We first define the machine and then explain the relationship between the two approaches.

Data Structures. All the data structures of the PAM are defined in Fig. 7.3. Positions are no longer logged, and noted with p, p', etc. An *index* i is simply a natural number. *Indexed positions* are pairs (p,i). A history H is a sequence of indexed variable positions (accumulated from right to left). The idea is that indices are pointers to entries in the history, that is, if the i-th entry of H is (p,j) then j points to a previous entry in H, that is, $j < i$. The tape of the λPAM is a stack containing variable positions and occurrences of $\bullet$.

Transitions and Look-Up. Initial states have the form $s_t := (\underline{t}, \langle\cdot\rangle, \epsilon, 0, \epsilon)$, the transitions of the λPAM are in Fig. 7.3, they are labeled exactly as in the λJAM, and their union is noted $\to_{\lambda\text{PAM}}$. Transitions $\to_{\mathsf{var}}$ and $\to_{\mathsf{jmp}}$ need to retrieve information from the history H, for which there are some dedicated notations. We use i_k^H, x_k^H, D_k^H to denote, respectively, the index, variable, and context of the k-th indexed position in H.

Transition $\to_{\mathsf{var}}$ moreover looks up into H in an unusual way. The idea is that it accesses H n times to retrieve an index. The first time it retrieves the indexed position (p_1, j_1) of index i, to then retrieve the position (p_2, j_2) of index j_1, and so on, until it retrieves j_n and makes it the new state index. This is formalized using the look-up function $\phi_H : \mathbb{N} \to \mathbb{N}$ defined as $\phi_H(k) := i_k^H$, and whose powers ϕ_H^n are defined as $\phi_H^n(k) := \phi_H(\phi_H^{(n-1)}(k))$, where $\phi_H^0(k) := k$. Note that implementing $\to_{\mathsf{var}}$ on RAM then costs n, that is bound by the size $\lvert t\rvert$ of the initial term, exactly as for the λJAM, while all other transitions have constant cost.

An Example. As for the other machines we have considered in this thesis, we give the execution trace of the λPAM on the term $(\lambda y.\lambda x.xy)\mathsf{II}$. The reader can grasp some intuition considering that the PAM is strongly bisimilar to the λJAM. In particular, the λPAM considers explicit pointers. Indeed, as we have already pointed out, λJAM logs are not actually copied in the λJAM $\to_{\mathsf{var}}$ transition: what is duplicated is just a pointer to them. The λPAM handles this mechanism directly

in its definition, and can thus be considered as a low-level implementation of the λJAM. In the following we will explain this in more detail.

	Sub-term	Context	Hist.	Index	Tape	Dir
	$\underline{(\lambda y.\lambda x.xy)\mathsf{II}}$	$\langle\cdot\rangle$	ϵ	0	ϵ	$\downarrow$
$\to_{\bullet 1}$	$\underline{(\lambda y.\lambda x.xy)\mathsf{I}}$	$\langle\cdot\rangle\mathsf{I}$	ϵ	0	$\bullet$	$\downarrow$
$\to_{\bullet 1}$	$\underline{\lambda y.\lambda x.xy}$	$\langle\cdot\rangle\mathsf{II}$	ϵ	0	$\bullet\cdot\bullet$	$\downarrow$
$\to_{\bullet 2}$	$\underline{\lambda x.xy}$	$(\lambda y.\langle\cdot\rangle)\mathsf{II}$	ϵ	0	$\bullet$	$\downarrow$
$\to_{\bullet 2}$	$\underline{xy}$	$(\lambda y.\lambda x.\langle\cdot\rangle)\mathsf{II}$	ϵ	0	ϵ	$\downarrow$
$\to_{\bullet 1}$	$\underline{x}$	$(\lambda y.\lambda x.\langle\cdot\rangle y)\mathsf{II}$	ϵ	0	$\bullet$	$\downarrow$

After having looked for the head variable through the spine of the term, the λPAM, now in $\uparrow$ mode, queries the argument of x, namely $\lambda z.z$, that then explores. The argument of its head variable z is y, that has to be found via *backtracking* or *jumping*. We set $p_x := (x, (\lambda y.\lambda x.\langle\cdot\rangle y)\mathsf{I}(\lambda z.z))$.

	Sub-term	Context	Hist.	Index	Tape	Dir
	$\underline{x}$	$(\lambda y.\lambda x.\langle\cdot\rangle y)\mathsf{I}(\lambda z.z)$	ϵ	0	$\bullet$	$\downarrow$
$\to_{\mathsf{var}}$	$\lambda x.xy$	$\underline{(\lambda y.\langle\cdot\rangle)\mathsf{I}(\lambda z.z)}$	ϵ	0	$p_x\cdot\bullet$	$\uparrow$
$\to_{\bullet 4}$	$\lambda y.\lambda x.xy$	$\underline{\langle\cdot\rangle\mathsf{I}(\lambda z.z)}$	ϵ	0	$\bullet\cdot p_x\cdot\bullet$	$\uparrow$
$\to_{\bullet 3}$	$(\lambda y.\lambda x.xy)\mathsf{I}$	$\underline{\langle\cdot\rangle(\lambda z.z)}$	ϵ	0	$p_x\cdot\bullet$	$\uparrow$
$\to_{\mathsf{arg}}$	$\underline{(\lambda z.z)}$	$(\lambda y.\lambda x.xy)\mathsf{I}\langle\cdot\rangle$	$(p_x,0)$	1	$\bullet$	$\downarrow$
$\to_{\bullet 2}$	$\underline{z}$	$(\lambda y.\lambda x.xy)\mathsf{I}(\lambda z.\langle\cdot\rangle)$	$(p_x,0)$	1	ϵ	$\downarrow$
$\to_{\mathsf{var}}$	$\lambda z.z$	$\underline{(\lambda y.\lambda x.xy)\mathsf{I}\langle\cdot\rangle}$	$(p_x,0)$	1	$(z,(\lambda y.\lambda x.xy)\mathsf{I}(\lambda z.\langle\cdot\rangle))$	$\uparrow$

The jump is simulated by the λPAM retrieving the position saved in the history at the current index, and then updating the index accordingly, *i.e.* diminishing it by one. Intuitively, this corresponds to the 'unpacking' made by the λJAM in the $\to_{\mathsf{jmp}}$ transition. We set $p_z := (z, (\lambda y.\lambda x.xy)\mathsf{I}(\lambda z.\langle\cdot\rangle))$ and $p_y = (y, (\lambda y.\lambda x.x\langle\cdot\rangle)\mathsf{II})$.

	Sub-term	Context	Hist.	Index	Tape	Dir
	$\lambda z.z$	$\underline{(\lambda y.\lambda x.xy)\mathsf{I}\langle\cdot\rangle}$	$(p_x,0)$	1	p_z	$\uparrow$
$\to_{\mathsf{jmp}}$	x	$\underline{(\lambda y.\lambda x.\langle\cdot\rangle y)\mathsf{I}(\lambda z.z)}$	$(p_x,0)$	0	p_z	$\uparrow$
$\to_{\mathsf{arg}}$	$\underline{y}$	$(\lambda y.\lambda x.x\langle\cdot\rangle)\mathsf{II}$	$(p_z,0)\cdot(p_x,0)$	2	ϵ	$\downarrow$
$\to_{\mathsf{var}}$	$\lambda y.\lambda x.xy$	$\underline{\langle\cdot\rangle\mathsf{II}}$	$(p_y,0)\cdot(p_z,0)\cdot(p_x,0)$	0	p_y	$\uparrow$
$\to_{\mathsf{arg}}$	$\underline{\mathsf{I}}$	$(\lambda y.\lambda x.xy)\langle\cdot\rangle\mathsf{I}$	$(p_y,0)\cdot(p_z,0)\cdot(p_x,0)$	3	ϵ	$\downarrow$

Final States and Invariants. Final states of the λPAM have, as expected, shape $(\underline{\lambda x.t}, C, H, i, \epsilon, \downarrow)$. This follows from the fact that the machine is never stuck on $\to_{\mathsf{var}}$ steps because $\phi_H^n(i)$ is undefined. Note indeed a subtle point: $\phi_H(0)$ is undefined, so, potentially, $\phi_H^n(i)$ may be undefined. We then need an invariant ensuring that—in the source state of $\to_{\mathsf{var}}$— $\phi_H^n(i)$ is always defined. The next statement collects also other minor invariants of the λPAM.

We say that H *has depth* (at least) $n \in \mathbb{N}$ at i if $n = 0$ or if $n > 0$ and $\phi_H^m(i) > 0$ for every $m < n$.

Lemma 7.6.1 (λPAM invariants). *Let $s = (t, C_n, H, i, T, d)$ be a reachable PAM state. Then:*

1. Depth*: H has depth n at i. Moreover, if $((u, D_m), j)$ is the k-th indexed position of H, with $k > 0$, then H has depth m at $k - 1$.*
2. Tape, index, and direction*: if $d = \downarrow$, then $i = |H|$ and T does not contain any logged position, otherwise if $d = \uparrow$ then T contains exactly one position.*

Proof. By induction on the length of the run reaching s, together with an immediate inspection of the transitions using the *i.h.*. □

History, Indices, and Logs. Let's now explain the relationship between the λJAM and the λPAM. The history H essentially stores the sequence of $\rightarrow_{\mathsf{var}}$ queries, consisting of the position of a variable needing an argument, that the λPAM has completed, that is, for which it has found the argument. The key point is that it stores them with an index i. Indices are a low-level mechanism to retrieve logs, that are crumbled and shuffled all over H. Let us explain how a log $(p_1, L_1)\cdot \ldots \cdot (p_n, L_n)$ of a reachable λJAM state is represented by the index i_1 and the history H of the corresponding λPAM state. There are two ideas:

- *The sequence of positions*: p_1 is in the i_1-th entry (p_1, i_2) of H, p_2 is in the i_2-th entry (p_2, i_3), and so on.
- *The log of each position*: the log L_1 of p_1 is represented in H (recursively following the same principle) starting from index $i_1 - 1$, the log L_2 starting from index $i_2 - 1$, and so on.

The Bisimulation. The given explanation underlies the following definition of relations $\simeq_T$,$\simeq_{LH}$ and $\simeq$ between data structures and states of the λJAM and the λPAM, that induce a strong bisimulation. The intended meaning of the relation $L \simeq_{LH} (H, i)$ is that the log L is represented in the history H starting from index i.

Definition 7.6.2. *The relations $\simeq_T, \simeq_{LH}$ and $\simeq$ are defined as follows.*

$$\textsc{Tapes}\quad \frac{}{\epsilon \simeq_T \epsilon} \qquad \frac{T_J \simeq_T T_P}{\bullet\cdot T_J \simeq_T \bullet\cdot T_P} \qquad \frac{T_J \simeq_T T_P}{(x, C, L)\cdot T_J \simeq_T (x, C)\cdot T_P}$$

$$\textsc{Log-His.}\quad \frac{}{\epsilon \simeq_{LH} (H, 0)} \qquad \frac{(x, C) = (x_i^H, D_i^H) \qquad L \simeq_{LH} (H, \phi_H(i)) \qquad L' \simeq_{LH} (H, i-1)}{(x, C, L')\cdot L \simeq_{LH} (H, i)}$$

$$\textsc{States}\quad \frac{T_J \simeq_T T_P \qquad L \simeq_{LH} (H, i)}{(t, C, L, T_J, d) \simeq (t, C, H, i, T_P, d)}$$

Note that in the second rule for $\simeq_{LH}$ the index i is ≥ 1, and that $\simeq$ contains all pairs of initial states. Note also that the (logged) positions case of $\simeq_T$ (rightmost rule for $\simeq_T$) the log L has no matching construct on the λPAM side. This is why the next theorem is stated together with an invariant (the *moreover* part), allowing to retrieve that log from the history. We need a preliminary lemma.

Lemma 7.6.3 (Logs and histories)**.** *Let $L \simeq_{LH} (H, i)$.*

1. Log splitting*: if $L = L_n\cdot L'$ then $L' \simeq_{LH} (H, \phi_H^n(i))$.*
2. History extension*: if (p, j) be an indexed position then $L \simeq_{LH} ((p, j)\cdot H, i)$.*

Proof.

1. By induction on n:
   - *Base case*: if $n = 0$, then it is trivially satisfied since $L_n = \epsilon$ and $\phi_H^n(i) = i$, so that $L' = L_n\cdot L' \simeq_{LH} (H, i) = (H, \phi_H^n(i))$, as required.
   - *Inductive case*: if $n > 0$ first of all note that $\phi_H^m(i)$ is defined for all $m \leq n$ by the depth invariant (Lemma 7.6.1). Then, $L = L_{n-1}\cdot e\cdot L'$ and by *i.h.* $e\cdot L' \simeq_{LH} (H, \phi_H^{n-1}(i))$. By definition of $\simeq_{LH}$ this is possible only if $L' \simeq_{LH} (H, \phi_H(\phi_H^{n-1}(i)))$, *i.e.* $L' \simeq_{LH} (H, \phi_H^n(i))$.
2. By induction on L. Cases:
   - *Empty, i.e.* $L = \epsilon$. We have that the hypothesis is $\epsilon \simeq_{LH} (H, 0)$, because it is the only derivable relation for empty logs. Then $\epsilon \simeq_{LH} ((p, j)\cdot H, 0)$.
   - *Non-empty, i.e.* $L = (x, C, L'')\cdot L'$. By hypothesis, $l\cdot L' \simeq_{LH} (H, i)$, which implies that
     (a) $(x, C) = (x_i^H, D_i^H)$,

(b) $L' \simeq_{LH} (H, \phi_H(i))$, and
(c) $L'' \simeq_{LH} (H, i-1)$.
By *i.h.*, we obtain $L' \simeq_{LH} ((p,j)\cdot H, \phi_H(i))$ and $L'' \simeq_{LH} ((p,j)\cdot H, i-1)$, and clearly $(x, C) = (x_i^{(p,j)\cdot H}, D_i^{(p,j)\cdot H})$. Then $L = (x, C, L'')\cdot L' \simeq_{LH} ((p,j)\cdot H, i)$.

□

Then, we are able to prove that $\simeq$ is a strong bisimulation.

Theorem 7.6.4 ($\simeq$ is a strong bisimulation)**.**

1. *for every run* $\rho_J : s_t^{\lambda JAM} \to^*_{\lambda\text{JAM}} s_J$ *there exists a run* $\rho_P : s_t^{\lambda PAM} \to^*_{\lambda\text{PAM}} s_P$ *such that* $s_J \simeq s_P$ *and* $|\rho_J| = |\rho_P|$ *and performing exactly the same transitions;*
2. *for every run* $\rho_P : s_t^{\lambda PAM} \to^*_{\lambda\text{PAM}} s_P$ *there exists a run* $\rho_J : s_t^{\lambda JAM} \to^*_{\lambda\text{JAM}} s_J$ *such that* $s_J \simeq s_P$ *and* $|\rho_J| = |\rho_P|$ *and performing exactly the same transitions.*

Moreover, if $s_J = (t, \underline{C}, L, T_J, \uparrow) \simeq (t, \underline{C}, H, i, T_P, \uparrow) = s_P$ *and* (x, D, L') *is the unique logged position in* T_J *then* $L' \simeq_{LH} (H, |H|)$.

Proof. We prove the first point, the second point is symmetrical (simply replacing the use of Lemma 7.6.1—in the case of transition $\to_{\mathsf{var}}$ below—with Lemma 7.1.1). By induction on $|\rho_J|$. If ρ_J is empty then simply take ρ_P as the empty run. Otherwise, by *i.h.* there exists a λPAM run $\rho_P : s_t^{\lambda\text{PAM}} \to^*_{\lambda\text{PAM}} s_P$ such that $s_J \simeq s_P$ and $|\rho_J| = |\rho_P|$. Note that $s_J \simeq s_P$ implies $s_J = (t, E, L, T_J, d)$ and $s_P = (t, E, T_P, H, i, d)$ with $T_J \simeq_T T_P$ and $L \simeq_{LH} (H, i)$.

Let's consider the possible extensions of ρ_J with a further transition. Cases:

- Transitions $\to_{\bullet 1}, \to_{\bullet 2}, \to_{\bullet 3}, \to_{\bullet 4}$: we show one such case, the other are analogous.

$$\begin{array}{ccc} (\underline{ur}, E, L, T_J) & \to_{\bullet 1} & (\underline{u}, E\langle\langle\cdot\rangle r\rangle, L, \bullet\cdot T_J) \\ \simeq & & \simeq \\ (\underline{ur}, E, H, i, T_P) & \to_{\bullet 1} & (\underline{u}, E\langle\langle\cdot\rangle r\rangle, H, i, \bullet\cdot T_P) \end{array}$$

  For $\to_{\bullet 3}$ and $\to_{\bullet 4}$ the *moreover* part follows from the *i.h.*

- Transition $\to_{\mathsf{var}}$. We are in the following situation:

$$s_J = (\underline{x}, C\langle\lambda x.D_n\rangle, L_n\cdot L', T_J) \simeq (\underline{x}, C\langle\lambda x.D_n\rangle, H, i, T_P) = s_P$$

  with $T_J \simeq_T T_P$ and $L = L_n\cdot L' \simeq_{LH} (H, i)$. The λPAM can do a $\to_{\mathsf{var}}$ transition (guaranteed by the depth invariant of Lemma 7.6.1), but we have to verify that the two target states are still $\simeq$-related. By Lemma 7.6.3.1, we have $L' \simeq_{LH} (H, \phi_H^n(i))$. Then:

$$\begin{array}{ccc} (\underline{x}, C\langle\lambda x.D_n\rangle, L_n\cdot L', T_J) & \to_{\mathsf{var}} & (\lambda x.D_n\langle x\rangle, \underline{C}, L', (x, C\langle\lambda x.D_n\rangle, L_n\cdot L')\cdot T_J) \\ \simeq & & \simeq \\ (\underline{x}, C\langle\lambda x.D_n\rangle, H, i, T_P) & \to_{\mathsf{var}} & (\lambda x.D_n\langle x\rangle, \underline{C}, H, \phi_H^n(i), (x, C\langle\lambda x.D_n\rangle)\cdot T_P) \end{array}$$

  Now, the *moreover* part. We have to prove that $L_n\cdot L' \simeq_{LH} (H, |H|)$. By hypothesis $L_n\cdot L' \simeq_{LH} (H, i)$ and by Lemma 7.6.1, we have $i = |H|$.

- Transition $\to_{\mathsf{arg}}$. We are in the following situation:

$$s_J = (t, \underline{C\langle\langle\cdot\rangle u\rangle}, L, l\cdot T_J') \simeq (t, \underline{C\langle\langle\cdot\rangle u\rangle}, H, i, p\cdot T_P') = s_P$$

  with $l\cdot T_J' \simeq_T p\cdot T_P'$ and $L \simeq_{LH} (H, i)$. The λPAM can do a $\to_{\mathsf{arg}}$ transition, but we have to verify that the two target states are still $\simeq$-related. Namely, we have to show that $l\cdot L \simeq_{LH} ((p,i)\cdot H, |H|+1)$. Let us set $H' := (p,i)\cdot H$ and $j := |H'| = |H| + 1$. We check that all three hypothesis of the rule defining $\simeq_{LH}$ hold:

1. Since $L \simeq_{LH} (H, i)$, Lemma 7.6.3.2 gives $L \simeq_{LH} ((p, i) \cdot H, i)$. Note that $\phi_{H'}(j) = i$, that is, $L \simeq_{LH} (H', \phi_{H'}(j))$.
2. Since $l \cdot T'_J \simeq_T p \cdot T'_P$, if $l = (x, D, L')$, then $p = (x, D)$ and thus $(x, D) = (x_j^{H'}, D_j^{H'}) = (x, D)$.
3. By *i.h.*, the logged position $l = (x, D, L')$ on the λJAM tape verifies $L' \simeq_{LH} (H, |H|)$. By Lemma 7.6.3.2, $L' \simeq_{LH} ((p, i) \cdot H, |H|)$, that is, $L' \simeq_{LH} (H', j - 1)$.

Then the two target states match:

$$\begin{array}{ccc} (t, \underline{C\langle\langle\cdot\rangle u\rangle}, L, l \cdot T'_J) & \to_{\mathsf{arg}} & (\underline{u}, C\langle t\langle\cdot\rangle\rangle, l \cdot L, T'_J) \\ \simeq & & \simeq \\ (t, \underline{C\langle\langle\cdot\rangle u\rangle}, H, i, p \cdot T'_P) & \to_{\mathsf{arg}} & (\underline{u}, C\langle t\langle\cdot\rangle\rangle, H', j, T'_P) \end{array}$$

- Transition $\to_{\mathsf{jmp}}$. We are in the following situation:

$$s_J = (t, \underline{C\langle u\langle\cdot\rangle\rangle}, (x, D, L') \cdot L', T_J) \simeq (t, \underline{C\langle u\langle\cdot\rangle\rangle}, H, i, T_P) = s_P$$

with $T_J \simeq_T T_P$ and $(x, D, L') \cdot L' \simeq_{LH} (H, i)$. The λPAM can do a $\to_{\mathsf{jmp}}$ transition, but we have to verify that the two target states are still $\simeq$-related. Note that, since $(x, D, L') \cdot L' \simeq_{LH} (H, i)$, we have $(x, D) = (x_i^H, D_i^H)$ and $L' \simeq_{LH} (H, i - 1)$. Therefore:

$$\begin{array}{ccc} (t, \underline{C\langle u\langle\cdot\rangle\rangle}, (x, D, L') \cdot L', T_J) & \to_{\mathsf{jmp}} & (x, \underline{D}, L', T_J) \\ \simeq & & \simeq \\ (t, \underline{C\langle u\langle\cdot\rangle\rangle}, H, i, T_P) & \to_{\mathsf{jmp}} & (x_i^H, \underline{D_i^H}, H, i - 1, T_P) \end{array}$$

The *moreover* part follows from the *i.h.*

□

Strong bisimulations trivially preserve termination and the length of runs.

Corollary 7.6.5 (Termination and λPAM implementation)**.** *λJAMt terminates if and only if λPAMt terminates, and the two runs use exactly the same transitions. Therefore, the λPAM implements Closed CbN.*

As a concluding remark, looking at how the λPAM is defined, it is obvious that it *cannot* be space reasonable. Indeed, its space consumption, in particular the history, is inflationary and dependent (linear) on the number of $\to_{\mathsf{var}}$ transitions, which is a reasonable *time* and not *space* measure. Since the λPAM is a low-level implementation of the λJAM, the same considerations hold also for the λJAM.

Chapter 8

The Space KAM is Reasonable

Bounding the amount of resources needed by algorithms or programs is a fundamental problem in computer science. Here we are concerned with sub-linear space. In many applications, say, stream processing or web crawling, *linear* bounds on computing space are not satisfactory, given the enormous amount of data processed. Theoretically, complexity classes such as L (logarithmic space), although apparently small, are already very interesting for complexity theory, and not even known to be distinct from P.

Dealing with sub-linear space bounds in the λ-calculus, or in functional programming languages, has always been considered a challenge. A first reason is that the natural notions of time and space in the λ-calculus have some puzzling properties, as we shall see. But sub-linear space is special, in that, since the λ-calculus does not distinguish between programs and data, there is also no distinction between input space and work space, and thus no natural notion of sub-linear space.

The literature about the λ-calculus does offer results about space complexity, but they are all *partial*, as they either concern variants of the λ-calculus (Dal Lago and Schöpp (Dal Lago and Schöpp, 2016; Schöpp, 2007), Mazza (2015) and Ghica (2007)), or they are not valid when the bounds in spaces are sub-linear (Forster et al. (2020)).

The main contribution of this chapter is the first fully-fledged space reasonability result for the pure, untyped λ-calculus. Precisely, we represent the *input space* as λ-terms, and the *work space* as the space used by a variant of the well-known Krivine's abstract machine (KAM). Two important aspects of our *Space KAM* are *eager garbage collection* and the fact that, for the first time, we distinguish between two forms of sharing usually considered as one: the *sharing of sub-terms* provided by environments, and the *sharing of environments* themselves. The Space KAM adopts the former but forbids the latter, which is essential to prove that its space cost model is reasonable. Designing the Space KAM, however, is only half of the story. The other half is the refinement of the encoding of Turing machines into the λ-calculus: the reference one by Dal Lago and Accattoli (2017) uses a linear amount of space to simulate the TM tapes, thus forbidding to preserve logarithmic space. Let us detail all this.

Reasonable Cost Models. We recall that according to the seminal work by Slot and Emde Boas (1988), the adequacy of space and time cost models is judged in relationship to whether they reflect the corresponding cost models of Turing machines (shortened to TM), the computational theory from which computational complexity stems. Namely, a cost model for a computational theory T is reasonable if there are mutual simulations of T and TMs (or another reasonable theory) working within:

- *for time*, a polynomial overhead;
- *for space*, a linear overhead.

In many cases the two bounds hold simultaneously for the same simulation, but this is not a strict requirement. The aim is to ensure that the basic hierarchy of complexity classes

$$\mathsf{L} \subseteq \mathsf{P} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}$$

can be equivalently defined on any reasonable theory, that is, that such classes are *robust*, or theory-independent. Note a slight asymmetry: while for time the complexity of the required overhead

(polynomial) coincides with the smallest robust time class (P), for space the smallest robust class is logarithmic (L) and not linear space.

Locked Time and Space. On TMs, space cannot be greater than time, because using space requires time—we shall then say that space and time are *locked*. If both the time and space cost models of a computational theory are reasonable, are they also necessarily locked? This seems natural, but it is not what happens in the λ-calculus, at least with respect to its natural cost models.

Natural Cost Models for the λ-Calculus. The natural time cost model is the number of β-steps (according to some fixed evaluation strategy), its only notion of computational step. The natural space cost model is the maximum size of λ-terms during reduction. The puzzling point is that, in the λ-calculus, natural space can be exponential in natural time (independently of the strategy), a degeneracy known as *size explosion*—we shall say that time and space are *explosive*.

Is the λ-calculus reasonable? This was unclear for a while, because of the intuition that reasonable cost models have to be locked. Note that there is, in principle, an alternative and non-explosive approach to time in the λ-calculus: the time to write down the whole evaluation rather than only the number of evaluation steps. Such *low-level* time is reasonable, and locked with natural space, but implies that the various β-steps in an evaluation sequence are given different costs, which is theoretically not ideal, and also distant from the practice of programming, that rather adopts and confirms the natural (and uniform) cost model (corresponding to the number of function calls).

In the study of natural time, what is delicate is the simulation of the λ-calculus into a reasonable theory, which typically is the one of RAMs rather than TMs. The difficulty stems from the explosiveness of natural time, and requires a slight paradigm shift. To circumvent the exponential explosion in space, λ-terms are usually evaluated *up to sharing*, that is, in abstract machines with sharing that compute shared representations of the results. These representations can be exponentially smaller than the results themselves: explosiveness is then encapsulated in the sharing unfolding process. The number of β steps (according to various evaluation strategies) then turns out to be a reasonable time cost model (up to sharing), despite explosiveness.

Indeed, we have already seen that it is possible to realize mechanisms to evaluate λ-terms in reasonable time, *e.g.* the Krivine Abstract Machine. The number of transitions it takes to evaluate a λ-term *is* a reasonable cost model *and* moreover is compatible with the *natural* time cost model for the λ-calculus.

What about space? When trying to prove that a space cost model for the λ-calculus is reasonable, the difficult direction is the simulation of TMs in the λ-calculus. TMs are space-minimalist, as their only data structure, the tape, juxtaposes cells rather then linking them—we shall see that this is one of the key points. Motivated by time-efficiency, all abstract machines for the λ-calculus rely instead on linked data structures, and the linking pointers add a logarithmic factor to the overhead for the simulation of TMs that is space unreasonable. Therefore, reasonable space requires to evaluate without using linked data structures when they are not needed, as it is the case for the encoding of TMs. It is a recent observation by Forster et al. (2020) that evaluating without any data structure (via plain rewriting, without sharing) is reasonable for natural space even if unreasonable for natural time (because of size explosion).

Pairing up Natural Time and Natural Space. Forster et al. also show a surprising fact. Given two simulations, one that is reasonable for space but not time, and one that is reasonable for time but not space, there is a smart way of interleaving them as to obtain reasonability for time and space *simultaneously*. Their result therefore shows that, surprisingly, a computational theory can be reasonable for time *and* space, without being locked.

Work vs Natural Space. Now, another puzzling fact is that sub-linear space *cannot* be measured using the natural space cost model, and is then *not* covered by Forster et al.'s result. The reason is that if space is the maximum size of terms in an evaluation sequence, the first of which contains the input, then space simply *cannot be* sub-linear. How could we account for *logarithmic* reasonable space? One needs *log-sensitivity*, that is, a distinction between an immutable *input space*, which

is not counted for space complexity (because otherwise the complexity would be at least linear), and a (smaller) mutable *work space*, that is counted. Moreover, logarithmic space usually requires manipulating *pointers* to the input (which are of logarithmic size) rather than pieces of the input (which can be linear).

Log-sensitivity thus seems to clash with the natural approach based on the rewriting of λ-terms, which does not distinguish between input and work space and that manipulates actual sub-terms rather than pointers. We shall then model evaluation using λ-terms for the input space and the state space of an abstract machine (computing over the input without ever modifying it, and manipulating sub-term pointers) for the work space.

We have seen in the previous chapters that both GoI-based machines such as the λIAM and the λJAM, and environment machines such as the KAM, do not seem the right tools to achieve space reasonability. The situation is actually subtler: we concentrate on how environments are implemented to clarify some points.

Environments. They are data structures used to achieve time reasonability. According to Accattoli and Barras (2017), there are two main styles of environments, local and global. Global environments (as in the Milner Abstract Machine, see Accattoli and Barras (2017)) are space-insensitive because they work over the input space. Local environments (as in the KAM) are log-sensitive. There are two reasons why they are usually space unreasonable. The first one is that garbage collection is not usually accounted for, which leads to ever increasing space usage, while reasonable space should be re-usable. The second and subtler one is the use of pointers for sharing. Local environments use two types of pointers, handling the two forms of sharing: sub-term pointers, which serve to avoid copying sub-terms, and environment pointers, which both realize their linked list structure and share them. Sub-terms pointers, as mentioned above, are a key aspect of logarithmic space computations, and are thus crucial. Environment pointers, which accordingly to Douence and Fradet (2007) are the *essence* of the KAM , are instead what makes environments space unreasonable: they introduce a logarithmic *pointer overhead* that, at best, gives simulations of TMs with a $\mathcal{O}(n \log n)$ overhead in space, instead of the required $\mathcal{O}(n)$ for reasonability. It was then generally concluded that environments cannot provide reasonable space.

Work Space Without Environment Pointers. The literature on abstract machines assumes that pointers are used in implementations without however accounting for them in the underlying specification. Here, we are instead very careful with pointers. We design an abstract machine, the Space KAM, using local environments with sub-term pointers, but—crucially—*without* environment pointers. Similarly to the tapes of TMs, the environments of the Space KAM then are *not* linked structures, but simple *unstructured strings*. Consequently, the unreasonable pointer overhead vanishes.

The moral is that the use of pointers is both essential and dangerous for logarithmic space: sub-term pointers, that is, those to the input, are mandatory for log-sensitivity, while the environment ones—those to the working tape, essentially—are space unreasonable.

Garbage Collection and Unchaining. The Space KAM crucially relies also on two optimizations. One is *eager garbage collection*, to maximize space re-usability. It is implemented in the most naive of ways, because it cannot rely on any pointers or counters, as they would add an unreasonable space overhead. In contrast to common practice, the collection happens *eagerly*, that is, immediately and not when reaching a threshold. The second optimization is *environment unchaining*, a folklore tweak for avoiding space leaks.

Encoding of TMs. Despite the fine tuning of the abstract machine, a reasonable simulation of TMs preserving logarithmic space is not yet obtained, as the encoding of single tape TMs has some inherent limitations with respect to logarithmic space. We then analyze its shortcomings, concerning how tapes are represented and scrolled in the λ-calculus, and modify it accordingly. The main result of this chapter is that the work space of the Space KAM—to be referred to simply as *the work space*—over the new encoding is a reasonable space cost model accommodating sub-linear space. Our new encoding is carefully designed so as to retain the key *indifference property* of

the reference one, that is, the fact that nothing changes if call-by-value rather than call-by-name evaluation is adopted. We then show that our results smoothly transfer to call-by-value evaluation.

Space KAM and Time. We then study the time behavior of the Space KAM. Disabling linked environments implies giving up environment sharing, which—we show with an example—makes the Space KAM unreasonable for natural time. We also prove that the low-level execution time of the Space KAM (which is less interesting than natural time, that is, the number of β steps) is reasonable. The situation is then a familiar one: natural time and work space are explosive, while low-level time and work space are locked. Work space is in this respect a conservative refinement of natural space.

Sub-Term Property. The techniques for reasonable time and reasonable space seem to be at odds, as they make essential but opposite uses of linked data structures. Both techniques, however, crucially rely on the *sub-term property* of abstract machines, that is, the fact that duplicated terms are sub-terms of the initial one. For time, it allows one to bound the cost of duplication, while for space it allows one to see sub-terms as (logarithmic) pointers to the input. The sub-term property seems to be the unavoidable ingredient for reasonability in the λ-calculus.

Related Work. The space inefficiency of environment machines is also observed by Krishnaswami et al. (2012), who propose some techniques to alleviate it in the context of functional-reactive programming and based on linear types. The relevance for space of disabling environment sharing is also stressed by Paraskevopoulou and Appel (2019) in their cost-aware study of closure conversion. A characterization of PSPACEin the λ-calculus is given by Gaboardi et al. (2012), but it relies on alternating time rather than on a notion of space. The already mentioned works by Schöpp (2007) and Dal Lago and Schöpp (2016), and by Mazza (2015) characterize Lin variants of the λ-calculus, while Jones characterizes Lusing a programming language but not based on the λ-calculus Jones, 1999. Blelloch and coauthors study in various papers (Blelloch and Greiner, 1995, 1996; Spoonhower et al., 2010) how to profile (that is, measure) space consumption of functional programs, also done by Sansom and Jones (1995). They are not interested, however, in the reasonability of the cost models, that is, being equivalent to the space of TMs, which is the difficult part of our work. Finally, there is an extensive literature on garbage collection, as witnessed by Jones et al. (2011). We here need a basic eager form, that need not be time efficient, as the Space KAM is time unreasonable anyway.

8.1 Reasonable Preliminaries

In the study of reasonable cost models for the λ-calculus, it is customary to show that the λ-calculus simulates Turing machines reasonably, and conversely that the λ-calculus can be simulated reasonably by random access machines (RAMs and TMs being both reasonable models) *up to sharing*. Since space is more delicate than time, we fix the involved theories and their cost measures carefully.

The Size of λ-Terms. The *(constructor) size* of a λ-term is defined as follows:

$$|x| := 1 \qquad |tu| := |t| + |u| + 1 \qquad |\lambda x.t| := |t| + 1$$

The *code size* $\|t\|$ of a λ-term t is instead bound by $\mathcal{O}(|t| \cdot \log |t|)$. The idea is that, when terms are explicitly represented, variables are some abstract kind of pointer (de Bruijn indices/levels, names, or actual pointers to the syntax tree), of size logarithmic in the number of constructors $|t|$ of t. Moreover, the other constructors are also usually represented using pointers to sub-terms, so that a term with n constructors requires space $\mathcal{O}(n \log n)$ to be represented. For our study, it is important to stress the difference between $|t|$ and $\|t\|$, because given a binary input string i, at first sight its encoding t_i as a λ-term satisfies $|t_i| = \Theta(|i|)$ and $\|t_i\| = \mathcal{O}(|i| \cdot \log |i|)$, and so $\|t_i\|$ has an additional (unreasonable) logarithmic factor. In Sect. 8.3, we shall encode strings in the λ-calculus using the Scott encoding, which has the property that, with respect to some concrete

representations of terms, the variable pointers have constant size, so that $\|t_i\| = |t_i| = \Theta(|i|)$, thus removing the unreasonable pointer overhead due to variables.

Turing Machines. We adopt TMs working on the boolean alphabet $\mathbb{B} := \{0,1\}$. For a study of logarithmic space complexity, one has to distinguish *input* space and *work* space, and to *not* count the input space for space complexity. On TMs, this amounts to having *two* tapes, a read-only *input tape* on the alphabet $\mathbb{B}_{\mathsf{I}} := \{0,1,\mathsf{L},\mathsf{R}\}$, where L and R are delimiter symbols for the start and the end of the input binary string, and an ordinary read-and-write *work tape* on the boolean alphabet extended with a blank symbol $\mathbb{B}_{\mathsf{W}} := \{0,1,\square\}$. To keep things simple, we use TMs without any output tape, the machine rather has two final states s_0 and s_1 encoding a boolean output—there are no difficulties in extending our results to TMs with an output tape. Let us call these machines *log-sensitive TM*.

A log-sensitive TM M computes the function $f : \mathbb{B}^* \to \mathbb{B}$ by a sequence of transitions $\rho : C_M(i) \to^n C'_M(f(i))$ where $i \in \mathbb{B}^*$, $C_M(i)$ is an initial configuration of M with input i and $C'_M(f(i))$ is a final configuration of M on the final state $s_{f(i)}$. We define the time of the run ρ as $T_{\mathrm{TM}}(\rho) := n$ and the space $S_{\mathrm{TM}}(\rho)$ of ρ as the maximum number of cells of the work tape used during the run ρ.

Random Access Machines. While we shall study in detail the encoding of Turing machines in the λ-calculus, we are not going to lay out the details of the simulation of the λ-calculus on RAMs. We shall provide an abstract machine for the λ-calculus and study its complexity using standard considerations for algorithmic analysis (which are grounded on RAMs), but without giving the details of the simulation. The RAM model we target has a read-only input register the space of which is not counted for space complexity, similarly to TMs. For the sake of completeness, we clarify the RAM cost models of reference: the logarithmic measure for time and Slot and Emde Boas (1988) $\mathtt{size}_b$ measure for space, counting 0 for unused registers and taking into account the logarithm of both the content and the index of used registers. Given a RAM R, we use $T_{\mathrm{RAM}}(\rho)$ and $S_{\mathrm{RAM}}(\rho)$ for the time and space used by R to reach a final configuration with a sequence of transitions ρ.

Abstract Machines and Abstract Implementations. Abstract machines do not specify how the (abstract) data structures of the machine are meant to be realized. In general an abstract machine can be implemented in various ways, inducing different, possibly incomparable performances. Therefore, it is not really possible to study the complexity of the machine without some assumptions about the implementation of its data structures. Now, the study of reasonable space requires to take into account the use, and especially the *size*, of pointers, which is instead usually omitted in the coarser study of reasonable time. In that context, indeed, pointers are assumed to be manipulable in constant time, which is safe because the omitted logarithmic factors are irrelevant for the required polynomial overhead. The more constrained study of space instead requires to clarify them. Switching to such a level of detail, apparently innocent gaps between the specification of a machine and how it is going to be implemented suddenly become relevant. To account for these subtleties, we specify for every construct of the abstract machine the space that it requires, and for every transition the time that it takes, both asymptotically.

Definition 8.1.1 (Abstract implementation)**.** *Let* M *be an abstract machine and* $\rho : \mathsf{init}(t_0) \to^* s$ *an initial run for* M*. An abstract implementation* I *for* M *is the assignment of asymptotic space costs* $|\cdot|^I_{\mathsf{sp}}$ *for every component of* s *and of asymptotic time costs* $|\cdot|^I_{\mathsf{tm}}$ *for every transition from* s*.*

Assigning costs to the state components provides the space cost $|s|^I_{\mathsf{sp}}$ of each state s, by summing over all components.

Definition 8.1.2 (Space and time of runs)**.** *Let* $\rho : s_0 \to^k s_k$ *be an initial run of an abstract machine* M *and* I *an abstract implementation for* M*.*

1. *The* I*-space cost of* ρ *is* $|\rho|^I_{\mathsf{sp}} := \max_{s\in\rho} |s|^I_{\mathsf{sp}}$.
2. *The* I*-time cost of* ρ *is* $|\rho|^I_{\mathsf{tm}} := \sum_{i=0}^{k-1} |s_i \to s_{i+1}|^I_{\mathsf{tm}}$.

Term	Env	Stack		Term	Env	Stack	
tu	e	π	$\to_{\mathsf{sea}}$	t	e	$(u,e)\cdot\pi$	
$\lambda x.t$	e	$c\cdot\pi$	$\to_{\beta}$	t	$[x\leftarrow c]\cdot e$	π	
x	e	π	$\to_{\mathsf{sub}}$	u	e'	π	if $e(x)=(u,e')$

TERMS $|u| := \mathcal{O}(\log|t_0|)$

CLOSURES $|(u,e)| := \mathcal{O}(|u|+|e|)$

STATES $|(u,e,\pi)| := \mathcal{O}(|u|+|e|+|\pi|)$

STACKS $|\epsilon| := \mathcal{O}(1)$, $|c\cdot\pi| := \mathcal{O}(|c|+|\pi|)$

ENVIRONMENTS $|\epsilon| := \mathcal{O}(1)$, $|[x\leftarrow c]\cdot e| := \mathcal{O}(|x|+|c|+|e|)$

TRANSITIONS $s\to s'$ $|\to|_{\mathsf{tm}} := \mathcal{O}(poly(|s|))$

FIGURE 8.1: Transitions and abstract implementation of the Naive KAM.

Term	Env	Stack		Term	Env	Stack	
tx	e	π	$\to_{\mathsf{sea_v}}$	t	$e\|_t$	$e(x)\cdot\pi$	
tu	e	π	$\to_{\mathsf{sea}_{\neg\mathsf{v}}}$	t	$e\|_t$	$(u,e\|_u)\cdot\pi$	if $u\notin\mathcal{V}$
$\lambda x.t$	e	$c\cdot\pi$	$\to_{\beta_{\mathsf{w}}}$	t	e	π	if $x\notin\mathsf{fv}(t)$
$\lambda x.t$	e	$c\cdot\pi$	$\to_{\beta_{\neg\mathsf{w}}}$	t	$[x\leftarrow c]\cdot e$	π	if $x\in\mathsf{fv}(t)$
x	e	π	$\to_{\mathsf{sub}}$	u	e'	π	if $e(x)=(u,e')$

where $e|_t$ denotes the restriction of e to the free variables of t.

FIGURE 8.2: Transitions of the Space KAM.

8.2 The Naive KAM and the Space KAM

We start considering a variant over the KAM, that we call the Naive KAM. The machine, reported in Fig. 8.1, appears the same as the traditional KAM. The difference, indeed, is in that it is implemented with sub-term pointers but *without* data pointers, which are instead employed in the implementation of the traditional KAM. This fact is expressed by the abstract implementation, again in Fig. 8.1. We need to eliminate data pointers since they are *space-hostile*, because (as discussed in Chapter 2) the number of environments is bounded only by $|\rho|_\beta$, that is, *time*. Data pointers have thus size $\mathcal{O}(\log|\rho|_\beta)$, entangling space with time, which is unreasonable for space and, even more problematic, do not allow an immediate form of garbage collection, because of *aliasing*. Indeed, different memory cells can be referenced more than once.

Abstract Implementation of the Naive KAM. Implementing the Naive KAM without data pointers means that environments *cannot* be implemented as linked lists, and the same is true for the stack, of which length also depends on the length of the run. The idea then is that they are implemented as unstructured *strings*, in a linear syntax. We abstract from the actual encoding, what we retain is the abstract implementation in Fig. 8.1, which captures its essence. The time cost of all $\to_{\mathrm{NaKAM}}$ transition depends polynomially on the size of the whole source state $|s|$, because the lack of data sharing forces to use a new string for the new stack and the new environment. To be precise, one could develop a finer analysis, thus obtaining slightly better bounds, but this would require entering in the details of the implementation and would not give a substantial advantage. As we shall see, indeed, the Naive KAM is unreasonable for time.

Since, we are concerned with space consumption, we define a space optimization of the Naive KAM. The Space KAM is derived from the Naive KAM implementing two modifications aimed at space efficiency: namely *unchaining* and *eager garbage collection*.

Unchaining. It is a folklore optimization for abstract machines bringing speed-ups with respect to both time and space, used *e.g.* by Sands et al. (2002), Wand (2007), Friedman et al. (2007), and Sestoft (1997). Its first systematic study is by Accattoli and Coen (2017), with respect to time. The optimization prevents the creation of chains of *renamings* in environments, that is, of delayed substitutions of variables for variables, of which the simplest shape in the KAM is:

$$[x_0 \leftarrow (x_1, [x_1 \leftarrow (x_2, [x_2 \leftarrow \dots])])]$$

where the links of the chain are generated by β-redexes having a variable as argument. On some families of terms, these chains keep growing and growing, leading to the quadratic dependency of the number of transitions from $|\rho|_\beta$.

Eager Garbage Collection. Beside the malicious chains connected to unchaining, the Naive KAM is not parsimonious with space also because there is no garbage collection (shortened to GC). In transition $\to_{\mathsf{sub}}$, the current environment is discarded, so something is collected, but this is not enough. It is thus natural to modify the machine as to maximize GC and space re-usage, that is, as to perform it *eagerly*.

The Space KAM. The Naive KAM optimized with both eager GC and unchaining (both optimizations are mandatory for space reasonability) is here called Space KAM and it is defined in Fig. 8.2. The data structures, namely closures and (local) environments, are defined as before, the changes concern the machine transitions only. Unchaining is realized by transition $\to_{\mathsf{sea_v}}$, while eager garbage collection is realized mainly by transition $\to_{\beta_{\mathsf{w}}}$, which collects the argument if the variable of the β redex does not occur. Transitions $\to_{\mathsf{sea}_{\neg\mathsf{v}}}$ and $\to_{\mathsf{sea_v}}$ also contribute to implement the GC, by restricting the environment to the occurring variables, when the environment is propagated to sub-terms. As a consequence, we obtain the following invariant.

Lemma 8.2.1 (Environment domain invariant)**.** *Let s be a Space KAM reachable state. Then $dom(e) = \mathsf{fv}(t)$ for every closure (t, e) in s.*

Because of the invariant, which concerns also the closure given by the active term and the local environment of the state, the variable transition simplifies as follows:

Term	Env	Stack		Term	Env	Stack
x	$[x \leftarrow (u, e)]$	π	$\to_{\mathsf{sub}}$	u	e	π

The abstract implementation of the Space KAM is the same of the Naive KAM, as the GC has a time cost which however stays within the polynomial (in the size of the states) cost of the transitions. It is mandatory that it is implemented by naively and repeatedly checking whether variables occur, and *not* via pointers or counters, as they would add an unreasonable space overhead. This fact is implicit in using the same abstract implementation of the Naive KAM, as a less naive GC would alter the space requirements.

8.3 Encoding and Moving over Strings

We now turn to the analysis of the encoding of TMs, taking as reference the one by Dal Lago and Accattoli (2017) based over the Scott encoding of strings. The first key step is understanding how to scroll Scott strings.

Encoding alphabets. Let $\Sigma = \{a_1, \dots, a_n\}$ be a finite alphabet. Elements of Σ are encoded in the λ-calculus in accordance to a fixed (but arbitrary) total order of the elements of Σ as follows:

$$\lceil a_i \rceil^\Sigma := \lambda x_1 \dots . \lambda x_n . x_i \ .$$

Note that the representation of an element $\lceil a_i \rceil^\Sigma$ requires a number of constructors that is linear (and not logarithmic) in $|\Sigma| = n$. Since the alphabet Σ shall not depend on the input of the TM, however, the cost in space is actually constant.

Encoding strings. A string in $s \in \Sigma^*$ is represented by a term $\overline{s}^{\Sigma^*}$, defined by induction on s as follows:

$$\overline{\varepsilon}^{\Sigma^*} := \lambda x_1.\ldots.\lambda x_n.\lambda y.y \ , \ \ \overline{a_i r}^{\Sigma^*} := \lambda x_1.\ldots.\lambda x_n.\lambda y.x_i \overline{r}^{\Sigma^*}.$$

Note that the representation depends on the cardinality of Σ. As before, however, the alphabet is a fixed parameter, and so such a dependency is irrelevant.

We now explain how to obtain that $\left\| \overline{s}^{\Sigma^*} \right\| = |\overline{s}^{\Sigma^*}|$, as announced in Sect. 8.1. Note that in $\overline{s}^{\Sigma^*}$ every variable occurrence is bound inside the list of binders immediately preceding the occurrence. Therefore, if de Bruijn indices are used to represent λ-terms, one needs only indices—that is, variable pointers—between 1 and $|\Sigma|$, that is, of constant size. Note that, similarly, if variables are represented with textual names, again having only $|\Sigma|$ distinct names is enough *if* one permits that different sequences of abstractions re-use the same names, that is, if one accepts Barendregt's convention to be violated. Remarkably, a notable folklore property of the (Space) KAM is that its implementation theorem does not need Barendregt's convention to hold.

Recursion and Fix-Points The encoding of TMs crucially relies on the use of a fix-point operator to implement recursion. Precisely, fix-points are used to model the transition function, making a copy of the (sub-term encoding the) transition table at each step. It is the only point of the encoding where duplication occurs, and it is thus where the expressive power is encapsulated. The rest of the encoding is affine—note that the representation of strings is affine.

Fix-Points and Toy Scrolling Algorithms. To understand the delicate interplay between the space of the KAM and fix-points, we analyze it via simple toy algorithms on strings. The first, simplest one is the *consuming scrolling algorithm*: going through an input string s doing nothing and accepting when arriving at the end of the string, without having to preserve the string itself—the aim is just to see the space used for scrolling a string. The toy algorithm is a very rough approximations of the moving of TMs over a tape, which is the most delicate aspect of the space reasonable simulation of TMs in the λ-calculus that we shall develop. It is used to illustrate the key aspects of the problems that arise and of their solutions, without having to deal with all the details of the encoding of TMs at once. On TMs, scrolling a string obviously runs in constant space, and on log-sensitive TMs the consuming aspect cannot be modeled—we shall consider non-consuming scrolling later in this section.

We encode the algorithm as a λ-term over Scott strings, where a fix-point combinator is used to iterate over the (term t_s encoding) the input string s. Since the input string s is consumed in the process, the normal form would be the encoding of the accepting state s_1 of the TM, which for simplicity here is simply given by the identity combinator I.

We use Turing's fix-point combinator and the boolean alphabet $\mathbb{B} := \{0,1\}$. Let fix be the term $\theta\theta$, where $\theta := \lambda x.\lambda y.y(xxy)$. Given a term u, $\mathsf{fix}\, u$ is a fix-point of u.

$$\begin{array}{lcccccl} \mathsf{fix}\, u & = & (\lambda x.\lambda y.y(xxy))\theta u & & & & \\ & \to_\beta & (\lambda y.y(\theta\theta y))u & \to_\beta & u(\theta\theta u) & = & u(\mathsf{fix}\, u) \end{array}$$

Algorithms moving over binary Scott strings always follow the same structure. They are given by the fix-point iteration of a term that does pattern matching on the leftmost character of the string and for each of the possible outcomes (in our case, first character is 0, 1, or the string is empty) does the corresponding action. The general term is $\mathsf{fix}\,(\lambda f.\lambda z.zA_0A_1A_\varepsilon)$, where f is the variable for the recursive call and A_0, A_1, and A_ε represent the three actions, which in our case are simply given by $A_0 = A_1 = f$ and $A_\varepsilon = \mathsf{I}$, using the identity I as encoding of the accepting state.

Proposition 8.3.1. *Let* $s \in \mathbb{B}^*$ *and* $\mathtt{toy} := \mathsf{fix}\,(\lambda f.\lambda z.zff\mathsf{I})$.

1. $\mathtt{toy}\,\overline{s}^{\mathbb{B}} \to_{wh}^{\Theta(|s|)} \mathsf{I}$.
2. *The Naive KAM evaluates* $\mathtt{toy}\,\overline{s}^{\mathbb{B}}$ *in space* $\Omega(2^{|s|})$.
3. *The Space KAM evaluates* $\mathtt{toy}\,\overline{s}^{\mathbb{B}}$ *in space* $\Theta(\log|s|)$.

Some preliminary definitions and lemmata are required to prove the statement. Let $s := b_1 \dots b_n \cdot \varepsilon$ be a string of length $n \geq 0$. Then, we can define e_i, e'_i, e''_i for $0 \leq i \leq n$ as follows:

$$\begin{array}{lcllcl} e_0 & := & [x{\leftarrow}(\theta,\epsilon)]\cdot[y{\leftarrow}(\texttt{toyaux},\epsilon)] & e_{i+1} & := & [x{\leftarrow}(x,e_i)]\cdot[y{\leftarrow}(y,e_i)] \\ e'_i & := & [z{\leftarrow}(\overline{b_{i+1}..b_n\cdot\varepsilon},e''_i)]\cdot[f{\leftarrow}(xxy,e_i)] & & & \\ e''_0 & := & \epsilon & e''_{i+1} & := & [x_\varepsilon{\leftarrow}(\mathsf{I},e'_i)]\cdot[x_1{\leftarrow}(f,e'_i)]\cdot[x_0{\leftarrow}(f,e'_i)]\cdot e''_i \end{array}$$

One can easy notice that the sizes of e_i, e'_i, e''_i are exponential in i.

Lemma 8.3.2. $(y(xxy), e_i, (\overline{s}, e''_i)) \rightarrow_{\text{NaKAM}}^{\Omega(|s|)} (\mathsf{I}, e'_{i+|s|}, \epsilon)$

Proof. By induction on the structure of s.

Term	Env	Stack	
$y(xxy)$	e_i	$(\overline{s}, e''_i)$	
y	e_i	$(xxy, e_i)\cdot(\overline{s}, e''_i)$	unfold e_i
$\texttt{toyaux} := \lambda f.\lambda z.zff\mathsf{I}$	ϵ	$(xxy, e_i)\cdot(\overline{s}, e''_i)$	
$zff\mathsf{I}$	$[z{\leftarrow}(\overline{s}, e''_i)]\cdot[f{\leftarrow}(xxy, e_i)] =: e'_i$	ϵ	
z	$[z{\leftarrow}(\overline{s}, e''_i)]\cdot[f{\leftarrow}(xxy, e_i)]$	$(f, e'_i)\cdot(f, e'_i)\cdot(\mathsf{I}, e'_i)$	
$\overline{s}$	e''_i	$(f, e'_i)\cdot(f, e'_i)\cdot(\mathsf{I}, e'_i)$	

Case $s = \varepsilon$.

Term	Env	Stack
$\overline{s} := \lambda x_0.\lambda x_1.\lambda x_\varepsilon.x_\varepsilon$	e''_i	$(f, e'_i)\cdot(f, e'_i)\cdot(\mathsf{I}, e'_i)$
x_ε	$[x_\varepsilon{\leftarrow}(\mathsf{I}, e'_i)]\cdot[x_1{\leftarrow}(f, e'_i)]\cdot[x_0{\leftarrow}(f, e'_i)]\cdot e''_i$	ϵ
I	e'_i	ϵ

Case $s = b\cdot r$.

Term	Env	Stack
$\overline{s} := \lambda x_0.\lambda x_1.\lambda x_\varepsilon.x_b\overline{r}$	e''_i	$(f, e'_i)\cdot(f, e'_i)\cdot(\mathsf{I}, e'_i)$
$x_b\overline{r}$	$[x_\varepsilon{\leftarrow}(\mathsf{I}, e'_i)]\cdot[x_1{\leftarrow}(f, e'_i)]\cdot[x_0{\leftarrow}(f, e'_i)]\cdot e''_i =: e''_{i+1}$	ϵ
x_b	$[x_\varepsilon{\leftarrow}(\mathsf{I}, e'_i)]\cdot[x_1{\leftarrow}(f, e'_i)]\cdot[x_0{\leftarrow}(f, e'_i)]\cdot e''_i$	$(\overline{r}, e''_{i+1})$
f	$[z{\leftarrow}(\overline{s}, e''_i)]\cdot[f{\leftarrow}(xxy, e_i)]$	$(\overline{r}, e''_{i+1})$
xxy	$[x{\leftarrow}(x, e_{i-1})]\cdot[y{\leftarrow}(y, e_{i-1})]$	$(\overline{r}, e''_{i+1})$
x	$[x{\leftarrow}(x, e_{i-1})]\cdot[y{\leftarrow}(y, e_{i-1})]$	$(x, e_i)\cdot(y, e_i)\cdot(\overline{r}, e''_{i+1})$
		unfolding e_{i-1}
$\theta := \lambda x.\lambda y.y(xxy)$	ϵ	$(x, e_i)\cdot(y, e_i)\cdot(\overline{r}, e''_{i+1})$
$y(xxy)$	$[x{\leftarrow}(x, e_i)]\cdot[y{\leftarrow}(y, e_i)] =: e_{i+1}$	$(\overline{r}, e''_{i+1})$ *i.h.*
I	$e'_{i+\lvert s\rvert}$	ϵ

□

Lemma 8.3.3. *The Space KAM executes the reduction* $(y(xxy), e_0, (\overline{s}, \epsilon)) \rightarrow_{SpKAM}^{\Theta(|s|)} (\mathsf{I}, \epsilon, \epsilon)$ *consuming* $\mathcal{O}(\log(|s|))$ *space.*

Proof. By induction on the structure of s.

Term	Env	Stack
$y(xxy)$	e_0	$(\overline{s}, \epsilon)$
y	$[y{\leftarrow}(\texttt{toyaux}, \epsilon)]$	$(xxy, e_0)\cdot(\overline{s}, \epsilon)$
$\texttt{toyaux} := \lambda f.\lambda z.zff\mathsf{I}$	ϵ	$(xxy, e_0)\cdot(\overline{s}, \epsilon)$
$zff\mathsf{I}$	$[z{\leftarrow}(\overline{s}, \epsilon)]\cdot[f{\leftarrow}(xxy, e_0)]$	ϵ
z	$[z{\leftarrow}(\overline{s}, \epsilon)]$	$(xxy, e_0)\cdot(xxy, e_0)\cdot(\mathsf{I}, \epsilon)$
$\overline{s}$	ϵ	$(xxy, e_0)\cdot(xxy, e_0)\cdot(\mathsf{I}, \epsilon)$

Case $s = \varepsilon$.

Term	Env	Stack
$\overline{s} := \lambda x_0.\lambda x_1.\lambda x_\varepsilon.x_\varepsilon$	ϵ	$(xxy, e_0)\cdot(xxy, e_0)\cdot(\mathsf{I}, \epsilon)$
x_ε	$[x_\varepsilon \leftarrow (\mathsf{I}, \epsilon)]$	ϵ
I	ϵ	ϵ

Case $s = b{\cdot}r$.

Term	Env	Stack	
$\overline{s} := \lambda x_0.\lambda x_1.\lambda x_\varepsilon.x_b\overline{r}$	ϵ	$(xxy, e_0)\cdot(xxy, e_0)\cdot(\mathsf{I}, \epsilon)$	
$x_b\overline{r}$	$[x_b \leftarrow (xxy, e_0)]$	ϵ	
x_b	$[x_b \leftarrow (xxy, e_0)]$	$(\overline{r}, \epsilon)$	
xxy	e_0	$(\overline{r}, \epsilon)$	
x	$[x \leftarrow (\theta, \epsilon)]$	$(\theta, \epsilon)\cdot(\mathtt{toyaux}, \epsilon)\cdot(\overline{r}, \epsilon)$	
$\theta := \lambda x.\lambda y.y(xxy)$	ϵ	$(\theta, \epsilon)\cdot(\mathtt{toyaux}, \epsilon)\cdot(\overline{r}, \epsilon)$	
$y(xxy)$	e_0	$(\overline{r}, \epsilon)$	*i.h.*
I	ϵ	ϵ	

The space bound is proved since there is a static bound (8) on the number of closures during the execution. □

Now we are able to prove the desired result.

Proposition 8.3.4. *Let* $s \in \mathbb{B}^*$ *and* $\mathtt{toy} := \mathsf{fix}\,(\lambda f.\lambda z.zff\mathsf{I})$.

1. $\mathtt{toy}\,\overline{s}^{\mathbb{B}} \to_{wh}^{\Theta(|s|)} \mathsf{I}$.
2. *The Naive KAM evaluates* $\mathtt{toy}\,\overline{s}^{\mathbb{B}}$ *in space* $\Omega(2^{|s|})$.
3. *The Space KAM evaluates* $\mathtt{toy}\,\overline{s}^{\mathbb{B}}$ *in space* $\Theta(\log|s|)$.

Proof.

1. This point follows from the implementation theorem, applied to the sequence of point 3.
2. We prove the statement executing $\mathtt{toy}\,\overline{s}$ with the Naive KAM. We set $\mathtt{toyaux} := \lambda f.\lambda z.zff\mathsf{I}$.

Term	Env	Stack	
$\mathtt{toy}\,\overline{s}$	ϵ	ϵ	
$\mathtt{toy} := \mathsf{fix}\,\mathtt{toyaux}$	ϵ	$(\overline{s}, \epsilon)$	
$\mathsf{fix} := \theta\theta$	ϵ	$(\mathtt{toyaux}, \epsilon)\cdot(\overline{s}, \epsilon)$	
$\theta := \lambda x.\lambda y.y(xxy)$	ϵ	$(\theta, \epsilon)\cdot(\mathtt{toyaux}, \epsilon)\cdot(\overline{s}, \epsilon)$	
$y(xxy)$	$[x \leftarrow (\theta, \epsilon)]\cdot[y \leftarrow (\mathtt{toyaux}, \epsilon)] =: e_0$	$(\overline{s}, \epsilon)$	L. 8.3.2
I	$e'_{\lvert s\rvert}$	ϵ	

   The space bound is proved since $e'_{|s|}$ is exponential in $|s|$.

3. We prove the statement executing $\mathtt{toy}\,\overline{s}$ with the Space KAM. We set $\mathtt{toyaux} := \lambda f.\lambda z.zff\mathsf{I}$.

Term	Env	Stack	
$\mathtt{toy}\,\overline{s}$	ϵ	ϵ	
$\mathtt{toy} := \mathsf{fix}\,\mathtt{toyaux}$	ϵ	$(\overline{s}, \epsilon)$	
$\mathsf{fix} := \theta\theta$	ϵ	$(\mathtt{toyaux}, \epsilon)\cdot(\overline{s}, \epsilon)$	
$\theta := \lambda x.\lambda y.y(xxy)$	ϵ	$(\theta, \epsilon)\cdot(\mathtt{toyaux}, \epsilon)\cdot(\overline{s}, \epsilon)$	
$y(xxy)$	$[x \leftarrow (\theta, \epsilon)]\cdot[y \leftarrow (\mathtt{toyaux}, \epsilon)] =: e_0$	$(\overline{s}, \epsilon)$	L 8.3.3
I	ϵ	ϵ	

   The space bound is proved considering the bound in Lemma 8.3.3.

□

We can see that the Naive KAM is desperately inefficient for space, while the Space KAM works within reasonable bounds. It turns out, however, that the Space KAM is still not enough in order to obtain a space reasonable simulation of TMs. The problem now concerns the standard encoding of TMs and its managing of the tapes, rather than the use of space by the abstract machine itself. The issues can be explained using further toy algorithms.

String-Preserving Scrolling. Consider the same scrolling algorithm as above, except that now the input string s is *not* consumed by the moving over s, that is, it has to be given back as output of the λ-term implementing the algorithm. This variant is a step forward towards approximating what happens to the tapes of TMs *during* the computation: the TM moves over the tapes *without consuming them*, it is only at the end of the computation that the TM can be seen as throwing them away. There are two ways of implementing the new algorithm:

1. *Local copy*: moving over the string s while accumulating in a new string r the characters that have already been visited, returning r.
2. *Global copy*: making a copy r of the string s, and then moving over s in a consuming way, returning r.

Local Copy. The local approach is the one underlying the reference encoding of TMs. In particular, it is almost affine, as duplication is isolated in the fix-point. The λ-term `localCopy` uses the same fix-point schema as before, but with different, more involved action terms A_0, A_1, and A_ε.

Proposition 8.3.5. *Let $s \in \mathbb{B}^*$.*

1. $\texttt{localCopy}\,\overline{s}^{\mathbb{B}} \to_{wh}^{\Theta(|s|)} \overline{s}^{\mathbb{B}}$.
2. *The Space KAM evaluates* $\texttt{localCopy}\,\overline{s}^{\mathbb{B}}$ *in space* $\Theta(|s| \log |s|)$.

Proof. The proof can be recovered considering the full encoding of TMs into the λ-calculus. This proposition is just about a subset of it. □

The $\Theta(|s| \log |s|)$ bound in point 2 is problematic for the space reasonable modeling in the λ-calculus of both the input and the work tapes, for different reasons.

Work Tape and Separate Address Spaces. For a space-reasonable managing of the work tape, the local algorithm should rather work in space $\mathcal{O}(|s|)$. This improvement can be realized by a finer complexity analysis. In Prop. 8.3.5.2, the cost comes from the use of $\mathcal{O}(|s|)$ sub-term pointers to the code $\texttt{localCopy}\,\overline{s}^{\mathbb{B}}$ used by the Space KAM. These pointers have size $\mathcal{O}(\log |s|)$ because $|\texttt{localCopy}\,\overline{s}^{\mathbb{B}}| = \mathcal{O}(|s|)$, that is, the size of `localCopy` is independent of $|s|$ and thus constant. A close inspection of the Space KAM run in Prop. 8.3.5.2 shows that, of the $\mathcal{O}(|s|)$ pointers used, only $\mathcal{O}(1)$ of them actually point to $\overline{s}^{\mathbb{B}}$, while all the others (that is, an $\mathcal{O}(|s|)$ amount) point to `localCopy`. Since `localCopy` is of size independent from $|s|$, if one admits separate address spaces for `localCopy` and $\overline{s}^{\mathbb{B}}$ then the pointers to `localCopy` have size $\mathcal{O}(1)$. Therefore, one obtains that the space cost is given by

$$\underbrace{\mathcal{O}(|s|) \cdot \mathcal{O}(1)}_{\text{pointers to } \texttt{localCopy}} \quad + \quad \underbrace{\mathcal{O}(1) \cdot \mathcal{O}(\log |s|)}_{\text{pointers to } \overline{s}^{\mathbb{B}}} = \mathcal{O}(|s|).$$

From now on then, the first argument of the code of the Space KAM—that in the encoding of TMs shall represent the input—has a dedicated address space.

Proposition 8.3.6 (Linear Space Local-Copy Scrolling)**.** *Let $s \in \mathbb{B}^*$. The Space KAM evaluates* $\texttt{localCopy}\,\overline{s}^{\mathbb{B}}$ *in space $\mathcal{O}(|s|)$ if* `localCopy` *and $\overline{s}^{\mathbb{B}}$ have separate space addresses.*

Input Tape and Global Copy. For the input tape, a linear space bound for scrolling is unreasonable, if one aims at preserving logarithmic space complexity. For meeting the required $\mathcal{O}(\log|s|)$ bound, we need a more radical solution, which is possible because the tape is read-only.

The first step is the straightforward modification of the consuming scrolling algorithm into a global-copy string-preserving algorithm: it is enough to capture the input at the beginning with an extra abstraction λx and to give it back at the end with the action A_ε. Namely, let $\texttt{globalCopy} := \lambda x.(\mathsf{fix}\,(\lambda f.\lambda s'.s'ffx)x)$. Clearly, this approach breaks the almost affinity of the encoding, as copying is no longer encapsulated only in the fix-point. Interestingly, the space cost stays logarithmic, because the global copy of the input (note that there actually is a copy for every iteration of the fix-point) is not performed by the Space KAM, which instead copies a *pointer* to it and only once.

Proposition 8.3.7. *Let $s \in \mathbb{B}^*$.*

1. $\texttt{globalCopy}\,\overline{s}^{\mathbb{B}} \to_{wh}^{\Theta(|s|)} \overline{s}^{\mathbb{B}}$.
2. *The Space KAM evaluates* $\texttt{globalCopy}\,\overline{s}^{\mathbb{B}}$ *in space* $\Theta(\log|s|)$.

Proof. We prove the second point of the statement by directly executing the Space KAM. The required lemma is reported below. The first point is then obtained as a corollary using the complexity and correctness properties of the Space KAM. Since $\mathbb{B}$ is the only alphabet that we are using, we remove all the superscripts. Let us define $t := \lambda f.\lambda s'.s'ffz$.

Term	Environment	Stack	
$\texttt{globalCopy}\,\overline{s}$	ϵ	ϵ	$\to$
$\texttt{globalCopy} := \lambda z.(\mathsf{fix}\,tz)$	ϵ	$(\overline{s},\epsilon) =: s^{\mathrm{K}}$	$\to$
$\mathsf{fix}\,tz$	$[z{\leftarrow}s^{\mathrm{K}}]$	ϵ	$\to$
$\mathsf{fix}\,t$	$[z{\leftarrow}s^{\mathrm{K}}]$	s^{K}	$\to$
$\mathsf{fix} := \theta\theta$	ϵ	$(t,[z{\leftarrow}s^{\mathrm{K}}])\cdot s^{\mathrm{K}}$	$\to$
$\theta := \lambda x.\lambda y.y(xxy)$	ϵ	$(\theta,\epsilon)\cdot(t,[z{\leftarrow}s^{\mathrm{K}}])\cdot s^{\mathrm{K}}$	$\to^{\Theta(\lvert s\rvert)}$ (L. 8.3.8)
$\overline{s}$	ϵ	ϵ	

□

Lemma 8.3.8. *Let $s \in \mathbb{B}^*$ and $t := \lambda f.\lambda s'.s'ffz$. Then $(\theta,\epsilon,(\theta,\epsilon)\cdot(t,[z{\leftarrow}(u,e)])\cdot s^{\mathrm{K}}) \to_{SpKAM}^{\Theta(|s|)} (u,e,\epsilon)$ and moreover the space used is $\mathcal{O}(|e|+\log|s|+\log|u|)$.*

Proof. We proceed by induction on the structure of s. The first steps are common to both the base case and the induction step. We define $\mathsf{fix}^{\mathrm{K}} := (xxy,[y{\leftarrow}(t,[z{\leftarrow}(u,e)])]\cdot[x{\leftarrow}(\theta,\epsilon)])$.

Term	Environment	Stack	
$\theta := \lambda x.\lambda y.y(xxy)$	ϵ	$(\theta,\epsilon)\cdot(t,[z{\leftarrow}(u,e)])\cdot s^{\mathrm{K}}$	$\to$ (L. E.0.4)
$t := \lambda f.\lambda s'.s'ffz$	$[z{\leftarrow}(u,e)]$	$\mathsf{fix}^{\mathrm{K}}\cdot s^{\mathrm{K}}$	$\to^2$
$s'ffz$	$[f{\leftarrow}\mathsf{fix}^{\mathrm{K}}]\cdot[s'{\leftarrow}s^{\mathrm{K}}]\cdot[z{\leftarrow}(u,e)]$	ϵ	$\to^3$
s'	$[s'{\leftarrow}s^{\mathrm{K}}]$	$\mathsf{fix}^{\mathrm{K}}\cdot\mathsf{fix}^{\mathrm{K}}\cdot(u,e)$	$\to$
$\overline{s}$	ϵ	$\mathsf{fix}^{\mathrm{K}}\cdot\mathsf{fix}^{\mathrm{K}}\cdot(u,e)$	

Base case: $s = \varepsilon$.

Term	Environment	Stack	
$\overline{s} := \lambda x_0.\lambda x_1.\lambda x_\varepsilon.x_\varepsilon$	ϵ	$\mathsf{fix}^{\mathrm{K}}\cdot\mathsf{fix}^{\mathrm{K}}\cdot(u,e)$	$\to^4$
u	e	ϵ	

Inductive case: $s : b\cdot r$ where $b \in \{0,1\}$.

Term	Environment	Stack	
$\bar{s} := \lambda x_0.\lambda x_1.\lambda x_\varepsilon.x_b\bar{r}$	ϵ	$\mathsf{fix}^{\mathrm{K}}\cdot\mathsf{fix}^{\mathrm{K}}\cdot(u,e)$	$\to^3$
$x_b\bar{r}$	$[x_b\leftarrow\mathsf{fix}^{\mathrm{K}}]$	ϵ	$\to$
x_b	$[x_b\leftarrow\mathsf{fix}^{\mathrm{K}}]$	$(\bar{r},\epsilon) =: r^{\mathrm{K}}$	$\to$
xxy	$[y\leftarrow(t,[z\leftarrow(u,e)])]\cdot[x\leftarrow(\theta,\epsilon)]$	r^{K}	$\to^2$
x	$[x\leftarrow(\theta,\epsilon)]$	$(\theta,\epsilon)\cdot(t,[z\leftarrow(u,e)])\cdot r^{\mathrm{K}}$	$\to$
θ	ϵ	$(\theta,\epsilon)\cdot(t,[z\leftarrow(u,e)])\cdot r^{\mathrm{K}}$	$\to^{\Theta(\lvert r\rvert)}$ *i.h.*
u	e	ϵ	

About space, it is immediate to see that all computations are constrained in space $\mathcal{O}(|e| + \log|s| + \log|u|)$, since at any point during the computation there is a *bounded* number of closures, independent from both s and u. □

The next step is refining this scheme as to implement a read-only tape. A slight digression is in order.

Intrinsic and Mathematical Tape Representations. A TM tape is a string plus a distinguished position, representing the head. There are two tape representations, dubbed *intrinsic* and *mathematical* by Emde Boas (2012). The *intrinsic* one represents both the string s and the current position of the head as the triple $s = s_l \cdot h \cdot s_r$, where s_l and s_r are the prefix and suffix of s surrounding the character h read by the head. This is the representation underlying the local-copy scrolling algorithm (and the reference encoding of TM). The *mathematical* representation, instead, is simply given by the index $n \in \mathbb{N}$ of the head position, that is, the triple $s_l \cdot h \cdot s_r$ is replaced by the pair $(s, |s_l| + 1)$.

Mathematical Input and Global Copy. Given a mathematical read-only tape (s, n), one can use the global-copy scrolling scheme for a simulation in the λ-calculus in space $\mathcal{O}(\log|s|)$. The idea is to represent n as a binary string $\lfloor n \rfloor$. Since $n \leq |s|$, we have $|\lfloor n \rfloor| \leq \log|s|$. Moreover, it is possible to pass from $\lfloor n \rfloor$ to $\lfloor n+1 \rfloor$ or $\lfloor n-1 \rfloor$—which is needed to move the position of the head—in $\mathcal{O}(\log|s|)$ space. Then one shows that in the λ-calculus the following is doable in space $\mathcal{O}(\log|s|)$: given (s, n), returning (s, n) plus the n-th character s_n of s, by making a global copy of the tape and scrolling the current copy of n positions, extracting the head s_n of the obtained suffix, and discarding the tail.

Two remarks. First, this approach works because the tape is read-only, so that one can keep making global copies of the same immutable tape, and only changing the index of the head. Second, there is a (reasonable) time slowdown, because at each read the simulation has to scroll sequentially the input tape to get to the n-th character.

8.4 The Space KAM is Reasonable for Space

We are ready for our main result, which is based on a new variant (in Appendix D) over Accattoli and Dal Lago encoding of TMs into the λ-calculus. The key points are:

- *Refined TMs*: the notion of TM we work with is log-sensitive TMs with mathematical input tape and intrinsic work tape (the definition is laid out in Appendix D.).
- *CPS and indifference*: following Dal Lago and Accattoli (2017), the encoding is in *continuation-passing style*, and carefully designed (by adding some η-expansions) as to fall into the *deterministic λ-calculus* $\Lambda_{\mathtt{det}}$, a particularly simple fragment of the λ-calculus where the right sub-terms of applications can only be variables or abstractions and where, consequently, call-by-name and call-by-value collapse on the same evaluation strategy $\to_{det}$. We shall exploit this *indifference* property in Sect. 8.6.
- *Duplication*: duplication is isolated in the unfolding of fix-points and in the managing of the input tape, all other operations are affine.

Theorem 8.4.1 (TMs are simulated by the Space KAM in reasonable space)**.** *There is an encoding $\bar{\cdot}$ of log-sensitive TMs into $\Lambda_{\mathtt{det}}$ such that if the run ρ of the TM M on input $i \in \mathbb{B}^*$:*

1. Termination*: ends in s_b with $b \in \mathbb{B}$, then there is a complete sequence $\sigma : \overline{M}\,\bar{i} \to^n_{det} \overline{s_b}$ where $n = \Theta((T_{TM}(\rho)+1)\cdot|i|\cdot\log|i|)$.*
2. Divergence*: diverges, then $\overline{M}\,\bar{i}$ is $\to_{det}$-divergent.*
3. Space KAM*: the space used by the Space KAM to simulate the evaluation of point 1 is $\mathcal{O}(S_{TM}(\rho)+\log|i|)$ if $\overline{M}$ and $\bar{i}$ have separate address spaces.*

The previous theorem provides the subtle and important half of the space reasonability result. The first two points establish the qualitative part of the simulation in the λ-calculus, together with the time bound (with respect to the number of β steps). The third point provides the space result for the Space KAM. They are connected by the fact that the Space KAM implements closed call-by-name (that coincides with $\to_{det}$ in $\Lambda_{\mathtt{det}}$).

The other half of the result amounts to showing that the Space KAM can be simulated on RAMs within a constant multiplicative overhead. The idea is that it can clearly be simulated reasonably by a multi tape TM using one work tape for the active term, one for the environment, and one for the stack, which in turn can be reasonably simulated by a RAM. We use RAM rather TM in the statement for uniformity with the works on time (which is relevant for the discussions in Sect. 8.5).

Theorem 8.4.2 (Space KAM is simulated by RAMs in reasonable space)**.** *Let t be a closed λ-term. Every Space KAM run $\rho : \mathsf{init}(t) \to^*_{SpKAM} s$ can be implemented on RAMs in space $\mathcal{O}(|\rho|_{\mathsf{sp}})$.*

From Theorems 8.4.1 and 8.4.2 follows the main result of this section.

Theorem 8.4.3 (The Space KAM is reasonable for space)**.** *Closed CbN evaluation $\to_{wh}$ and the space of the Space KAM provide a reasonable space cost model for the λ-calculus.*

The Space KAM is not Reasonable for Natural Time. We complete our study of the Space KAM by analyzing its time behavior. For natural time (in our case, the number of Closed CbN β steps), the Space KAM is unreasonable, because simulating Closed CbN at times requires exponential overhead. The number of transitions of the Space KAM is reasonable, while it is the cost of single transitions, thus of the manipulation of data structures, that can explode. The failure stems from the lack of data sharing, which on the other hand we showed being mandatory for space reasonability. Essentially, there are size exploding families such that their Space KAM run produces environments of size exponential in the number of β steps/transitions, which is the key point in the proof of the next proposition.

Proposition 8.4.4 (Space KAM natural time overhead explosion)**.** *There is a family $\{t_n\}_{n\in\mathbb{N}}$ of closed λ-terms such that there is a complete evaluation $\rho_n : t_n \to^n_{wh} u_n$ is simulated by Space KAM runs σ_n taking both space and time exponential in n, that is, $|\sigma_n|_{\mathsf{sp}} = |\sigma_n|_{\mathsf{tm}} = \Omega(2^n)$.*

Before the main proof we need some preliminaries. We define the following data structures:

$$\begin{array}{rclcrcl} e_0 & := & [x_0{\leftarrow}(\mathsf{I},\epsilon)] & & e_{n+1} & := & [x_{n+1}{\leftarrow}\pi_n]\cdot e_n \\ \pi_0 & := & (x_0 x_0, e_0) & & \pi_{n+1} & := & (x_0..x_{n+1}, e_{n+1}) \end{array}$$

We observe that the size of e_n is exponential in n, since
$e_{n+1} := [x_{n+1}{\leftarrow}\pi_n]\cdot e_n = [x_{n+1}{\leftarrow}(x_0..x_n, e_n)]\cdot e_n$ *i.e.* $|e_{n+1}| \geq 2|e_n|$ and thus $|e_n| \geq 2^n$.

We define C_n as follows:

$$\begin{array}{rcl} C_0 & := & \lambda x_0.\langle\cdot\rangle(x_0 x_0) \\ C_{n+1} & := & \lambda x_{n+1}.\langle\cdot\rangle(x_0 \dots x_{n+1}) \end{array}$$

Lemma 8.4.5. *If t contains $x_0,..,x_n$ free, then $(C_0\langle C_1\langle\cdots C_n\langle t\rangle\cdots\rangle\rangle|\mathsf{I},\epsilon,\epsilon) \to^{\Theta(n)}_{SpKAM} (t, e_n, \pi_n)$.*

Proof. Case 0.

Term	Env	Stack
$C_0\langle t\rangle \mathsf{I}$	ϵ	ϵ
$C_0\langle t\rangle := \lambda x_0.t(x_0x_0)$	ϵ	(I,ϵ)
$t(x_0x_0)$	$[x_0{\leftarrow}(\mathsf{I},\epsilon)]$	ϵ
t	$[x_0{\leftarrow}(\mathsf{I},\epsilon)]$	$(x_0x_0,[x_0{\leftarrow}(\mathsf{I},\epsilon)])$

Case $n+1$.

We observe that $C_0\langle C_1\langle ..C_n\langle C_{n+1}\langle t\rangle\rangle..\rangle\rangle \mathsf{I}$ can be rewritten to $C_0\langle C_1\langle ..C_n\langle u\rangle..\rangle\rangle \mathsf{I}$ when $u := C_{n+1}\langle t\rangle$. Of course u contains $x_0,..,x_n$ free. We can thus apply the *i.h.*.

Term	Env	Stack	
$C_0\langle C_1\langle ..C_n\langle C_{n+1}\langle t\rangle\rangle..\rangle\rangle \mathsf{I}$	ϵ	ϵ	*i.h.*
$C_{n+1}\langle t\rangle := \lambda x_{n+1}.t(x_0..x_{n+1})$	e_n	π_n	
$t(x_0..x_{n+1})$	$[x_{n+1}{\leftarrow}\pi_n]\cdot e_n =: e_{n+1}$	ϵ	
t	e_{n+1}	$(x_0..x_{n+1},e_{n+1}) =: \pi_{n+1}$	

□

Now we are able to prove the desired result.

Proposition 8.4.6 (Space KAM natural time overhead explosion)**.** *There is a family $\{t_n\}_{n\in\mathbb{N}}$ of closed λ-terms such that there is a complete evaluation $\rho_n : t_n \to_{wh}^n u_n$ is simulated by Space KAM runs σ_n taking both space and time exponential in n, that is, $|\sigma_n|_{\mathsf{sp}} = |\sigma_n|_{\mathsf{tm}} = \Omega(2^n)$.*

Proof. We define t_n as follows:

$$t_n \quad := \quad C_0\langle C_1\langle \cdots C_n\langle \lambda y.\mathsf{I}\rangle \cdots \rangle\rangle \mathsf{I}$$

Now, we execute it.

Term	Env	Stack	
$t_n := C_0\langle C_1\langle \cdots C_{n-1}\langle C_n\langle \lambda y.\mathsf{I}\rangle\rangle \cdots \rangle\rangle \mathsf{I}$	ϵ	ϵ	Lemma 8.4.5
$C_n\langle \lambda y.\mathsf{I}\rangle := \lambda x_n.(\lambda y.\mathsf{I})(x_0 \dots x_n)$	e_{n-1}	π_{n-1}	
$(\lambda y.\mathsf{I})(x_0 \dots x_n)$	e_n	ϵ	
$\lambda y.\mathsf{I}$	ϵ	π_n	
I	ϵ	ϵ	

The space consumed (and thus also the low-level time) is exponential in n because the size of e_n is exponential in n. □

8.5 Time vs Space

Here we discuss how to obtain, or approximate, reasonability for both space and time.

Reasonable Low-Level Time. Changing the time cost model from the number of β steps to the time taken by the Space KAM, which is a low-level notion of time, provides a reasonable time cost model. The key point is that the explosions of Prop. 8.4.4 never happen on λ-terms encoding TMs.

Theorem 8.5.1 (TMs are simulated by the Space KAM in reasonable low-level time)**.**

1. *Every TM run ρ can be simulated by the Space KAM in time $\mathcal{O}(poly(|\rho|))$.*
2. *Every Space KAM run $\rho : \mathsf{init}(t) \to_{SpKAM}^* s$ can be implemented on RAMs in time $\mathcal{O}(|\rho|_{\mathsf{tm}})$.*
3. *Closed CbN and the time of the Space KAM provide a reasonable time cost model for the λ-calculus.*

Proof. The first point is the only one which is non-trivial. We have already proved that the Space KAM can simulate TMs runs ρ in a number of *transitions* which is polynomial in $|\rho|$. However, this

does not necessarily means that the (low-level) time is also polynomial in ρ, see Proposition 8.4.4. About the execution of terms which are the image of the encoding of TMs into the λ-calculus, we can say however that the overhead stays polynomial. Indeed, the exponential blowup comes from the fact that environments are duplicated in an uncontrolled way. This does not happen in the execution of the encoding of TMs, where duplication is restrained to the fix-point operator and to the input components of the state. In other words, we duplicate only objects of fixed size, thus confirming the polynomial bound.

There is another, indirect, way of proving the same results. If the (low-level) time were exponential in $|\rho|$, then the space should be at least linear in $|\rho|$[1]. But we have proved that this is not the case since space is linearly related with the *space* consumption of ρ, and *not* with its length. □

The drawback of this solution is that one gives up the natural cost model for time. Moreover, the low-level time cost model can be very lax in comparison, as Prop. 8.4.4 shows.

The Interleaving Technique. Forster et al. (2020) show that, given one machine that is reasonable for time but not for space and one machine that is reasonable for space but not for time, it is possible to build a third machine that is reasonable for both space and time, by interleaving the two machines in a smart way. Despite being presented on a specific case, their construction is quite general (in fact it is not even limited to the λ-calculus), and can be adapted to our case (the two starting machines being the KAM and the Space KAM). The drawback of this solution is that it admits space exponential in time, as Prop. 8.4.4 shows.

Trading Time for Space. From a practical rather than theoretical point of view, there is a further *semi solution* that we now sketch. Tweaking the KAM with a synchronous reference counting GC one obtains a Shared Space KAM which is reasonable for time and *slightly unreasonable* for space. One could observe that the Shared Space KAM and the Space KAM are indeed strongly bisimilar[2] and use the same number of closures. Then, the number of data pointers used by the Shared Space KAM is no longer entangled with the number of β-steps (as for the KAM), it is instead related to the space cost—this is the effect of GC plus unchaining of the Space KAM. Therefore, data pointers add a space overhead that is logarithmic in the space of the Space KAM. Also the GC mechanism, based on reference counting, adds the same logarithmic factor, while certainly costs time, though still remaining polynomial. This way, if the Space KAM uses $\mathcal{O}(f(n))$ space, then the Shared Space KAM operates in $\mathcal{O}(f(n)\log f(n))$ space. Such an overhead is not reasonable but not too unreasonable, and probably the best compromise between reasonability and efficiency for the practice of implementing functional programs.

8.6 Call-by-Value and Other Strategies

How robust is our space cost model to changes of the evaluation strategy? The short answer is *very robust*.

Closed Call-by-Value. We refer to weak call-by-value evaluation with closed terms as to *Closed CbV* . Our results smoothly adapt to such a setting, as we now explain.

First, it is easy to adapt the Space KAM to Closed CbV . The LAM (Leroy Abstract Machine) is a right-to-left[3] CbV analogue of the KAM defined by by Accattoli et al. (2014a) and modeled after the ZINC by Leroy (1990) (whence the name). It uses a further data structure, the *dump*, storing the left sub-terms of applications yet to be evaluated. It is upgraded to the Space LAM in Fig. 8.3 by removing data pointers and adding GC. Unchaining comes for free in CbV, if one considers values to be only abstractions, see Accattoli and Coen (2017).

[1] This is because space cannot be less than logarithmic in time.
[2] Actually, one should add the *unchaining* optimization to the KAM, which is standard.
[3] The argument presented here smoothly adapts to the left-to-right order.

DUMPS	CLOSURES	ENVIRONMENTS
$d ::= \epsilon \mid d \cdot c \diamond \pi$	$c ::= (t, e)$	$e ::= \epsilon \mid [x \leftarrow c] \cdot e$
STACKS	STATES	
$\pi ::= \epsilon \mid c \cdot \pi$	$s ::= (d, t, e, \pi)$	

Dump	Term	Env	Stack		Dump	Term	Env	Stack	
d	tu	e	π	$\to_{\mathsf{sea}}$	$d \cdot (t, e\|_t) \diamond \pi$	u	$e\|_u$	ϵ	
$d \cdot (u, e') \diamond \pi$	$\lambda x.t$	e	ϵ	$\to_{\mathsf{ret}}$	d	u	e'	$(\lambda x.t, e) \cdot \pi$	
d	$\lambda x.t$	e	$c \cdot \pi$	$\to_{\beta_{\mathsf{w}}}$	d	t	e	π	if $x \notin \mathsf{fv}(t)$
d	$\lambda x.t$	e	$c \cdot \pi$	$\to_{\beta_{\neg\mathsf{w}}}$	d	t	$[x \leftarrow c] \cdot e$	π	if $x \in \mathsf{fv}(t)$
d	x	e	π	$\to_{\mathsf{sub}}$	d	u	e'	π	if $e(x) = (u, e')$

where $e|_t$ denotes the restriction of e to the free variables of t.

FIGURE 8.3: The Space LAM.

The next step is realizing that, because of the mentioned *indifference property* of the deterministic λ-calculus $\Lambda_{\mathtt{det}}$ (containing the image of the encoding of TMs), the run of the Space LAM on a term $t \in \Lambda_{\mathtt{det}}$ is almost identical (technically, weakly bisimilar) to the one of the Space KAM on t.

Proposition 8.6.1. *The SpaceKAM and the SpaceLAM are weakly bisimilar when executed on $\Lambda_{\mathtt{det}}$-terms. Moreover, their space consumption is the same.*

Proof. The transitions of the Space KAM not dealing with applications are identical to the corresponding ones of the Space LAM (if one ignores the dump, that remains untouched). For the two transitions of the Space KAM dealing with applications, we show that, when the argument is a variable or an abstraction (as in $\Lambda_{\mathtt{det}}$), the Space LAM behaves as the Space KAM. If the active term is tx, indeed, the $\to_{\mathsf{sea_v}}$ transition of the Space KAM is simulated on the Space LAM by (with $e(x) = (\lambda y.u, e')$):

$$\begin{array}{lll} (\epsilon, tx, e, \pi) & \to_{\text{SpLAM}} & ((t, e|_t) \diamond \pi, x, e|_x, \epsilon) \\ & \to_{\text{SpLAM}} & ((t, e|_t) \diamond \pi, \lambda y.u, e', \epsilon) \\ & \to_{\text{SpLAM}} & (\epsilon, t, e|_t, (\lambda y.u, e') \cdot \pi) \\ & = & (\epsilon, t, e|_t, e(x) \cdot \pi) \end{array}$$

If the active term instead is $t(\lambda x.u)$, the $\to_{\mathsf{sea}_{\neg\mathsf{v}}}$ transition of the Space KAM is simulated on the Space LAM by:

$$\begin{array}{lll} (\epsilon, t(\lambda x.u), e, \pi) & \to_{\text{SpLAM}} & ((t, e|_t) \diamond \pi, \lambda x.u, e|_{\lambda x.u}, \epsilon) \\ & \to_{\text{SpLAM}} & (\epsilon, t, e|_t, (\lambda x.u, e|_{\lambda x.u}) \cdot \pi) \end{array}$$

In particular, these macro steps show that to evaluate TM there is no need of the dump. Now, by defining a relation $\mathcal{R}$ between states of the Space KAM and the Space LAM as

$$s_K \mathcal{R} s_L \qquad \text{iff} \qquad s_K = (t, e, \pi) \text{ and } s_L = (\epsilon, t, e, \pi)$$

the previous reasoning shows that $\mathcal{R}$ is a weak bisimulation preserving time and space complexity (modulo a constant overhead). □

Since the simulation of the Space LAM on RAMs is as smooth as for the Space KAM, we have the following result.

Theorem 8.6.2 (The Space LAM is reasonable for space)**.** *Closed CbV evaluation and the space of the Space LAM provide a reasonable space cost model for the λ-calculus.*

	Natural Time Reasonable (cost model = num. of trans. = $poly(\|\beta\|)$)	**Low-Level Time Reasonable** (actual implementation cost)	**Space Reasonable** (low-level by def.)
KAM	Yes, Thm. 2.2.2	Yes, Thm. 2.2.2	No, Thm. 2.2.2
Naive KAM	No, Prop. 8.4.4	No, Prop. 8.3.1.2	No, Prop. 8.3.1.2
Space KAM	No, Prop. 8.4.4	Yes, Thm. 8.5.1	Yes, Thm. 8.4.3
Space LAM	No, via Prop. 8.6.1 and Prop. 8.4.4	Yes, via Prop. 8.6.1 and Thm. 8.5.1	Yes, Thm 8.6.2

FIGURE 8.4: Summary of the results of this chapter.

Open and Strong Evaluation. Extending CbN/CbV evaluation to deal with open terms or even under abstractions, which is notoriously very delicate in the study of reasonable time, is instead straightforward for space. This is because these extensions play no role in the simulation of TMs, which is the delicate direction for space. Given the absence of difficulties, we refrain from introducing variants of the Space KAM/LAM for open and strong evaluation.

Call-by-Need. The only major scheme for which our technique breaks is call-by-need (CbNeed) evaluation. To our knowledge, implementations of CbNeed inevitably rely on a heap and on data pointers similar to those of the Time KAM, to realize the memoization mechanism at the heart of CbNeed. Therefore, they are space unreasonable. This is not really surprising: CbNeed, being a time optimization of CbN, trades space for time, sacrificing space reasonability.

Summary. To help the reader in the understanding of the reasonableness of the several machines of this chapter, we have reported the main results in Fig. 8.4.

Chapter 9

Closure Types Capture Space KAM Space

We refine in this section the original multi type system of de Carvalho (2018) for the KAM, so as to take into account the space consumption of the Space KAM. The intuition is quite simple. We have already noticed that KAM transitions and typing rules are in a one-to-one correspondence. We take this correspondence further, observing that each closure in the environment of a KAM state corresponds to a multiset in the type environment of the correspondent judgment. The same, although a bit more tricky, can be done for closures in the stack. This way, weighting multisets adds to the type system all the necessary information to recover the KAM space consumption. Since of course we are interested in observing a reasonable space measure, we directly consider the Space KAM, rather that than the KAM.

For simplicity, but without loss of generality, we consider the weights as the number of pointers inside a closure, and not the actual size of the closure itself. It is easy to recover the actual space consumption by just multiplying the number of pointers for their size, which is $\log |t|_0$, where t_0 is the term under evaluation. This is why we consider the following size functions, which are slightly different from those of the previous chapter. We consider that the Space KAM is implemented *without* sharing of environments and closures. The space needed to represent a state is then equal to a pointer (here considered of constant size, but the type system could be tweaked in order to separate the address spaces, as done in the previous chapter) for each piece of code in the state. This quantity is captured by the following definition[1].

$$\begin{array}{cc} \textsc{Environments} & \textsc{Stacks} \\ |\epsilon| := 0 & |\epsilon| := 0 \\ |[x \leftarrow c] \cdot e| := |c| + |e| & |c \cdot \pi| := |c| + |\pi| \end{array}$$

$$\begin{array}{cc} \textsc{Closures} & \textsc{States} \\ |(t,e)| := 1 + |e| & |(t,e,\pi)| := |e| + |\pi| \end{array}$$

Finally, the space of a Space KAM run is the maximum space over the states of the run.

Definition 9.0.1 (Run space). *Let $\rho : s_0 \to^*_{SpKAM} s$ be a Space KAM run. Then the space consumption of the run ρ is defined as follows:*

$$|\rho|_{\mathsf{sp}} := \max_{s' \in \rho} |s'|$$

9.1 Closure (Intersection) Types

Here we define our variant of multi types, dubbed closure types, that we are going to use to measure the space consumption of (typable) terms. The definition of types is standard, but for the fact that multi-sets come labeled with an index k. As in the previous chapters, the idea is that multi-sets of types are associated to arguments (according to the call-by-name translation of the λ-calculus into linear logic), and arguments give rise to closures (hence the name closure types):

[1] In the clause about states, we could have written $|(t,e,\pi)| := |e| + |\pi| + 1$, counting also the space for the closure (t,e). Since the difference is just about a constant, we can remove that "+1" without loss of generality.

the index represent the size of the closure $|\cdot|$ (*i.e.* the number of pointers to implement it) that shall be associated to that argument/multi-set.

$$\begin{array}{rrcll} \text{LINEAR TYPES} & A, B & ::= & \star \mid M^k \to A & \\ \text{CLOSURE TYPES} & M^k & ::= & [A_1, \dots, A_n]^k & k > 0, n \geq 0 \end{array}$$

Type judgments, type environments and type derivations are defined exactly as in the traditional multi type case. Please note that however some subtleties become apparent, regarding how labeled multi sets are handled. The empty multi-set is noted $[\cdot]^k$ and it also comes labeled with k. Note that k has to be strictly positive. The sum $\uplus$ of multi-sets requires the two multi-sets to have the same index, that is, we have that $M^k \uplus N^k := (M \uplus N)^k$, where on the right hand side we treat M and N as ordinary multi-sets, while $M^k \uplus N^h$ is undefined for $h \neq k$.

Space and Weakenings The study of space requires an unusual approach to weakenings. In a weak evaluation setting, a judgment $\Gamma \vdash t : A$ implies only that $dom(\Gamma) \subseteq \mathsf{fv}(t)$ and not necessarily that $dom(\Gamma) = \mathsf{fv}(t)$, because there can be untyped free variables, that is, free variables occurring under abstraction that are not evaluated and thus not typed. In the usual approach to multi types for weak evaluation, given an abstraction $\lambda x.t$ with $\mathsf{fv}(\lambda x.t) \neq \emptyset$, one can type it with $\star$, deriving a judgment $\vdash \lambda x.t : \star$ with an empty type context. Here this shall not be possible, because in the machine these variables play a role in the space usage, as they forbid to garbage collect some closures. Therefore, we modify the type system as to enforce the property $dom(\Gamma) = \mathsf{fv}(t)$, even if evaluation is weak, and this is done via an unusual use of weakenings. In particular, we distinguish between a variable $x \notin dom(\Gamma)$ and a variable $\Gamma(x) = [\cdot]^{k_x}$ that gets assigned an empty multi-set. The intuition is that $x \notin dom(\Gamma)$ would correspond to x being typed with $[\cdot]^0$, which is however not a valid type because the index k_x must be > 0. Instead, $\Gamma(x) = [\cdot]^{k_x}$ means that x is not going to be used but it shall nonetheless have an associated closure of size k_x.

We shall need a notion of type context assigning the empty multi-set (with a positive index) to all the variable in its domain.

Definition 9.1.1 (Dry type contexts)**.** *A type context Γ is* dry *if for every $x \in dom(\Gamma)$, there exists k_x such that $\Gamma(x) = [\cdot]^{k_x}$.*

Typing Rules. In order to define the typing rules we need two further notions. First, we need a notion of size of types and type contexts.

$$|\star| := 0 \qquad |M^k \to A| := k + |A| \qquad |x : M^k, \Gamma| := k + |\Gamma|$$

Note that for a type context one sums only the indices over the multi-sets, ignoring the size of linear types inside the multi-sets themselves. Second, because the multi-sets sum is restricted to those having the same index, we need a predicate over type environments to ensure that they are summable.

Definition 9.1.2. *Two type environments Γ and Δ are* summable*, noted $\Gamma\#\Delta$, if when $\Gamma(x) = M^k$ and $\Delta(x) = N^h$ then $k = h$, for all $x \in dom(\Gamma) \cap dom(\Delta)$. The notion can be naturally generalized to an arbitrary number of type environment Γ_i as $\#_i\Gamma_i$.*

The typing rules are in Fig. 9.1, for now just ignore the weights. As in the case of the multi type system and of the KAM, we have crafted this system in such a way that given a type derivation π for a term t, there is a one to one correspondence between the transitions used by the Space KAM run on t and the occurrences of the typing rules (excluded T-MANY and T-NONE) of π, plus the fact that T-$\lambda_\star$ is used to type final states. Namely, $\to_{\mathsf{sea_v}}$ corresponds to T-@$_2$, $\to_{\mathsf{sea}_{\neg\mathsf{v}}}$ to T-@$_1$, $\to_{\beta_{\neg\mathsf{w}}}$ to T-λ_1, $\to_{\beta_\mathsf{w}}$ to T-λ_2, and $\to_{\mathsf{sub}}$ to T-VAR. The intuition behind the closure types is that there is a correspondence between Space KAM data structures and the type theoretic side. The idea is that every sequence M^k corresponds to a closure c such that $|c| = k$. In particular, given a state $s = (t, e, \pi)$ and a judgment $\Gamma \vdash t : A$, the type environment $\Gamma := x_1 : M^{k_1} \dots x_n : M^{k_n}$ morally corresponds to the environment $e := [x_1 \leftarrow c_1] \dots [x_n \leftarrow c_n]$ and moreover $|c_i| = k_i$ for each $1 \leq i \leq n$. In the same way, one could define a correspondence between the stack and the type A.

$$\frac{}{x : [A]^k \overset{k+|A|}{\vdash} x : A}\ \text{T-VAR} \qquad \frac{dom(\Gamma) = \mathsf{fv}(\lambda x.t) \quad \Gamma \text{ is dry}}{\Gamma \overset{|\Gamma|}{\vdash} \lambda x.t : \star}\ \text{T-}\lambda_\star$$

$$\frac{\Gamma, x : M^k \overset{w}{\vdash} t : A}{\Gamma \overset{w}{\vdash} \lambda x.t : M^k \to A}\ \text{T-}\lambda_1 \qquad \frac{\Gamma \overset{w}{\vdash} t : A \quad x \notin dom(\Gamma)}{\Gamma \overset{\max\{w, |\Gamma|+|A|+k\}}{\vdash} \lambda x.t : [\cdot]^k \to A}\ \text{T-}\lambda_2$$

$$\frac{\Gamma_i \overset{v_i}{\vdash} t : A_i \quad 1 \le i \le n \quad \#_i \Gamma_i}{\uplus_{i=1}^n \Gamma_i \overset{\max_i\{v_i\}}{\vdash} t : [A_1..A_n]^{1+|\uplus_{i=1}^n \Gamma_i|}}\ \text{T-MANY} \qquad \frac{dom(\Gamma) = \mathsf{fv}(t) \quad \Gamma \text{ is dry}}{\Gamma \overset{0}{\vdash} t : [\cdot]^{1+|\Gamma|}}\ \text{T-NONE}$$

$$\frac{\Gamma \overset{w}{\vdash} t : M^k \to A \quad \Delta \overset{v}{\vdash} u : M^k \quad \Gamma \# \Delta}{\Gamma \uplus \Delta \overset{\max\{w,v\}}{\vdash} tu : A}\ \text{T-@}_1 \qquad \frac{\Gamma \overset{w}{\vdash} t : M^k \to A \qquad \Gamma \# x : M^k}{\Gamma \uplus x : M^k \overset{w}{\vdash} tx : A}\ \text{T-@}_2$$

FIGURE 9.1: The closure type system.

$A := M_1^{k_1} \to \cdots \to M_m^{k_m} \to \star$ corresponds to the stack $\pi := \pi_1 \dots \pi_m$ and moreover $|\pi_i| = k_i$ for each $1 \le i \le m$. Essentially, this means that we are able to read from the type derivation of a term t the sizes of all the states reached by the Space KAM evaluating t.

Rules T-MANY and T-NONE is where multi-sets are introduced on the right. In order to explain the index on the introduced multi-set, let us first consider T-NONE. The idea is that t shall be paired by the machine with an environment e as to form a closure $c = (t, e)$. By the invariant of the machine (Lemma 8.2.1), $dom(e)$ is exactly $\mathsf{fv}(t)$, that is, e contains a closure for each variable x in $\mathsf{fv}(t)$. Now, each such closure shall have size k_x, where k_x is the index given to $[\cdot]$ by the dry context Γ. Therefore, $1 + |\Gamma|$ shall correspond to $1 + |e|$, which is exactly the size of c.

For T-MANY, the reasoning is analogous. Let us explain a point about the quantity $|\uplus_{i=1}^n \Gamma_i|$ in its conclusion. An invariant shall guarantee that $dom(\Gamma) = \mathsf{fv}(t)$, whenever $\Gamma \vdash t : A$. Then in T-MANY, the type contexts Γ_i have all the same domain, and by the summable hypothesis, they all give the same index k to the (potentially different) multi-sets $\Gamma_i(x)$ for a same variable x. Thus the various $|\Gamma_j|$ all coincide (for $j \in \{1, \dots, n\}$) and also coincide with the quantity $|\uplus_{i=1}^n \Gamma_i|$.

We highlight the main differences w.r.t. the traditional multi type system.

- *Weakening.* As we have already observed, the approach to weakenings is slightly more liberal than in multi types. It is implicitly part of rules T-$\lambda_\star$, T-λ_2, and T-NONE. In rules T-$\lambda_\star$ and T-NONE, a context Γ can be injected in the conclusion, but only if it is dry, and if its domain coincides with the free variables of the typed term. The situation at the level of the Space KAM for the rule T-$\lambda_\star$ corresponds to final states such as $(\lambda x.y, [y \leftarrow t], \epsilon)$. A closure is indeed present, hence the index $k > 0$, although it will never be used, hence its type being the empty multi-set. In rule T-λ_2, which was not present in the multi type system (as it was not present the transition $\to_{\beta_w}$ in the KAM), the variable x that does not occur free in t is morally typed with $[\cdot]^0$ in the premise in the rule. In the conclusion, however, a type $[\cdot]^0 \to A$ would not be correct. In fact, the closure corresponding to $[\cdot]$, before being eliminated by rule $\to_{\beta_w}$ of the Space KAM, has size strictly greater than zero.

  Please notice that all this consideration ensure the invariant for which for any judgment $\Gamma \vdash t : A$ or $\Gamma \vdash t : M^k$ one has $dom(\Gamma) = \mathsf{fv}(t)$.

- *Explicit bang rules.* In the traditional multi types rules T-NONE and T-MANY are incorporated inside rule T-@. This was impossible to do in the closure type system because the rule T-NONE was necessarily different from the rule T-MANY, and not just the special case when there are zero premises. This is because of the way we deal with weakenings: weakening is present just in the rule T-NONE, while there is not in rule T-MANY.

Rules T-NONE and T-MANY correspond to the creation of closures. This why the index of the created multi-set is $1 + |\Gamma|$. Morally, this corresponds to the closure (t, e), where t is the typed term and e corresponds to Γ. Hence, we have $|(t, e)| = 1 + |e| = 1 + |\Gamma|$.

- *Unchaining.* The type rule T-@$_2$ is the type theoretic equivalent of the Space KAM transition rule $\to_{\mathsf{sea_v}}$. It is just a specialization of the rule T-@$_1$, from which it can be derived (using also an instance of rule T-VAR).
- *Coherence.* All the rules which sum type environments have to enforce the coherence relation $\#\Gamma_i$ between the type environments Γ_i in the premises.

The Weight System. The intuition behind the weight system is very simple. The weight of the type derivation π ending in $\vdash t : \star$ should correspond to the space consumed by the complete run ρ of the Space KAM starting from t. We know that can read the size of a Space KAM state s belonging to ρ corresponding to a type judgment $\Gamma \vdash u : A$ belonging to π: it is $|s| = |\Gamma| + |A|$. Then, the space consumed by ρ is the maximum of the sizes of all the type judgments occurring in π. This is exactly what the weighting system does.

Example 9.1.3. *We provide the closure type derivation of the term* $(\lambda x.(\lambda y.(\lambda z.x)(xy))x)\mathsf{I}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : [\star]^1 \vdash^1 x : \star}\,\text{T-VAR}}{x : [\star]^1 \vdash^4 \lambda z.x : [\cdot]^3 \to \star}\,\text{T-}\lambda_2 \quad \dfrac{}{x : [\cdot]^1, y : [\cdot]^1 \vdash^0 xy : [\cdot]^3}\,\text{T-NONE}}{x : [\star]^1, y : [\cdot]^1 \vdash^4 (\lambda z.x)(xy) : \star}\,\text{T-@}_1}{x : [\star]^1 \vdash^4 \lambda y.(\lambda z.x)(xy) : [\cdot]^1 \to \star}\,\text{T-}\lambda_1}{x : [\star]^1 \vdash^4 (\lambda y.(\lambda z.x)(xy))x : \star}\,\text{T-@}_2}{\vdash^4 \lambda x.(\lambda y.(\lambda z.x)(xy))x : [\star]^1 \to \star}\,\text{T-}\lambda_1 \qquad \dfrac{\dfrac{\dfrac{}{\vdash^0 \mathsf{I} : \star}\,\text{T-}\lambda_\star}{\vdash^0 \mathsf{I} : [\star]^1}\,\text{T-MANY}}{\vdash^4 (\lambda x.(\lambda y.(\lambda z.x)(xy))x)\mathsf{I} : \star}\,\text{T-@}_1$$

Also in this case, we can observe the precise correspondence between this type derivation and the execution of the Space KAM. Not only rules and transitions are into a one-to-one correspondence, but also stack and environment entries (with their sizes) can be seen, respectively, in types and in type environments. Of course, as a consequence, the final weight is 4, as the space consumption of the Space KAM execution.

9.2 Quantitative Soundness

This and the following sections are devoted to the proof of correctness of the weighted closure type system. The proof is carried out in a rather standard way. First, we prove soundness, that is the fact that typable terms terminate, and moreover that their evaluation respects the weight. Second, we prove completeness, that is the fact that all terminating terms are typable.

9.2.1 Preliminary Properties

The first property is qualitative, that is, it does not concern weights or indices[2]. It is simply the already mentioned fact that the domain of type contexts is exactly the set of free variables of the typed term. This is the type analog of the environment domain invariant of the Space KAM.

Lemma 9.2.1 (Type contexts domain invariant)**.**

1. *If* $\pi \triangleright \Gamma \vdash t : A$ *then* $dom(\Gamma) = \mathsf{fv}(t)$.
2. *If* $\pi \triangleright \Gamma \vdash t : M$ *then* $dom(\Gamma) = \mathsf{fv}(t)$.

Proof. By induction on π. □

[2]When a definition or a statement does not rest on weights, we do not report them, for the sake of readability.

$$\frac{\overset{w_x}{\vdash} e(x) : \Gamma(x) \qquad \forall x \in dom(e)}{\overset{\max\{w_x \mid x \in dom(e)\}}{\vdash} e : \Gamma}\ \text{T-ENV} \qquad \frac{\overset{w}{\vdash} e : \Gamma \qquad \Gamma \overset{v}{\vdash} t : M^k}{\overset{\max\{w,v\}}{\vdash} (t,e) : M^k}\ \text{T-CL}$$

$$\frac{\Gamma \overset{w}{\vdash} t : M_1^{k_1} \to \cdots \to M_n^{k_n} \to \star \quad \overset{u}{\vdash} e : \Gamma \quad \overset{v_i}{\vdash} c_i : M_i^{k_i}}{\overset{\max_i\{w,v_i,u\}}{\vdash} (t,e,c_1 \cdots c_n) : \star}\ \text{T-ST}$$

FIGURE 9.2: Typing rules for the machine components of the Space KAM.

Then, we have two quantitative properties. Before stating them, we need to introduce the definition of typed Space KAM states. The idea is very simple, and it is due to de Carvalho (2018). In order to state quantitative properties that involve abstract machines, we type the execution of the abstract machine itself. We have already mentioned that type environments correspond to the Space KAM environments, and that closure types correspond to Space KAM closures. In Fig. 9.2, the connection is made formal.

The next lemma states that the size of a closure is equal to the size of its type, for every closure, and similarly for environments.

Lemma 9.2.2 (Types capture the size of closures and environments)**.**

1. *If* $\pi \triangleright \vdash e : \Gamma$ *then* $|e| = |\Gamma|$.
2. *If* $\pi \triangleright \vdash c : M^k$ *then* $|c| = k$.

The lemma expresses one of the key properties of the type system, which has an almost *magical* feeling. At first sight, indeed, the indices on multi types are completely arbitrary, as the typing rules do not seem to impose strong constraint. Surprisingly, instead, when one builds the type derivation of a closure, then the indices are uniquely determined, and they capture *exactly* the size of the closure.

Proof. By mutual induction on e and c.

1. π has the following form:
$$\frac{\vdash e(x) : \Gamma(x)}{\vdash e : \Gamma}\ \text{T-ENV}$$
Let $\Gamma(x) = M_x^{k_x}$. By *i.h.* (point 2), $|e(x)| = k_x$. Then $|e| = \sum_{x \in dom(e)} |e(x)| =_{i.h.} \sum_{x \in dom(e)} k_x = |\Gamma|$.
2. π has the following form:
$$\frac{\vdash e : \Gamma \qquad \Gamma \vdash t : M^k}{\vdash (t,e) : M^k}\ \text{T-CL}$$
with $c = (t,e)$. By *i.h.* (point 1) applied to e, we have $|e| = |\Gamma|$. Since the typing of t comes necessarily from a rule T-NONE or T-MANY, we have $k = 1 + |\Gamma|$. Then $|c| = 1 + |e| =_{i.h.} 1 + |\Gamma| = k$.

□

The second easy property is the fact that the size of a state is given by the size of the types in the judgment for its code, which give the size of the stack and the code, plus one, to account for the pointer to the code itself.

Lemma 9.2.3. *If*

$$\frac{\Gamma \vdash t : M_1^{k_1} \to \cdots \to M_n^{k_n} \to \star \quad \vdash c_i : M_i^{k_i} \quad \vdash e : \Gamma}{\vdash (t, c_1 \cdots c_n, e) : \star}\ \text{T-ST}$$

then $|\Gamma| + \sum_{i=1}^{n} k_i = |s|$.

Proof. We have:

$$\frac{\Gamma \vdash t : M_1^{k_1} \to \cdots \to M_n^{k_n} \to \star \quad \vdash c_i : M_i^{k_i} \quad \vdash e : \Gamma}{\vdash (t, c_1 \cdots c_n, e) : \star} \text{ T-ST}$$

By definition, $|s| = |e| + |c_1 \cdots c_n| = |e| + \sum_{i=1}^{n} |c_i|$. By Lemma 9.2.2, $|e| = |\Gamma|$ and $|c_i| = k_i$ for $1 \leq i \leq n$. Then $|s| = |\Gamma| + \sum_{i=1}^{n} k_i$. □

9.2.2 The Proof of Soundness

Soundness is the fact that on typed states the Space KAM terminates. Here it is refined with our space analysis, showing that the weight of the type judgment is exactly the maximum space used by the run of the Space KAM. The proof technique is mostly standard. It is based on a subject reduction property plus a space analysis of final states. What is slightly unusual is that subject reduction is not stated as an independent property, it is instead plugged into the proof of soundness. This is needed to prove the space bound.

Space Analysis of Final States. We need an auxiliary lemma that allows us to prove that the weight system correctly measures the size of final states.

Lemma 9.2.4.

1. *If* $\pi \triangleright \overset{w}{\vdash} e : \Gamma$ *and* Γ *is dry, then* $w = 0$.
2. *If* $\pi \triangleright \overset{w}{\vdash} c : [\cdot]^k$, *then* $w = 0$.

Proof. By mutual induction on e and c.

1. π has the following form:

$$\frac{\overset{w_x}{\vdash} e(x) : \Gamma(x)}{\overset{\max\{w_x | x \in dom(e)\}}{\vdash} e : \Gamma} \text{ T-ENV}$$

   with $w = \max\{w_x | x \in dom(e)\}$. Two cases:

   (a) e is empty, *i.e.* $e = \epsilon$: then $w = 0$ and Γ is the empty type environment, for which $|\Gamma| = 0$. Therefore, we have $w = 0 = |\Gamma|$, validating the statement.

   (b) e is not empty, *i.e.* $e \neq \epsilon$: Since Γ is dry, then $\overset{w_x}{\vdash} e(x) : [\cdot]^{k_x}$ for some k_x. By *i.h.* (point 2), $w_x = 0$. Then $w = \max\{w_x | x \in dom(e)\} = 0$.

2. π has the following form:

$$\frac{\overset{v}{\vdash} e : \Gamma \qquad \Gamma \overset{u}{\vdash} t : [\cdot]^k}{\overset{\max\{v,u\}}{\vdash} (t, e) : [\cdot]^k} \text{ T-CL}$$

   with $c = (t, e)$ and $w = \max\{v, u\}$. Since the typing of t comes necessarily from a rule T-NONE, Γ is dry, $u = 0$. Then we can apply the *i.h.* (point 1) to e, obtaining $v = 0$. Thus we have $w = 0$.

□

Lemma 9.2.5 (The weight of final states is their space)**.** *Let* s *be a final state and* $\pi \triangleright \vdash^w s : \star$. *Then* $w = |s|$.

Proof. Final states have the shape $(\lambda x.t, e, \epsilon)$. Then π has the following shape:

$$\dfrac{\dfrac{dom(\Gamma) = \mathrm{fv}(\lambda x.t) \quad \Gamma \text{ is dry}}{\Gamma \overset{|\Gamma|}{\vdash} \lambda x.t : \star}\ \text{T-}\lambda_\star \qquad \overset{v}{\vdash} e : \Gamma}{\overset{\max\{v,|\Gamma|\}}{\vdash} (\lambda x.t, e, \epsilon) : \star}\ \text{T-ST}$$

with $w = \max\{v, |\Gamma|\}$. By Lemma 9.2.4.1, $v = 0$. Therefore, $w = |\Gamma|$ and so $w = |e| = |s|$. □

Soundness. Soundness shall be proved by induction on the size of type derivation, which is defined ignoring some typing rules, as follows.

Definition 9.2.6 (Type derivations size)**.** *The size $|\pi|$ of a type derivation π is its number of rules without counting rules* T-MANY, T-NONE, T-CL, *and* T-ENV.

The subject reduction argument shall need the following auxiliary lemma.

Lemma 9.2.7 (Multi-set splitting)**.**

1. Terms*: let $\pi \rhd \Gamma \overset{w}{\vdash} t : M^k \uplus N^k$ with $k = 1 + |\Gamma|$. Then there exists two type derivations $\pi_M \rhd \Gamma_M \overset{w_M}{\vdash} t : M^k$ and $\pi_N \rhd \Gamma_N \overset{w_N}{\vdash} t : N^k$ such that $\Gamma_M \# \Gamma_N$, $\Gamma = \Gamma_M \uplus \Gamma_N$, $|\pi| = |\pi_M| + |\pi_N|$ and $w = \max\{w_M, w_N\}$.*

2. Closures*: let $\pi \rhd \overset{w}{\vdash} c : M^k \uplus N^k$. Then there exists two type derivations $\pi_M \rhd \overset{w_M}{\vdash} c : M^k$ and $\pi_N \rhd \overset{w_N}{\vdash} c : N^k$ such that $|\pi| = |\pi_M| + |\pi_N|$ and $w = \max\{w_M, w_N\}$.*

3. Environments*: let Γ and Δ summable and $\pi \rhd \overset{w}{\vdash} e : \Gamma \uplus \Delta$. Then there exists two type derivations $\pi_\Gamma \rhd \overset{w_\Gamma}{\vdash} e|_{dom(\Gamma)} : \Gamma$ and $\pi_\Delta \rhd \overset{w_\Delta}{\vdash} e|_{dom(\Delta)} : \Delta$ such that $|\pi| = |\pi_\Gamma| + |\pi_\Delta|$ and $w = \max\{w_\Gamma, w_\Delta\}$.*

Proof.

1. Suppose that $M^k = [\cdot]^k$. Then $M^k \uplus N^k = N^k$. Now, let Γ_M be defined as the unique dry type context such that $dom(\Gamma_M) = dom(\Gamma)$ and it is summable with Γ—note that necessarily $|\Gamma| = |\Gamma_M|$. By Lemma 9.2.1, $dom(\Gamma) = \mathrm{fv}(t)$, and so we obtain the following derivation:

$$\dfrac{}{\Gamma_M \overset{0}{\vdash} t : [\cdot]^{1+|\Gamma_M|}}\ \text{T-NONE}$$

which is the π_M of the statement. We also take $\pi_N := \pi$, and the statement holds.

If $N^k = [\cdot]^k$ the proof is as in the previous case.

Assume now that both M^k and N^k are non-empty, say $M^k = [A_1, \ldots, A_m]^k$ and $N^k = [A_{m+1}, \ldots, A_n]^k$. Then π has the following shape:

$$\dfrac{\Gamma_i \overset{v_i}{\vdash} t : A_i \quad 1 \le i \le n \quad \#_i \Gamma_i}{\uplus_{i=1}^n \Gamma_i \overset{\max_i\{v_i\}}{\vdash} t : [A_1, .., A_n]^{1+|\uplus_{i=1}^n \Gamma_i|}}\ \text{T-MANY}$$

and the two derivations π_M and π_N are obtained as follows

$$\pi_M := \dfrac{\Gamma_i \overset{v_i}{\vdash} t : A_i \quad 1 \le i \le m \quad \#_i \Gamma_i}{\uplus_{i=1}^m \Gamma_i \overset{\max_i\{v_i\}}{\vdash} t : [A_1, .., A_m]^{1+|\uplus_{i=1}^m \Gamma_i|}}\ \text{T-MANY}$$

and

$$\pi_N := \dfrac{\Gamma_i \overset{v_i}{\vdash} t : A_i \quad m+1 \leq i \leq n \quad \#_i \Gamma_i}{\uplus_{i=m+1}^{n} \Gamma_i \overset{\max_i\{v_i\}}{\vdash} t : [A_{m+1}, .., A_n]^{1+|\uplus_{i=m+1}^{n}\Gamma_i|}}\ \text{T-MANY}$$

which clearly satisfy the statement.

2. If $c = (t, e)$ then π has the following form

$$\dfrac{\overset{w}{\vdash} e : \Gamma \qquad \Gamma \overset{v}{\vdash} t : M^k}{\overset{\max\{w,v\}}{\vdash} (t, e) : M^k}\ \text{T-CL}$$

Then the *i.h.* (point 1) applied to the right premise gives two derivations for t with respect to M^k and N^k. In particular it gives two summable type contexts Γ_M and Γ_N with which one applies the *i.h.* (point 3) to the left premise, obtaining two derivations for e with respect to Γ_M and Γ_N. Pairing the respective derivations for t and e one obtains the statement.

3. If e is empty then both Γ and Δ are empty and the statement trivially holds. Otherwise, it follows from applying the *i.h.* (point 2) for each variable in $dom(\Gamma) \cap dom(\Delta)$.

□

Theorem 9.2.8 (Soundness)**.** *Let s be a Space KAM reachable state such that there is a derivation $\pi \triangleright \vdash^w s : \star$. Then*

1. Termination*: there is a run $\rho : s \to^*_{SpKAM} s_f$ to a final state. Moreover,*
2. Space bound*: $w = |\rho|_{\mathsf{sp}}$.*

Proof. The proof is by induction on $|\pi|$ and case analysis on whether s is final. If s is final then the run ρ in the statement is given by the empty run. Therefore, we have $|\rho|_{\mathsf{sp}} = |s|$ and the required space bound becomes $w = |s|$, which is given by Lemma 9.2.5. If s is not final then $s \to_{SpKAM} s'$. By examining all the transition rules. We set $A := M_1^{k_1} \to \cdots \to M_n^{k_n} \to \star$ and $\pi := \pi_1 \cdots \pi_n$.

- Case $\to_{\mathsf{sea_v}}$, *i.e.* $s = (tx, e, \pi)$ and $s' = (t, e|_t, e(x) \cdot \pi)$. The type derivation π typing s has the following shape:

$$\dfrac{\dfrac{\Gamma \overset{w'}{\vdash} t : M \to A}{\Gamma \uplus x : M \overset{w'}{\vdash} tx : A}\ \text{T-@}_2 \qquad \pi_e \triangleright \overset{u}{\vdash} e : \Gamma \uplus x : M \quad \overset{v_i}{\vdash} \pi_i : M_i^{k_i}}{\overset{\max_i\{w', u, v_i\}}{\vdash} (tx, e, \pi) : \star}\ \text{T-ST}$$

with $w = \max_i\{w', v_i, u\}$. By Lemma 9.2.7, there are two derivations $\pi_\Gamma \triangleright \overset{u_\Gamma}{\vdash} e|_{dom(\Gamma)} : \Gamma$ and $\pi_{x:M} \triangleright \overset{u_{x:M}}{\vdash} e(x) : x : M$ such that $|\pi_e| = |\pi_\Gamma| + |\pi_{x:M}|$ and $u = \max\{u_\Gamma, u_{x:M}\}$. By the environment domain invariant (Lemma 8.2.1), $dom(\Gamma) = \mathrm{fv}(t)$. Moreover, $\pi_{x:M}$ is necessarily the conclusion of a unary T-ENV rule of premise $\pi_M \triangleright \overset{u_M}{\vdash} e(x) : M$ for which $|\pi_M| = |\pi_{x:M}|$ and $u_M = u_{x:M}$. Then, s' can be typed by the following derivation π':

$$\pi' := \dfrac{\Gamma' \overset{w'}{\vdash} t : M \to A \quad \pi_\Gamma \triangleright \overset{u_\Gamma}{\vdash} e|_t : \Gamma \quad \pi_M \triangleright \overset{u_M}{\vdash} e(x) : M \quad \overset{v_i}{\vdash} \pi_i : M_i^{k_i}}{\overset{\max_i\{w', u_\Gamma, u_M, v_i\}}{\vdash} (t, e|_t, e(x) \cdot \pi) : \star}\ \text{T-ST}$$

Since $|\pi_e| = |\pi_\Gamma| + |\pi_{x:M}| = |\pi_\Gamma| + |\pi_M|$, we have $|\pi| > |\pi'|$ (beccause the T-@$_2$ rule is removed), and so we can apply the *i.h.*, obtaining a run $\sigma : s' \to^*_{SpKAM} s_f$ to a final state such

that $\max_i\{w', u_\Gamma, u_M, v_i\} = |\sigma|_{\mathsf{sp}}$. Then there is a run $\rho : s \to^*_{SpKAM} s_f$, proving the first part of the statement (termination).

For the space bound, note that, since $u = \max\{u_\Gamma, u_{x:M}\} = \max\{u_\Gamma, u_M\}$, we have $w = \max_i\{w', u_\Gamma, u_M, v_i\}$. Additionally, $|s| \leq |s'|$, because by the environment domain invariant $e|_t$ removes at most $e(x)$ from e, which is however added to the stack. Then $|\rho|_{\mathsf{sp}} = |\sigma|_{\mathsf{sp}} = \max_i\{w', u_\Gamma, u_M, v_i\} = w$, proving the second part of the statement.

- Case $\to_{\mathsf{sea}_{\neg\mathsf{v}}}$, *i.e.* $s = (tu, e, \pi)$ and $s' = (t, e|_t, (u, e|_u) \cdot \pi)$.

  The type derivation π typing s has the following shape:

$$\dfrac{\dfrac{\Gamma \overset{w_1}{\vdash} t : M^k \to A \quad \Delta \overset{w_2}{\vdash} u : M^k \quad \Gamma\#\Delta}{\Gamma \uplus \Delta \overset{\max\{w_1,w_2\}}{\vdash} tu : A}\,\text{T-@}_1 \qquad \overset{u}{\vdash} e : \Gamma \uplus \Delta \quad \overset{v_i}{\vdash} \pi_i : M_i^{k_i}}{\overset{\max_i\{w_1,w_2,v_i,u\}}{\vdash} (tu, e, \pi) : \star}\,\text{T-ST}$$

  with $w = \max_i\{w_1, w_2, v_i, u\}$. By Lemma 9.2.7.3, there are two derivations $\pi_\Gamma \triangleright \overset{u_\Gamma}{\vdash} e|_{dom(\Gamma)} : \Gamma$ and $\pi_\Delta \triangleright \overset{u_\Delta}{\vdash} e|_{dom(\Delta)} : \Delta$ such that $|\pi_e| = |\pi_\Gamma| + |\pi_\Delta|$ and $u = \max\{u_\Gamma, u_\Delta\}$. By the environment domain invarian (Lemma 8.2.1), $dom(\Gamma) = \mathsf{fv}(t)$ and $dom(\Delta) = \mathsf{fv}(u)$. Then, s' can be typed by the following derivation π':

$$\pi' := \dfrac{\Gamma \overset{w_1}{\vdash} t : M^k \to A \quad \overset{u_\Gamma}{\vdash} e|_t : \Gamma \qquad \dfrac{\Delta \overset{w_2}{\vdash} u : M^k \qquad \overset{u_\Delta}{\vdash} e|_u : \Delta}{\overset{\max\{w_2,u_\Delta\}}{\vdash} (u, e|_u) : M^k}\,\text{T-CL} \qquad \overset{v_i}{\vdash} \pi_i : M_i^{k_i}}{\overset{\max_i\{w_1,w_2,v_i,u_\Gamma,u_\Delta\}}{\vdash} (t, e|_t, (u, e|_u) \cdot \pi) : \star}\,\text{T-ST}$$

  Since $|\pi_e| = |\pi_\Gamma| + |\pi_\Delta|$, we have $|\pi| > |\pi'|$ (beccause the T-@$_2$ rule is removed and rule T-CL does not count for the size), and so we can apply the *i.h.*, obtaining a run $\sigma : s' \to^*_{SpKAM} s_f$ to a final state such that $\max_i\{w_1, w_2, v_i, u_\Gamma, u_\Delta\} = |\sigma|_{\mathsf{sp}}$. Then there is a run $\rho : s \to^*_{SpKAM} s_f$, proving the first part of the statement (termination).

  For the space bound, note that, since $u = \max\{u_\Gamma, u_\Delta\}$, we have $\max_i\{w_1, w_2, v_i, u_\Gamma, u_\Delta\} = \max_i\{w_1, w_2, v_i, u\} = w$. Additionally, $|s| \leq |s'|$, because by the environment domain invariant all the pointer is e are in $e|_t$ or in $e|_u$. Then $|\rho|_{\mathsf{sp}} = |\sigma|_{\mathsf{sp}} = \max_i\{w', u_\Gamma, u_M, v_i\} = w$, proving the second part of the statement.

- Case $\to_{\beta_{\mathsf{w}}}$, *i.e.* $s = (\lambda x.t, e, c \cdot \pi)$ and $s' = (t, e, \pi)$. The type derivation π typing s has the following shape:

$$\dfrac{\dfrac{\Gamma \overset{w'}{\vdash} t : A \quad x \notin dom(\Gamma)}{\Gamma \overset{\max\{w',|\Gamma|+|A|+k\}}{\vdash} \lambda x.t : [\cdot]^k \to A}\,\text{T-}\lambda_2 \qquad \overset{u}{\vdash} e : \Gamma \quad \overset{v}{\vdash} c : [\cdot]^k \quad \overset{v_i}{\vdash} \pi_i : M_i^{k_i}}{\overset{\max_i\{w',|\Gamma|+|A|+k,u,v,v_i\}}{\vdash} (\lambda x.t, e, c \cdot \pi) : \star}\,\text{T-ST}$$

  with $w = \max_i\{w', |\Gamma| + |A| + k, u, v, v_i\}$. The target state s' can be typed by the following derivation π':

$$\pi' := \dfrac{\Gamma \overset{w'}{\vdash} t : A \quad x \notin dom(\Gamma) \quad \overset{u}{\vdash} e : \Gamma \quad \overset{v_i}{\vdash} \pi_i : M_i^{k_i}}{\overset{\max_i\{w',u,v_i\}}{\vdash} (t, e, \pi) : \star}\,\text{T-ST}$$

Since the T-λ_2 rule is removed, we have $|\pi| > |\pi'|$, and so we can apply the *i.h.*, obtaining a run $\sigma : s' \to^*_{SpKAM} s_f$ to a final state such that $\max_i\{w', u, v_i\} = |\sigma|_{\mathsf{sp}}$. Then there is a run $\rho : s \to^*_{SpKAM} s_f$, proving the first part of the statement (termination).

For the space bound, note that:

- $|s| = |\Gamma| + |A| + k$ by Lemma 9.2.3, giving $w = \max_i\{w', |s|, u, v, v_i\}$.
- $v = 0$ by Lemma 9.2.4.2, giving $w = \max_i\{w', |s|, u, v_i\}$.

Now, there are two cases:

- $|\rho|_{\mathsf{sp}} = |\sigma|_{\mathsf{sp}}$: that is, $|s| \leq |\sigma|_{\mathsf{sp}} = \max_i\{w', u, v_i\}$. Then $|s| \leq \max_i\{w', u, v_i\}$, and so $w = \max_i\{w', |s|, u, v_i\} = \max_i\{w', u, v_i\} =_{i.h.} |\sigma|_{\mathsf{sp}} = |\rho|_{\mathsf{sp}}$.
- $|\rho|_{\mathsf{sp}} > |\sigma|_{\mathsf{sp}}$: that is, $|s| > |\sigma|_{\mathsf{sp}} = \max_i\{w', u, v_i\}$. Then $|s| > \max_i\{w', u, v_i\}$, and so $w = \max_i\{w', |s|, u, v_i\} = |s| = |\rho|_{\mathsf{sp}}$.

- Case $\to_{\beta_{\neg \mathsf{w}}}$, *i.e.* $s = (\lambda x.t, e, c \cdot \pi)$ and $s' = (t, [x{\leftarrow}c] \cdot e, \pi)$. The type derivation π typing s has the following shape:

$$\dfrac{\dfrac{\Gamma, x : M^k \overset{w'}{\vdash} t : A}{\Gamma \overset{w'}{\vdash} \lambda x.t : M^k \to A}\ \text{T-}\lambda_1 \qquad \overset{u}{\vdash} e : \Gamma \quad \overset{v}{\vdash} c : M^k \quad \overset{v_i}{\vdash} \pi_i : M_i^{k_i}}{\overset{\max_i\{w', v, v_i, u\}}{\vdash} (\lambda x.t, e, c \cdot \pi) : \star}\ \text{T-ST}$$

with $w = \max_i\{w', v, v_i, u\}$. The target state s' can be typed by the following derivation π':

$$\pi' := \dfrac{\Gamma, x : M^k \overset{w'}{\vdash} t : A \qquad \dfrac{\overset{u}{\vdash} e : \Gamma \quad \overset{v}{\vdash} c : M^k}{\overset{\max\{u,v\}}{\vdash} [x{\leftarrow}c] \cdot e : \Gamma, x : M^k}\ \text{T-ENV} \qquad \overset{v_i}{\vdash} \pi_i : M_i^{k_i}}{\overset{\max_i\{w', v, v_i, u\}}{\vdash} (t, [x{\leftarrow}c] \cdot e, \pi) : \star}\ \text{T-ST}$$

Since the T-λ_1 rule is removed and the T-ENV rule does not count for the size of type derivations, we have $|\pi| > |\pi'|$, and so we can apply the *i.h.*, obtaining a run $\sigma : s' \to^*_{SpKAM} s_f$ to a final state such that $\max_i\{w', v, v_i, u\} = |\sigma|_{\mathsf{sp}}$. Then there is a run $\rho : s \to^*_{SpKAM} s_f$, proving the first part of the statement (termination).

For the space bound, note that

$$|s| \;=\; |e| + |c \cdot \pi| \;=\; |e| + |c| + |\pi| \;=\; |[x{\leftarrow}c] \cdot e| + |\pi| \;=\; |s'|.$$

and so the space bound follows from the *i.h.*

- Case $\to_{\mathsf{sub}}$, *i.e.* $s = (x, [x{\leftarrow}(u, e)], \pi)$ and $s' = (u, e, \pi)$. The type derivation π typing s has the following shape:

$$\dfrac{\dfrac{}{x : [A]^k \overset{k+|A|}{\vdash} x : A}\ \text{T-VAR} \qquad \dfrac{\dfrac{\Gamma \overset{w'}{\vdash} u : [A]^k \quad \overset{u}{\vdash} e : \Gamma}{\overset{\max\{u,w'\}}{\vdash} (u, e) : [A]^k}\ \text{T-CL}}{\overset{\max\{u,w'\}}{\vdash} [x{\leftarrow}(u, e)] : x : [A]^k}\ \text{T-ENV} \qquad \overset{v_i}{\vdash} \pi_i : M_i^{k_i}}{\overset{\max_i\{k+|A|, u, w', v_i\}}{\vdash} (x, [x{\leftarrow}(u, e)], \pi) : \star}\ \text{T-ST}$$

with $w = \max_i\{k + |A|, u, w', v_i\}$. The target state s' can be typed by the following derivation π':

$$\pi' := \dfrac{\Gamma \overset{w'}{\vdash} u : A \quad \overset{u}{\vdash} e : \Gamma \quad \overset{v_i}{\vdash} \pi_i : M_i^{k_i}}{\overset{\max_i\{w',u,v_i\}}{\vdash} (u, e, \pi) : \star}\ \text{T-ST}$$

Since the T-VAR rule is removed, we have $|\pi| > |\pi'|$, and so we can apply the *i.h.*, obtaining a run $\sigma : s' \to^*_{SpKAM} s_f$ to a final state such that $1 + \max_i\{w', u, v_i\} = |\sigma|_{\mathsf{sp}}$. Then there is a run $\rho : s \to^*_{SpKAM} s_f$, proving the first part of the statement (termination).

For the space bound, note that $|s| = k + |A|$ by Lemma 9.2.3, giving $w = \max_i\{|s|, u, w', v_i\}$. Now, there are two cases:

- $|\rho|_{\mathsf{sp}} = |\sigma|_{\mathsf{sp}}$: that is, $|s| \leq |\sigma|_{\mathsf{sp}} = \max_i\{w', u, v_i\}$. Then $|s| \leq \max_i\{w', u, v_i\}$, and so $w = \max_i\{|s|, u, w', v_i\} = \max_i\{w', u, v_i\} =_{i.h.} |\sigma|_{\mathsf{sp}} = |\rho|_{\mathsf{sp}}$.
- $|\rho|_{\mathsf{sp}} > |\sigma|_{\mathsf{sp}}$: that is, $|s| > |\sigma|_{\mathsf{sp}} = \max_i\{w', u, v_i\}$. Then $|s| > \max_i\{w', u, v_i\}$, and so $w = \max_i\{|s|, u, w', v_i\} = |s| = |\rho|_{\mathsf{sp}}$.

□

As a corollary, we can transfer back the soundness result from typed *states* to typed *terms*.

Corollary 9.2.9 (Soundness, on Terms)**.** *Let t be a closed λ-term. If there exists $\pi \triangleright \vdash^w t : \star$, then there exists a complete Space KAM run ρ from t such that $|\rho|_{\mathsf{sp}} = w$.*

9.3 Completeness

Completeness is the fact that all states on which the Space KAM terminates are typable. Here the proof technique is standard: we show that final states are typable, that a subject expansion property holds, and then we infer completeness. We do not perform any space analysis, since it is already it follows from the soundness part, once we know that a state is typable.

Final States are Typable. We need an auxiliary lemma about closures and environments.

Lemma 9.3.1 (Closures and environments are typable)**.**

1. *There exists a dry type context Γ and a derivation $\pi \triangleright \vdash e : \Gamma$ for every environment e.*
2. *There exist k and a derivation $\pi \triangleright \vdash c : [\cdot]^k$ for every closure c.*

Proof. By mutual induction on e and c.

1. If e is empty then Γ is the empty type context. Otherwise, by *i.h.* (point 2), for every closure $e(x)$ with $x \in dom(e)$ there exists k_x and a derivation $\pi \triangleright \vdash c : [\cdot]^k$. Then π is defined as follows:
$$\dfrac{\vdash e(x) : \Gamma(x)}{\vdash e : \Gamma}\ \text{T-ENV}$$
2. Let $c = (t, e)$. By the environments domain invariant (Lemma 8.2.1), $dom(e) = \mathsf{fv}(t)$. By *i.h.* (point 1), there exists a dry type context Γ and a derivation $\pi \triangleright \vdash e : \Gamma$. Since $dom(e) = \mathsf{fv}(t)$, we can apply rule T-NONE deriving a judgment $\Gamma \vdash t : [\cdot]^{1+|\Gamma|}$. Then π is defined as follows:
$$\dfrac{\vdash e : \Gamma \qquad \Gamma \vdash t : [\cdot]^{1+|\Gamma|}}{\vdash (t, e) : [\cdot]^{1+|\Gamma|}}\ \text{T-CL}$$

□

Lemma 9.3.2 (Final states are typable). *Let s be a final state. Then there exists a type derivation $\pi \triangleright \vdash s : \star$.*

Proof. Final states have the shape $(\lambda x.t, e, \epsilon)$. By Lemma 9.3.1.1, there is a type derivation $\vdash e : \Gamma$ with Γ dry. By the environments domain invariant (Lemma 8.2.1), $dom(e) = \mathsf{fv}(\lambda x.t)$. Then π is defined as follows:

$$\dfrac{\dfrac{dom(\Gamma) = \mathsf{fv}(\lambda x.t) \quad \Gamma \text{ is dry}}{\Gamma \vdash \lambda x.t : \star}\ \text{T-}\lambda_\star \qquad \vdash e : \Gamma}{\vdash (\lambda x.t, e, \epsilon) : \star}\ \text{T-ST}$$

□

Subject Expansion. The proof of subject expansion is standard. There is only one delicate point, in expanding the application transitions $\to_{\mathsf{sea}_{\neg \mathsf{v}}}$ and $\to_{\mathsf{sea}_{\mathsf{v}}}$, where one needs to ensure that the two type contexts for the premises of the application are summable.

Proposition 9.3.3 (Subject expansion). *If $s \to_{SpKAM} s'$ and there exists $\pi' \triangleright \vdash s' : \star$, then there exists $\pi \triangleright \vdash s : \star$.*

Proof. There are three cases that require something more than simply reading backwards the subject reduction argument in the proof of the correctness theorem. One is $\to_{\beta_{\mathsf{w}}}$, where in addition one needs to type the garbage collected closure, but this is ensured by Lemma 9.3.1.1. The other two are the application transitions $\to_{\mathsf{sea}_{\neg \mathsf{v}}}$ and $\to_{\mathsf{sea}_{\mathsf{v}}}$. We treat $\to_{\mathsf{sea}_{\neg \mathsf{v}}}$, the case of $\to_{\mathsf{sea}_{\mathsf{v}}}$ is analogous. We have $s = (tu, e, c_1 \cdots c_n) \to_{\mathsf{sea}_{\neg \mathsf{v}}} (t, e|_t, (u, e|_u) \cdot c_1 \cdots c_n) = s'$. The type derivation π' typing s' has the following shape:

$$\pi' = \dfrac{\Gamma \vdash t : M^k \to A \quad \vdash e|_t : \Gamma \qquad \dfrac{\Delta \vdash u : M^k \qquad \vdash e|_u : \Delta}{\vdash (u, e|_u) : M^k}\ \text{T-CL} \qquad \vdash c_i : M_i^{k_i}}{\vdash (t, e|_t, (u, e|_u) \cdot c_1 \cdots c_n) : \star}\ \text{T-ST}$$

Assuming that $\Gamma \# \Delta$, that we shall prove below, the type derivation π for s is given by:

$$\pi := \dfrac{\dfrac{\Gamma \vdash t : M^k \to A \quad \Delta \vdash u : M^k \quad \Gamma \# \Delta}{\Gamma \uplus \Delta \vdash tu : A}\ \text{T-@}_1 \qquad \vdash e : \Gamma \uplus \Delta \quad \vdash c_i : M_i^{k_i}}{\vdash (tu, e, c_1 \cdots c_n) : \star}\ \text{T-ST}$$

where the derivation for $\vdash e : \Gamma \uplus \Delta$ is obtained by an omitted and straightforward *multi-sets merging* lemma dual to the multi-sets splitting lemma (Lemma 9.2.7) used for correctness.

We now prove $\Gamma \# \Delta$, which is the additional bit not present in the proof of subject reduction. Let $x \in dom(\Gamma) \cap dom(\Delta)$. Then $e_t(x) = e_u(x) = e(x) = c$ for some closure c. By Lemma 9.2.2.2, we have both $\Gamma(x) = N_1^{|c|}$ and $\Delta(x) = N_2^{|c|}$ for some multi types N_1 and N_2, that is, $\Gamma \# \Delta$.

□

Theorem 9.3.4 (Completeness). *Let s a reachable state and $\rho : s \to_{SpKAM}^* s_f$ be a KAM run to a final state s_f. Then there exist $\pi \triangleright \vdash s : \star$.*

Proof. By induction on the length $|\rho|$ of ρ. If $|\rho| = 0$ then $s = s_f$, and the existence of a typing derivation for s is given by Lemma 9.3.2. If $|\rho| > 0$ then ρ is given by a transition $s \to_{SpKAM} s'$ followed by an execution $\sigma : s' \to_{SpKAM}^* s_f$. By *i.h.*, we obtain $\pi \triangleright \vdash s' : \star$, and by subject expansion (Prop. 9.3.3) we obtain $\pi \triangleright \vdash s : \star$. □

Again, as a corollary, we can transfer the completeness property from *states* to *terms*.

Corollary 9.3.5 (Completeness, on Terms). *If t is a closed λ-term such that there exists a complete Space KAM run from t, then there exists $\pi \triangleright \vdash t : \star$.*

Putting together quantitative soundness and completeness gives the quantitative correctness of the closure type system.

Theorem 9.3.6 (Correctness)**.** *Let t be a closed λ-term. Then there exists a complete Space KAM run ρ from t such that $|\rho|_{\mathsf{sp}} = w$ if and only if there exists $\pi \triangleright \vdash^{w} t : \star$.*

Chapter 10

Conclusions

This dissertation has brought new contributions in several research areas: the complexity of Girard's geometry of interaction, its relationship with non-idempotent intersection types, and the understanding of reasonable space cost models for the λ-calculus. In the following sections we draw some conclusions, in particular putting our results in context, and proposing some directions for future investigations.

10.1 Our Results in Perspective

This dissertation is set in the wider effort that many researchers have made to understand the (space) complexity of higher-order computation. We have made the state-of-the-art advance in a significant way, as we are going to detail.

Geometry of Interaction. Since the birth of linear logic in the late '80s, a new notion of proofs (or programs) as interaction has come to the scene. Concurrent languages such as the π-calculus, game semantics, and the geometry of interaction were some of the concrete instances of this new approach. All these tools were very successful, and allowed to solve many open problems, *e.g.* the long quest for a fully abstract model of PCF. Some questions, however, remained open until now. The space complexity of the GoI is one of these. While in the community it was conjectured that the way the GoI uses to compute could be space efficient, the available results were limited only to some particular cases. Our study instead reveals that in the general case of the untyped λ-calculus, the GoI *cannot* be space efficient. Being based on the solid tool of non-idempotent intersection types, our proof gives also a high level motivation of this fact: the space (and time) inefficiency comes from the way in which *recursion* is handled. This is why we believe that there is no easy way out, and we thus consider the problem settled, although in a negative way.

Reasonable Space Cost Models. After having found out that the GoI could not be the right answer, we have had *to take one step backward to take two steps forward*. We had investigated the GoI in order to understand how to measure the space complexity of the λ-calculus in a reasonable way. Typical execution mechanisms, such as environment abstract machines, did not seem to fit the task. However, a closer look to Krivine's abstract machine led us to the counterintuitive fact that its space consumption could be used as a reasonable space measure. The point was that all the sharing mechanisms used for time efficiency should have been turned off. Actually, in order to prove the result, many other tweaks have to be considered. Very few degrees of freedom were possible in the design of the optimized machine. This consideration makes us think that it is very unlikely that *better* results could be achieved. This is somehow disappointing, since originally there was the idea that reasonable cost models for the λ-calculus should be actually defined on the λ-calculus, in the abstract, and not on machines. Although we do not have a formal statement, we believe that this is simply impossible for space, if one wants to account for sub-linear complexity. This is because of the very nature of the λ-calculus, in which there is no distinction between program and data. This distinction is indeed fundamental when one wants to measure sub-linear space complexity.

10.2 Future Directions

It is certainly very difficult to the answer to the question "*what is to be done?*", *i.e.* to sketch some future directions that stem from our work. This is mainly due to the fact that we have mostly closed paths, rather than opened new directions.

Game Semantics. One of our contributions has been to develop a new formulation for the GoI. Not only we have defined it directly on the λ-calculus, but we have also recovered it via non-idempotent intersection types, in a very natural way. It is our opinion that game semantics (GS), very related to GoI, needs an equivalent *restyling*. GS is generally presented in a very categorical way on typed languages, focusing on full abstraction results. We believe that it could be presented in a more operational way (a proposal can be found in (Jaber, 2015)), in an untyped setting. Non-idempotent intersection types could help also in this respect, see the work by Tsukada and Ong (2016). In particular, it is well-known that Abramsky *et al.* GS is somehow isomorphic to the IAM. This is why we believe that the connection should be made clearer, also at the syntactic level. All the computational mechanisms presented in this dissertation, multi types, abstract machines, the LSC, indeed, compute in the very same way. GS is just another example of this behavior, but the right formulation has not yet been found.

Call-by-Value. Many of our results hold only in the call-by-name case. It is an interesting research direction to look at what happens in call-by-value. The GoI, in particular, is CbN almost by definition, although some effort has been done to port it to CbV (see *e.g.* (Dal Lago et al., 2015) and (Fernández and Mackie, 2002)). Would our complexity results still hold in this new scenario? Moreover, it would be interesting to adapt the type system of Chapter 9 to CbV evaluation, in order to capture the (reasonable) space measure for CbV defined in Chapter 8.

Formalization. All our results have been proved with pen and paper. Although in the very simple scenario on the pure untyped λ-calculus, this already requires very long and error-prone proofs. It is probably the right moment to develop some tools that could allow researchers to deal with abstract machines and multi types in a semi automatic way. Developing such libraries for proof assistants seems to be an interesting and very useful task. This would allow the community to focus less on writing very verbose proofs, and more on developing new ideas.

Concrete Languages and Automation. Typing programs with intersection types is very difficult. This is mainly due to the fact that it is impossible to type functions in a uniform way. Transferring our results to more concrete programming languages seems an interesting research direction. The natural starting point would be considering the language `PCF`, and a type system based on *linear dependent types* (Dal Lago and Gaboardi, 2011). This is essentially a multi type system crafted to capture the complexity of functions, rather than single λ-terms. At that point, it could be possible to devise automatic tools that could infer the type and hence the complexity of programs. Of course, the methodology would be sound but not complete, because of the undecidability of the underlying problem.

Appendices

Appendix A

Proofs of Exhaustible Invariants

A.1 Proof of the Exhaustible Invariant

Proposition A.1.1 (Exhaustible invariant)**.** *Let s be a λIAM reachable state. Then s is exhaustible.*

Proof. Let $s = (\underline{t}, \langle\cdot\rangle, \epsilon, \epsilon) \to^k_{\lambda\text{IAM}} s'$. By induction on k. For $k = 0$ there is nothing to prove because the state has no tape nor log tests. Then suppose $s \to^{k-1}_{\lambda\text{IAM}} s'' \to_{\lambda\text{IAM}} s'$. By *i.h.*, $s'' = (u, C, L, T, d)$ is exhaustible, and with this hypothesis we need to conclude that s' is exhaustible, too. There are many cases to take into account, depending on the transition used to move from s'' to s'. We recall that we use $|T|_l$ for the number of logged position in T, called *position length of T* in the proof.

First, suppose that $d = \downarrow$. Cases of $s'' \to_{\lambda\text{IAM}} s'$:

1. *Application, i.e.* $u = rw$ and

$$(\underline{rw}, C, L, T) \to_{\bullet 1} (\underline{r}, C\langle\langle\cdot\rangle w\rangle, L, \bullet\cdot T) = s'.$$

   We have to show that the obtained state s' is exhaustible. For log tests, it follows from Lemma 3.5.10.3 and the *i.h.*: s'' is a head translation of s', and the lemma states that they have the same log tests, which are exhaustible because s'' is exhaustible by *i.h.*

   For tape tests, consider a decomposition $T = T'\cdot l\cdot T''$. Two cases, depending on the parity of $|T'|_l$:

   (a) $|T'|_l$ *is odd*. Then the position length of the tape $\bullet\cdot T'\cdot l$ is even (occurrences of $\bullet$ are ignored) and so the direction of the corresponding tape test s'_l is $\uparrow$. Note that s'_l reduces to a tape test s''_l for s'' having the same focus l of s'_l:

$$s'_l = (r, \underline{C\langle\langle\cdot\rangle w\rangle}, L, \bullet\cdot T'\cdot l) \to_{\bullet 3} (rw, \underline{C}, L, T'\cdot l) = s''_l$$

   By *i.h.*, s'' is exhaustible, and so s''_l evolves to an exhaustible state surrounding l, call it q_l. Then s'_l evolves to q_l and the test is successful.

   (b) $|T'|_l$ *is even*. Then $|\bullet\cdot T'\cdot l|_l$ is odd, and the direction of the corresponding tape test s'_l os s' is $\downarrow$. Note that the corresponding tape test s''_l of s'' reduces to s'_l:

$$s''_l = (\underline{rw}, C, L, T'\cdot l) \to_{\bullet 3} (\underline{r}, C\langle\langle\cdot\rangle w\rangle, L, \bullet\cdot T'\cdot l) = s'_l$$

   By *i.h.*, s'' is exhaustible, then s''_l evolves to an exhaustible state surrounding l, call it q_l. The IAM is deterministic, so s'_l itself reduces to q_l.

2. *Abstraction 1, i.e.* $u = \lambda x.r$ and $T = \bullet\cdot T'$. Identical to the previous one.

3. *Variable bound by an abstraction, i.e.* $u = x$ and

$$\begin{array}{ll} s'' = & (\underline{x}, C\langle\lambda x.D_n\rangle, L_n\cdot L, T) \\ \to_{\mathsf{var}} & (\lambda x.D_n\langle x\rangle, \underline{C}, L, (x, \lambda x.D_n, L_n)\cdot T) = s' \end{array}$$

   The proof that s' is exhaustible is divided in two parts:

(a) *Log testing*. By Lemma 3.5.10.4, all log tests of s' are also log tests of s''. Since the latter is exhausible by *i.h.*, then all the log tests of s' are successful.

(b) *Tape testing*. We need to consider various cases, corresponding to the various decompositions of the tape $l'\cdot T$ where $l' = (x, \lambda x.D_n, L_n)$:

i. *The logged position to test is* $l = l'$, *i.e.* the first one. We are then considering a prefix of odd length of $l'\cdot T$, so the direction of the corresponding tape test s'_l is $\downarrow$. Observe, however, that by definition

$$\begin{array}{ll} s'_l = & (\underline{\lambda x.D_n\langle x\rangle}, C, L, (x, \lambda x.D_n, L_n)) \\ \to_{\mathsf{bt2}} & (x, \underline{C\langle \lambda x.D_n\rangle}, L_n\cdot L, \epsilon) = s''^{\perp} \end{array}$$

where $s''^{\perp}$ is trivially surrounding l. Moreover, by *i.h.*, s'' is exhaustible, a property which is easily transferred to $s''^{\perp}$: the log tests are the same by Lemma 3.5.10.1, while $s''^{\perp}$ satisfies tape testing trivially, because the tape is empty.

ii. *The prefix* $T'\cdot l$ *of the tape has even length* and the direction of the corresponding tape test s'_l is $\uparrow$. Let $T' = (x, \lambda x.D_n, L_n)\cdot T''$. Note that the corresponding tape test s''_l of s'' reduces to s'_l:

$$\begin{array}{ll} s''_l = & (\underline{x}, C\langle \lambda x.D_n\rangle, L_n \cdot L, T''\cdot l) \\ \to_{\mathsf{var}} & (\lambda x.D_n\langle x\rangle, \underline{C}, L, (x, \lambda x.D_n, L_n)\cdot T''\cdot l) = s'_l \end{array}$$

By *i.h.*, s'' is exhaustible, then s''_l evolves to an exhaustible state surrounding l, call it q_l. The IAM is deterministic, so s'_l itself reduces to q_l, and the test is successful.

iii. *The prefix* $T'\cdot l$ *of the tape has odd strictly positive length* and the direction of the corresponding log test s'_l is $\downarrow$. Let $T' = (x, \lambda x.D_n, L_n)\cdot T''$. Note that s'_l reduces to the corresponding log test s''_l of s'':

$$\begin{array}{ll} s'_l = & (\underline{\lambda x.D_n\langle x\rangle}, C, L, (x, \lambda x.D_n, L_n)\cdot T''\cdot l) \\ \to_{\mathsf{bt2}} & (x, \underline{C\langle \lambda x.D_n\rangle}, L_n \cdot L, T''\cdot l) = s''_l \end{array}$$

We can then proceed as usual using the *i.h.*

4. *Abstraction 2, i.e.* $u = \lambda x.r$ and

$$\begin{array}{ll} s'' = & (\underline{\lambda x.r}, C, L, (x, \lambda x.D_n, L')\cdot T) \\ \to_{\mathsf{bt2}} & (x, \underline{C\langle \lambda x.D_n\rangle}, L'\cdot L, T) = s' \end{array}$$

(a) *Log testing*. Let $l = (x, C\langle \lambda x.D_n\rangle, L')$ and note that the tape tests of s'' of focus l does the following transition:

$$\begin{array}{ll} s''_l = & (\underline{\lambda x.r}, C, L, (x, \lambda x.D_n, L')) \\ \to_{\mathsf{bt2}} & (x, \underline{C\langle \lambda x.D_n\rangle}, L'\cdot L, \epsilon) = s'_\epsilon \end{array}$$

Now, s'_ϵ surrounds l and thus, by *i.h.* (tape testing of s''), s'_ϵ is exhaustible. By Lemma 3.5.10.2, s'_ϵ and s' have the same log tests, so log testing for s' holds because it does for s'_ϵ.

(b) *Tape testing*. As usual, we have to consider various cases, corresponding to the possible decompositions $T = T'\cdot l\cdot T''$ of the tape.

i. $|T'|_l$ *is odd*, so that the prefix $T'\cdot l$ of the tape has even length and the direction of the tape test s'_l corresponding to l is $\uparrow$. Note that the tape test s''_l of s'' reduces to the corresponding tape test s'_l of s':

$$\begin{array}{ll} s''_l = & (\underline{\lambda x.D_n\langle x\rangle}, C, L, (x, \lambda x.D_n, L')\cdot T'\cdot l) \\ \to_{\mathsf{bt2}} & (x, \underline{C\langle \lambda x.D_n\rangle}, L' \cdot L, T'\cdot l) = s'_l \end{array}$$

We can then proceed as usual, exploiting the determinism of the λIAM and the *i.h.*

ii. $|T'|_l \neq 0$ *is even*, so that the prefix $T'\cdot l$ of the tape has odd length and the direction of the tape test s'_l corresponding to l is $\downarrow$. Note that s'_l reduces to the corresponding tape test s''_l of s'':

$$\begin{array}{ll} s'_l = & (\underline{x}, C\langle \lambda x.D_n\rangle, L' \cdot L, T'\cdot l) \\ \to_{\mathsf{var}} & (\lambda x.D_n\langle x\rangle, \underline{C}, L, (x, \lambda x.D_n, L')\cdot T'\cdot l) = s''_l \end{array}$$

Again, we can then proceed as usual using the *i.h.*

5. *Explicit Substitution*, *i.e.* $u = r[x \leftarrow w]$ and

$$s'' = (\underline{r[x \leftarrow w]}, C, L, T) \to_{\mathsf{es}} (\underline{r}, C\langle\langle\cdot\rangle[x \leftarrow w]\rangle, L, T) = s'$$

For log testing, it follows from Lemma 3.5.10.3 and the *i.h.*: s'' is a head translation of s', and the lemma states that they have the same log tests, which are exhaustible because s'' is exhaustible by *i.h.*

For tape testing it goes exactly as the application case. We spell it out anyway. Consider a decomposition $T = T' \cdot l \cdot T''$. Two cases, depending on the parity of $|T'|_l$:

(a) $|T'|_l$ *is odd*. Then the position length of the tape $T' \cdot l$ is even and so the direction of the corresponding tape test s'_l is $\uparrow$. Note that s'_l reduces to a tape test s''_l for s'':

$$\begin{array}{ll} s'_l = & (r, \underline{C\langle\langle\cdot\rangle[x \leftarrow w]\rangle}, L, T' \cdot l) \\ \to_{\mathsf{es2}} & (r[x \leftarrow w], \underline{C}, L, T' \cdot l) = s''_l \end{array}$$

Again, we then proceed as usual using the *i.h.*

(b) $|T'|_l$ *is even*. Then $|T' \cdot l|_l$ is odd, and the direction of the corresponding tape test s'_l os s' is $\downarrow$. Note that the corresponding tape test s''_l of s'' reduces to s'_l:

$$\begin{array}{ll} s''_l = & (\underline{r[x \leftarrow w]}, C, L, T' \cdot l) \\ \to_{\mathsf{es}} & (\underline{r}, C\langle\langle\cdot\rangle[x \leftarrow w]\rangle, L, T' \cdot l) = s'_l \end{array}$$

Again, we then proceed as usual, exploiting the determinism of the λIAM and the *i.h.*

6. *Variable bound by an explicit substitution*, *i.e.* $u = x$ and

$$\begin{array}{ll} s'' = & (\underline{x}, C\langle D_n[x \leftarrow r]\rangle, L_n \cdot L, T) \\ \to_{\mathsf{var2}} & (\underline{r}, C\langle D_n\langle x\rangle[x \leftarrow \langle\cdot\rangle]\rangle, (x, D_n[x \leftarrow r], L_n) \cdot L, T) = s' \end{array}$$

(a) *Log testing*: let $l := (x, D_n[x \leftarrow r], L_n)$ and $m = |l \cdot L|$. The m-log test of s' is

$$s_l = (r, \underline{C\langle D_n\langle x\rangle[x \leftarrow \langle\cdot\rangle]\rangle}, (x, D_n[x \leftarrow r], L_n) \cdot L, \epsilon)$$

which makes a transition

$$\to_{\mathsf{var3}} (x, \underline{C\langle D_n[x \leftarrow r]\rangle}, L_n \cdot L, \epsilon) = (s''_\epsilon)^\perp$$

that is a state surrounding l, as required by log testing. We have to prove that $(s''_\epsilon)^\perp$ is exhaustible. Tape testing is trivial, because the tape is empty. Log testing follows from the *i.h.* and the fact that $(s''_\epsilon)^\perp$ is s'' with reversed direction and without the tape, and so by Lemma 3.5.10.1 and Lemma 3.5.10.2 they have the same log tests.

Note that the i-log tests of s' for $i < m$ are the i-log tests of s' (Lemma 3.5.10.4), and so they satisfy the log testing clause by the *i.h.*

(b) *Tape testing*: it goes exactly as in the previous ordinary cases. We spell it out anyway. Consider a decomposition $T = T' \cdot l \cdot T''$. Two cases, depending on the parity of $|T'|_l$:

i. $|T'|_l$ *is odd*. Then the position length of the tape $T' \cdot l$ is even and so the direction of the corresponding tape test s'_l is $\uparrow$. Note that s'_l reduces to a tape test s''_l for s'':

$$\begin{array}{ll} s'_l = & (r, \underline{C\langle D_n\langle x\rangle[x \leftarrow \langle\cdot\rangle]\rangle}, (x, D_n[x \leftarrow r], L_n) \cdot L, T' \cdot l) \\ \to_{\mathsf{var3}} & (x, \underline{C\langle D_n[x \leftarrow r]\rangle}, L_n \cdot L, T' \cdot l) = s''_l \end{array}$$

Again, we then proceed as usual using the *i.h.*

ii. $|T'|_l$ *is even*. Then $|T' \cdot l|_l$ is odd, and the direction of the corresponding tape test s'_l os s' is $\downarrow$. Note that the corresponding tape test s''_l of s'' reduces to s'_l:

$$\begin{array}{ll} s''_l = & (\underline{x}, C\langle D_n[x \leftarrow r]\rangle, L_n \cdot L, T' \cdot l) \\ \to_{\mathsf{var2}} & (\underline{r}, C\langle D_n\langle x\rangle[x \leftarrow \langle\cdot\rangle]\rangle, (x, D_n[x \leftarrow r], L_n) \cdot L, T' \cdot l) = s'_l \end{array}$$

Again, we then proceed as usual, exploiting the determinism of the λIAM and the *i.h.*

Now, suppose that $d = \uparrow$. Cases of $s'' \to_{\lambda\text{IAM}} s'$:

1. *Coming from the left of an application, i.e.* $C = D\langle\langle\cdot\rangle r\rangle$ and

$$s'' = (u, \underline{D\langle\langle\cdot\rangle r\rangle}, L, l \cdot T) \to_{\mathsf{arg}} (\underline{r}, D\langle u\langle\cdot\rangle\rangle, l \cdot L, T) = s'.$$

   The proof that s' is exhaustible is divided in two parts:

   (a) *Log testing*. The log tests of s' are those of s'' plus $(r, \underline{D\langle u\langle\cdot\rangle\rangle}, l \cdot L, \epsilon)$. The former are fine because of the *i.h.*, while about the latter, observe that $(r, \underline{D\langle u\langle\cdot\rangle\rangle}, l \cdot L, \epsilon)$ evolves to $(\underline{u}, D\langle\langle\cdot\rangle r\rangle, L, l)$ which is a tape test of s''. The thesis easily follows by *i.h.*

   (b) *Tape testing*. Let T' be a prefix of T such that $T' = T''\cdot l'$. Two cases:

      i. $|T'|_l$ *is odd*, and the direction is $\downarrow$. Note that the tape test s''_l of s'' corresponding to l reduces to a tape test s'_l of s':

$$\begin{array}{rl} s''_l = & (r, \underline{D\langle u\langle\cdot\rangle\rangle}, L, l\cdot T') \\ \to_{\mathsf{arg}} & (\underline{r}, D\langle u\langle\cdot\rangle\rangle, l \cdot L, T') = s'_l \end{array}$$

      We can then proceed as usual, using the *i.h.* and determinism of the IAM.

      ii. $|T'|_l$ *is even*, and the direction is $\uparrow$. Note that s'_l reduces to the corresponding tape test s''_l of s'':

$$\begin{array}{rl} s'_l = & (r, \underline{D\langle u\langle\cdot\rangle\rangle}, l \cdot L, T') \\ \to_{\mathsf{bt1}} & (\underline{r}, D\langle u\langle\cdot\rangle\rangle, L, l\cdot T') = s''_l \end{array}$$

      Again, we can proceed as usual, using the *i.h.*

2. *Coming from the right of an application, i.e.* $C = D\langle r\langle\cdot\rangle\rangle$ and

$$s'' = (u, \underline{D\langle r\langle\cdot\rangle\rangle}, l \cdot L, T) \to_{\mathsf{bt1}} (\underline{r}, D\langle\langle\cdot\rangle u\rangle, L, l \cdot T) = s'.$$

   The proof that s' is exhaustible is divided in two parts:

   (a) *Log testing*: the log tests of s' are among the log tests of s'', so log testing follows from *i.h.*

   (b) *Tape testing*. Let T' be a prefix of T. Two cases:

      i. $T' = T$ *is empty*. So that the tape contains only l, its length is odd, and the direction is $\downarrow$. The state to be proven exhaustible is

$$s'_l = (\underline{r}, D\langle\langle\cdot\rangle u\rangle, L, l)$$

      Now, note that the log test $s''_{|l\cdot L|}$ of s'' reduces in one step to s'_l:

$$\begin{array}{rl} s''_{|l\cdot L|} = & (u, \underline{D\langle r\langle\cdot\rangle\rangle}, l \cdot L, \epsilon) \\ \to_{\mathsf{bt1}} & (\underline{r}, D\langle\langle\cdot\rangle u\rangle, L, l) \end{array}$$

      By log testing for s'', there is a state q_l surrounding l such that $s''_{|l\cdot L|} \to^*_{\lambda\text{IAM}} q_l$. By determinism of the IAM, $s'_l \to^*_{\lambda\text{IAM}} q_l$.

      ii. $T' \neq T$ *is non-empty*. Then $T' = T''\cdot l'$ Two cases:

         A. $|T''\cdot l'|_l$ *is even*, so that the tape $l\cdot T''\cdot l'$ has odd length and the direction is $\downarrow$. Note that the tape test $s''_{l'}$ corresponding to l' of s'' reduces to the tape test $s'_{l'}$ corresponding to l' of s':

$$\begin{array}{rl} s''_{l'} = & (r, \underline{D\langle\langle\cdot\rangle u\rangle}, l\cdot L, T''\cdot l') \\ \to_{\mathsf{bt1}} & (\underline{r}, D\langle\langle\cdot\rangle u\rangle, L, l\cdot T''\cdot l') = s'_{l'} \end{array}$$

         In this case, as usual, we can conclude by determinism of the λIAM.

         B. $|T'|_l$ *is odd*, so that the tape $l\cdot T''\cdot l'$ has even length and the direction is $\uparrow$. Note that $s'_{l'}$ reduces to the corresponding tape test $s''_{l'}$ of s'':

$$\begin{array}{rl} s'_{l'} = & (r, \underline{D\langle\langle\cdot\rangle u\rangle}, L, l\cdot T''\cdot l') \\ \to_{\mathsf{arg}} & (\underline{r}, D\langle\langle\cdot\rangle u\rangle, l\cdot L, T''\cdot l') = s''_{l'} \end{array}$$

Again, the usual scheme allows us to conclude that tape testing holds.

3. *Explicit Substitution*

$$s'' = (u, \underline{C\langle\langle\cdot\rangle[x\leftarrow r]\rangle}, L, T) \to_{\mathsf{es2}} (u[x\leftarrow r], \underline{C}, L, T) = s'$$

   (a) *Log testing*: by Lemma 3.5.10.3 (head translation), the log tests of s' are log tests of s'', which satisfy log testing by the *i.h.*

   (b) *Tape testing*: it goes exactly as for the other ordinary cases (*i.h.*, plus determinism in one of the two sub-cases).

4. *Coming from inside an explicit substitution*:

$$\begin{array}{rl} s'' = & (u, \underline{C\langle D\langle x\rangle[x\leftarrow\langle\cdot\rangle]\rangle}, (x, D[x \leftarrow u], L')\cdot L, T) \\ \to_{\mathsf{var3}} & (x, \underline{C\langle D[x \leftarrow u]\rangle}, L'\cdot L, T) = s' \end{array}$$

   (a) *Log testing*: by *i.h.*, s'' is exhaustible, and its $|L|+1$-log test evolves to

$$\begin{array}{rl} s''_{|L|+1} = & (u, \underline{C\langle D\langle x\rangle[x\leftarrow\langle\cdot\rangle]\rangle}, (x, D[x \leftarrow u], L')\cdot L, \epsilon) \\ \to_{\mathsf{var3}} & (x, \underline{C\langle D[x \leftarrow u]\rangle}, L'\cdot L, \epsilon) = s'_\epsilon \end{array}$$

   which is exhaustible. By Lemma 3.5.10.2, s'_ϵ and s' have the same log tests, which are then successful.

   (b) *Tape testing*: since the tape is unaffected by the transition, this case goes exactly as the other ordinary ones.

□

A.2 Proof of the S-Exhaustible Invariant

Lemma A.2.1 (S-exhaustible invariant)**.** *Let t be a closed term, $\pi \triangleright \Gamma \vdash t : A$ a sequence type derivation for it, and $\rho : \vdash t : \langle A\rangle_\uparrow \to^k_{\mathrm{SIAM}} s$ an initial SIAM run. Then s is S-exhaustible.*

Proof. By induction on k. For $k = 0$ there is nothing to prove because the initial state $s_0 = \vdash t : \langle A\rangle_\uparrow$ has has no judgment nor type tests. Then suppose $\rho' : s_0 \to^{k-1}_{\lambda\mathrm{IAM}} s'$ and that the run continues with $s' \to_{\mathrm{SIAM}} s$. By *i.h.*, s' is S-exhaustible.

Terminology: when a test state satisfies the clause in the definition of S-exhaustible states we say that it is *positive*.

Cases of $s' \to_{\mathrm{SIAM}} s$:

- Case $\to_{\bullet 1}$.

$$s' = \dfrac{\dfrac{\vdash t : S \to A \quad [\vdash]}{\vdash tu : \mathbb{A}\langle\star_\uparrow\rangle (= A)}}{\vdash tu : A} \to_{\bullet 1} \dfrac{\vdash t : S \to \mathbb{A}\langle\star_\uparrow\rangle \quad [\vdash]}{\vdash tu : A} = s$$

  – *judgment tests.* Note that s has the same judgment tests of s', which are positive by the *i.h.*

  – *Type tests.* We first consider the type tests of direction $\uparrow$. Let us s_f be one of them. We observe that there is a corresponding type test s'_f of s', that by *i.h.* it is positive, and that $s'_f \to_{\mathrm{SIAM}} s_f$. Since the machine is deterministic also s_f is positive. Let us now consider a type test s_f of direction $\downarrow$. We observe that there is a corresponding type test s'_f of s', that it is positive by *i.h.*, and that $s_f \to s'_f$. Then s_f is positive.

- Case $\to_{\bullet 2}$. Identical to the previous one.

- Case $\to_{\mathsf{var}}$.

$$s' = \dfrac{\dfrac{}{\vdash x : \mathbb{A}\langle\star_{\uparrow}\rangle_i (= A_i)}\ i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : [\ldots A_i \ldots] \to B}\ \to_{\mathsf{var}}\ \dfrac{\dfrac{}{\vdash x : A_i}\ i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : [\ldots \mathbb{A}\langle\star_{\downarrow}\rangle_i \ldots] \to B}\ = s$$

  - *judgment tests.* judgment tests of s are a subset of judgment tests of s' and thus positive by *i.h.*
  - *Type tests.* Let n be the level of $\mathbb{A}$. Let s^j be the type test of s associated to the j-th triple in $\mathsf{DiPref}([\ldots \mathbb{A} \ldots] \to B)$. Three cases, depending on the index j of s^j:
    1. $j = 1$: then s^1 is $\vdash \lambda x.C\langle x\rangle : \mathbb{A}\langle\star\rangle_{i\uparrow}, [\ldots \langle\cdot\rangle \ldots] \to B$. Note that $s^1 \to_{\mathsf{bt2}} \vdash x : \mathbb{A}\langle\star\rangle_{i\downarrow}, \langle\cdot\rangle$, which has no type tests and has the same judgment tests of s', which by *i.h.* are positive. Hence, s^1 is S-exhaustible.
    2. *j is even*: for s^j (of direction $\downarrow$) there is a corresponding type test s'^{j-1} of odd index of s', having direction $\uparrow$ and such that $s'^{j-1} \to_{\mathsf{var}} s^j$. Thus one can conclude by *i.h.* and determinism of the SIAM.
    3. *$j \neq 1$ is odd*: for s^j (of direction $\uparrow$) there is a corresponding type test s'^{j-1} of even index of s', having direction $\downarrow$ and such that $s^j \to_{\mathsf{bt2}} s'^{j-1}$. Thus one can conclude by *i.h.*

- Case $\to_{\mathsf{bt2}}$.

$$s' = \dfrac{\dfrac{}{\vdash x : A_i (= \mathbb{A}\langle\star\rangle_i)}\ i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : [\ldots \mathbb{A}\langle\star_{\uparrow}\rangle_i \ldots] \to B}\ \to_{\mathsf{bt2}}\ \dfrac{\dfrac{}{\vdash x : \mathbb{A}\langle\star_{\downarrow}\rangle_i}\ i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : [\ldots A_i \ldots] \to B}\ = s$$

  - *judgment tests.* The first type test of s' is $s'^1 :=\vdash \lambda x.C\langle x\rangle : \mathbb{A}\langle\star\rangle_{i\uparrow}, [\ldots \langle\cdot\rangle \ldots] \to B$. Note that $s'^1 \to_{\mathsf{bt2}} \vdash x : \mathbb{A}\langle\star\rangle_{i\downarrow}, \langle\cdot\rangle =: s''$ and that s'' exhausts s'^1, and it is the first such state. Since s'^1 is positive, s'' is S-exhaustible. Note that s'' has the same judgment tests of s, which are then positive.
  - *Type tests.* For each odd type test s^i of s (whose direction is $\uparrow$), the corresponding even type test s'^{i+1} of s' has direction $\downarrow$, is positive by *i.h.*, and such that $s^i \to_{\mathsf{var}} s'^{i+1}$. Then s^i is positive. For each even type test s^i of s (whose direction is $\downarrow$), the corresponding odd type test s'^{i+1} of s' has direction $\uparrow$, is positive by *i.h.*, and such that $s'^{i+1} \to_{\mathsf{bt2}} s^i$. Then s^i is positive by determinism of the SIAM.

- Cases $\to_{\bullet 3}$ and $\to_{\bullet 4}$. They are identical to case $\to_{\bullet 1}$.

- Case $\to_{\mathsf{arg}}$.

$$s' = \dfrac{\vdash t : [..\mathbb{A}\langle\star_{\downarrow}\rangle_i..] \to A \quad \vdash_i u : B_i}{\vdash tu : A}\ \to_{\mathsf{arg}}\ \dfrac{\vdash t : [..B_i..] \to A \quad \vdash_i u : \mathbb{A}\langle\star_{\uparrow}\rangle_i}{\vdash tu : A}\ = s$$

  - *judgment tests.* judgment tests of s are those of s', which are positive by *i.h.*, plus $s^u :=\vdash u : \mathbb{A}\langle\star\rangle_{i\downarrow}, \langle\cdot\rangle$. Please note that $s^u \to_{\mathsf{bt1}} \vdash t : \mathbb{A}\langle\star\rangle_{i\uparrow}, [\ldots \langle\cdot\rangle \ldots] \to A =: s'^t$. Now, s'^t is a type test of s' and by *i.h.* is positive. Then s^u is positive.
  - *Type tests.* For each odd type test s^i of s (whose direction is $\uparrow$), the corresponding even type test s'^{i+1} of s' has direction $\downarrow$, is positive by *i.h.*, and such that $s'^{i+1} \to_{\mathsf{arg}} s^i$. Then s^i is positive by determinism of the SIAM. For each even type test s^i of s' (whose direction is $\downarrow$), the corresponding odd type test s'^{i+1} of s' has direction $\uparrow$, is positive by *i.h.*, and such that $s^i \to_{\mathsf{bt1}} s'^{i+1}$. Then s^i is positive.

- Case $\to_{\mathsf{bt1}}$.

$$s' = \dfrac{\vdash t : [..B_i..] \to A \quad \vdash_i u : \mathbb{A}\langle\star_{\downarrow}\rangle_i}{\vdash tu : A} \to_{\mathsf{bt1}} \dfrac{\vdash t : [..\mathbb{A}\langle\star_{\uparrow}\rangle_i..] \to A \quad \vdash_i u : B_i}{\vdash tu : A} = s$$

  - *judgment tests.* All judgment tests of s are judgment test of s', which are this way positive by *i.h.*
  - *Type tests.* The first type test of s is $s^1 :=\vdash t : \mathbb{A}\langle\star\rangle_{i\uparrow}, [\ldots\langle\cdot\rangle\ldots] \to A$. Please note that $s'^u :=\vdash u : \mathbb{A}\langle\star\rangle_{i\downarrow}, \langle\cdot\rangle$ is a judgment test of s' such that $s'^u \to_{\mathsf{bt1}} s^1$. By *i.h.*, s'^u is positive. By determinism of the SIAM, s^1 is positive.
    For each odd type test s^i of s (whose direction is $\uparrow$), the corresponding even type test s'^{i-1} of s' has direction $\downarrow$, is positive by *i.h.*, and such that $s'^{i-1} \to_{\mathsf{bt1}} s^i$. Then s^i is positive by determinism of the SIAM. For each even type test s^i of s' (whose direction is $\downarrow$), the corresponding odd type test s'^{i-1} of s' has direction $\uparrow$, is positive by *i.h.*, and such that $s^i \to_{\mathsf{arg}} s'^{i-1}$. Then s^i is positive.

□

A.3 Proof of the I-Exhaustible Invariant

Lemma A.3.1 (I-exhaustible invariant). *Let t be a closed term and $\rho : s_t \to^k_{\lambda\mathrm{JAM}} s$ a λJAM run. Then s is I-exhaustible.*

Proof. By induction on k. For $k = 0$ there is nothing to prove because the tape has no logged positions (so it does not decompose) and s has no outer states. Then suppose $\rho' : s_0 \to^{k-1}_{\lambda\mathrm{IAM}} s'$ and that the run continues with $s' \to_{\lambda\mathrm{JAM}} s$. By *i.h.*, s' is I-exhaustible.

Terminology: when a test state satisfies the clause in the definition of I-exhaustible states we say that it is *positive*.

There are many cases to take into account, depending on the transition used to move from s' to s—the cases for $\bullet$ are rather trivial, the other ones instead are subtle, the subtlest one being the jump, that is, transition $\to_{\mathsf{jmp}}$ (it is the last case). First, suppose that $d = \downarrow$. Cases of $s' \to_{\lambda\mathrm{IAM}} s$:

1. Transition $\to_{\bullet 1}$:
$$s' = (\underline{rw}, C, L, T) \to_{\bullet 1} (\underline{r}, C\langle\langle\cdot\rangle w\rangle, L, \bullet\cdot T) = s.$$
   - *Log tests.* Positivity of log tests follows from Lemma 3.5.10.3 and the *i.h.*: s' is a head translation of s, and the lemma states that they have the same log tests, which are positive because s' is I-exhaustible by *i.h.*
   - *Tape tests.* The direction is $\downarrow$ and by Lemma 7.1.1, the tape of s has no logged positions, and so there are no tape tests.
2. Transition $\to_{\bullet 2}$:
$$s' = (\underline{\lambda x.r}, C, L, \bullet \cdot T) \to_{\bullet 2} (\underline{r}, C\langle\lambda x.\langle\cdot\rangle\rangle, L, T) = s$$
   Exactly as the previous case.
3. Transition $\to_{\mathsf{var}}$:
$$s' = (\underline{x}, C\langle\lambda x.D_n\rangle, L_n \cdot L, T) \to_{\mathsf{var}} (\lambda x.D_n\langle x\rangle, \underline{C}, L, (x, C\langle\lambda x.D_n\rangle, L_n \cdot L) \cdot T) = s$$
   - *Log tests.* By Lemma 3.5.10.4, all log tests of s are also log tests of s'. Since the latter is I-exhaustible by *i.h.*, then all these tests are positive.
   - *Tape tests.* Let $l := (x, C\langle\lambda x.D_n\rangle, L_n \cdot L)$. The only tape state of s is $s_l := (\underline{\lambda x.D_n\langle x\rangle}, C, L, l)$ and the one-step run
$$\begin{array}{rll} \sigma : I(s_l) = & (\underline{\lambda x.D_n\langle x\rangle}, C, I(L), I(l)) & \\ = & (\underline{\lambda x.D_n\langle x\rangle}, C, I(L), (x, \lambda x.D_n, I(L_n))) & \\ \to_{\mathsf{bt2},I(l)} & (x, \underline{C\langle\lambda x.D_n\rangle}, I(L_n) \cdot I(L), \epsilon) & \\ = & I(x, \underline{C\langle\lambda x.D_n\rangle}, L_n \cdot L, \epsilon) & = I(l^\circ) \end{array}$$

exhausts l as required. Now, we prove that l° is I-exhaustible. Note that l° is s' with empty tape, so they have the same log tests, which are positive because s' is I-exhaustible by *i.h.*, and l° has no tape tests.

Now, suppose that $d = \uparrow$. Cases of $s' \to_{\lambda\mathrm{IAM}} s$:

1. Transition $\to_{\bullet 3}$:
$$s' = (u, \underline{D\langle\langle\cdot\rangle r\rangle}, L, \bullet \cdot T) \to_{\bullet 3} (ur, \underline{D}, L, T) = s.$$
   (a) *Log tests*. Positivity of log tests follows from Lemma 3.5.10.3 and the *i.h.*: s' is a head translation of s, and the lemma states that they have the same log tests, which are positive because s' is I-exhaustible by *i.h.*
   (b) *Tape tests*. The direction of s is $\uparrow$ and by Lemma 7.1.1, the tape of s has exactly one logged positions l, and so just one tape test s_l. Note that s' also has a tape test s'_l and that by *i.h.* it is positive, that is, there is a run $\sigma : I(s'_l) \to^*_{\lambda\mathrm{IAM}} \to_{\mathsf{bt2},I(l)} I(l^\circ)$ with l° I-exhaustible. Since the direction of s_l and s'_l is $\downarrow$, we have a run $\pi : I(s_l) \to_{\bullet 1} I(s'_l) \to^*_{\lambda\mathrm{IAM}} \to_{\mathsf{bt2},I(l)} I(l^\circ)$ prefixing σ with a step and exhausting s_l.

2. Transition $\to_{\bullet 4}$
$$s' = (u, \underline{D\langle\lambda x.\langle\cdot\rangle\rangle}, L, T) \to_{\bullet 4} (\lambda x.u, \underline{D}, L, \bullet \cdot T) = s.$$
   This case is exactly as the previous one.

3. Transition $\to_{\mathsf{arg}}$:
$$s' = (u, \underline{D\langle\langle\cdot\rangle r\rangle}, L, l \cdot T) \to_{\mathsf{arg}} (\underline{r}, D\langle u\langle\cdot\rangle\rangle, l \cdot L, T) = s.$$
   (a) *Log tests*. The log tests of s are those of s' plus $s_l = (r, \underline{D\langle u\langle\cdot\rangle\rangle}, l \cdot L, \epsilon)$. The former are positive because of the *i.h.*, while about the latter, observe that
$$\begin{array}{rl} I(s_l) = & I(r, \underline{D\langle u\langle\cdot\rangle\rangle}, l \cdot L, \epsilon) \\ = & (r, \underline{D\langle u\langle\cdot\rangle\rangle}, I(l) \cdot I(L), \epsilon) \\ \to_{\mathsf{bt1}} & (\underline{u}, D\langle\langle\cdot\rangle r\rangle, I(L), I(l)) \\ = & I(\underline{u}, D\langle\langle\cdot\rangle r\rangle, L, l) = s'_l. \end{array} \tag{A.1}$$
   Note that s'_l is a tape test of s'. By *i.h.*, there is a run $\sigma : I(s'_l) \to^*_{\lambda\mathrm{IAM}} \to_{\mathsf{bt2},I(l)} I(l^\circ)$ such that l° is I-exhaustible. Now, the run for the test of interest is $\pi : I(s_l) \to_{\lambda\mathrm{IAM}} I(s'_l) \to^*_{\lambda\mathrm{IAM}} \to_{\mathsf{bt2},I(l)} I(l^\circ)$, obtained by prefixing σ with the step in A.1.
   (b) *Tape tests*. The direction is $\downarrow$ of s and by Lemma 7.1.1, the tape of s has no logged positions, and so there are not tape tests for of s.

4. Transition $\to_{\mathsf{jmp}}$:
$$s' = (u, \underline{D\langle r\langle\cdot\rangle\rangle}, (x, C, L') \cdot L, T) \to_{\mathsf{jmp}} (x, \underline{C}, L', T) = s.$$
   Let $l := (x, C, L')$.
   (a) *Log tests*: By *i.h.*, s' is I-exhaustible, and since $s'_l = (u, \underline{D\langle r\langle\cdot\rangle\rangle}, (x, C, L') \cdot L, \epsilon)$ is a log test of s', then it is positive and there exist a run
$$\sigma : I(s'_l) \quad \to^*_{\lambda\mathrm{IAM}} \to_{\mathsf{bt2},I(l)} \quad I(l^\circ)$$
   where l° is I-exhaustible. By Lemma 3.5.10.2, s and l° have the same log tests, which are then positive.
   (b) *Tape tests*. Since the direction of s is $\uparrow$, by Lemma 7.1.1 $|T|_l = 1$, there is only one possible decomposition: $T = T' \cdot l \cdot T''$. Then the only tape test of s is
$$s_l = (\underline{x}, C, L', T' \cdot l)$$

and the only tape test of s' is

$$s'_l = (\underline{u}, D\langle r\langle\cdot\rangle\rangle, (x, C, L') \cdot L, T' \cdot l)$$

that by *i.h.* is positive and so there is a run $\sigma : I(s'_l) \to^*_{\lambda\text{IAM}} \to_{\text{bt2},I(l)} I(l^\circ)$ with l° I-exhaustible.
Now, we show that $I(s_l) \to^+_{\lambda\text{IAM}} I(s'_l)$, that proves the positivity of the tape tests, using an argument analogous to the one for the log tests. Let $l' := (x, C, L')$ and consider the state $s'_{l'} := (u, \underline{D\langle r\langle\cdot\rangle\rangle}, l' \cdot L, \epsilon)$, that it is a log test of s'. By *i.h.*, it is positive, thus there is a run $\pi : I(s'_{l'}) \to^*_{\lambda\text{IAM}} \to_{\text{bt2},I(l')} I(l'^\circ)$. By reversibility, we obtain a run $\pi' : I(l'^\circ)^\perp \to^+_{\lambda\text{IAM}} I(s'_{l'})^\perp$, where $\cdot^\perp$ is the operation on states that changes the direction. Explicitly, we have:

$$\sigma' : I(l'^\circ)^\perp = I(\underline{x}, C, L', \epsilon) \to^+_{\lambda\text{IAM}} I(\underline{u}, D\langle r\langle\cdot\rangle\rangle, (x, C, L') \cdot L, \epsilon) = I(s'_{l'})^\perp$$

By Lemma 3.3.6, we can lift the run to states extended with the tape $T' \cdot l$, obtaining:

$$\pi'' : I(s_l) = I(\underline{x}, C, L', T' \cdot l) \to^+_{\lambda\text{IAM}} I(\underline{u}, D\langle r\langle\cdot\rangle\rangle, (x, C, L') \cdot L, T' \cdot l) = I(s'_l)$$

The run $I(s_l) \to^+_{\lambda\text{IAM}} I(l^\circ)$ obtained by concatenating π'' and σ exhausts s_l.

□

A.4 Proofs of the ↑-Exhaustible Invariant

Lemma A.4.1 (HAM basic invariants). *Let $s = (t, C_n, L, E, T, d)$ be a HAM reachable state. Then*

1. Position and log*: $(t, C_n, L)^E$ is a closed position, and*
2. Tape and direction*: if $d = \downarrow$, then T does not contain any closed positions, otherwise, if $d = \uparrow$, then T contains exactly one closed position.*

Proof. By induction on the length of the run reaching s, together with an immediate inspection of the transitions using the *i.h.* □

Tape Tests. We start by giving the definition of tape tests, that was omitted from the body of the thesis. By the *tape and direction* invariant, there is exactly one closed position $\hat{l}$ on the tape in direction ↑ and none in direction ↓. Essentially, we test only the logged closures added to the tape in a ↓ phase, which—in ↑-states—are those on the right of $\hat{l}$ on the tape. Moreover, in a ↑-state we test them starting on $\hat{l}$ (which records a ↓-state), not from the current position.

Definition A.4.2 (HAM Tape tests). *Let $s = (t, C, L, E, T, d)$ be a HAM state. Tape tests of s are defined depending on whether there is a closed position $\hat{l}$ on T, that is, on whether $|T|_{\hat{l}}$ is 1 or 0.*

- *If $d = \downarrow$ then $s_{\hat{c}} := (t, \underline{C}, L, E, T')$ is a tape test of s of focus $\hat{c}$ for each decomposition $T = T' \cdot \hat{c} \cdot T''$ of the tape.*
- *If $d = \uparrow$ then $s_{\hat{c}} := (x, \underline{D}, L', E', T_1)$ is a tape test of s of focus $\hat{c}$ for each decomposition $T = T' \cdot \hat{l} \cdot T_1 \cdot c \cdot T_2$ with $\hat{l} = (x, D, L')^{E'}$ of the tape.*

Lemma A.4.3 (Invariance properties of HAM environment tests). *Let $s = (t, C_n, L_n, E, T, d)$ be a state. Then:*

1. Direction*: the dual (t, C_n, L_n, E, T, d^1) of s induces the same environment tests;*
2. Tape*: the state (t, C_n, L_n, E, T', d) obtained from s replacing T with an arbitrary tape T' induces the same environment tests;*
3. Weak shift*: let weak contexts be defined by $W ::= \langle\cdot\rangle \mid Wu \mid uW$. Then*

1. *if* $t = W\langle r\rangle$, *then for every* L' *and* T' *the state* $(r, C_n\langle W\rangle, L', E, T', d)$ *induces the same environment tests of* s.
2. *if* $C_n = D_h\langle E_k\rangle$ *with* E_k *weak context, then for every* L' *and* T' *the state* $(E_k\langle t\rangle, C_h, L', E, T', d)$ *induces the same environment tests of* s.

4. Inclusion: *if* $C_n = C_m\langle\lambda x.C_i\rangle$, $L_n = L_i \cdot L_m$ *and* $E = E' \cdot [x{\leftarrow}\hat{c}] \cdot E''$ *then the environment tests of* $(\lambda x.C_i\langle t\rangle, C_m, L_m, E'', T', d)$ *are environment tests of* s.

As in Sect. 7.3, we consider the λJAM and the KAM as special instances of the HAM. In particular the λJAM always uses the $\to_{\mathsf{varJ}}$ transition, while the KAM always the transition $\to_{\mathsf{hop/varK}}$. This way, states can be compared without any kind of projection.

Lemma A.4.4 (Logged closures and closed positions were visited)**.** *Let* $\rho : s_t \to^*_{\mathrm{HAM}} s$.

1. Logged closures: *if* $\hat{c} = (u, C\langle t\langle\cdot\rangle\rangle, E)^L$ *is a logged closure in* s *then* ρ *passes through a state* $(\underline{t}, C\langle\langle\cdot\rangle u\rangle, L, E, T)$ *for some tape* T.

2. Closed positions: *if* $\hat{l} = (u, C, L)^E$ *is a closed position in* s *then* ρ *passes through a state* $(\underline{u}, C, L, E, T)$ *for some tape* T.

Proof. By induction on the length of ρ, together with an immediate inspection of the transitions using the *i.h.* □

Lemma A.4.5 (↑-exhaustible invariant)**.** *Let* s *be a HAM reachable state. Then* s *is* ↑*-exhaustible.*

Proof. By induction on k. For $k = 0$ there is nothing to prove because $s = s_t$ has no tests. Then suppose $s_t \to^{k-1}_{\mathrm{HAM}} s' \to_{\mathrm{HAM}} s$. By *i.h.* $s' = (u, C, L, E, T, d)$ is ↑-exhaustible, and with this hypothesis we need to conclude that s is ↑-exhaustible, too. There are many cases to take into account, depending on the transition used to move from s' to s.

Terminology: when a test satisfies the clause for tests in the definition of ↑-exhaustibility, we say that it is *positive*.

First, suppose that $d = \downarrow$. Cases of $s' \to_{\mathrm{HAM}} s$:

1. Transition $\to_{\bullet 1/\mathsf{app}}$:

$$s' = (\underline{tu}, C, L, E, T) \to_{\bullet 1/\mathsf{app}} (\underline{t}, C\langle\langle\cdot\rangle u\rangle, L, E, (u, C\langle t\langle\cdot\rangle\rangle, E)^L \cdot T) = s$$

   - *Environment tests*. It follows by the *i.h.*, since the environment tests of s are the same of those of s'.
   - *Tape tests*. The first tape test of s is trivially positive since $s_{\hat{c}} = \hat{c}^\circ$. All other tape tests of s are in the form $s_{\hat{c}} = (t, \underline{C\langle\langle\cdot\rangle u\rangle}, L, E, (u, C\langle t\langle\cdot\rangle\rangle, E)^L \cdot T')$, where $T' \cdot \hat{c}$ is a prefix of T. Clearly

$$s_{\hat{c}} = (t, \underline{C\langle\langle\cdot\rangle u\rangle}, L, E, (u, C\langle t\langle\cdot\rangle\rangle, E)^L \cdot T') \to_{\bullet 3} (tu, \underline{C}, L, E, T') = s'_{\hat{c}}$$

     and $s'_{\hat{c}}$ is a tape test for s'. By *i.h.*, $s'_{\hat{c}}$ is positive. Hence, since

$$s_{\hat{c}} \to_{\uparrow} s'_{\hat{c}}$$

     also $s_{\hat{c}}$ is positive.

2. Transition $\to_{\bullet 2/\mathsf{abs}}$:

$$s' = (\underline{\lambda x.t}, C, L, E, \hat{c} \cdot T) \to_{\bullet 2/\mathsf{abs}} (\underline{t}, C\langle\lambda x.\langle\cdot\rangle\rangle, L, [x{\leftarrow}\hat{c}] \cdot E, T) = s$$

   - *Environment tests*. The environment tests of s are those of s' plus $s_{\hat{c}} := (\lambda x.t, \underline{C}, L, E, \epsilon)$. Note that $s_{\hat{c}}$ is also a tape test of s', which by *i.h.* is positive.

- *Tape tests*. Each tape test of s is in the form $s_{\hat{c}'} = (t, \underline{C\langle \lambda x.\langle\cdot\rangle\rangle}, L, [x{\leftarrow}\hat{c}]{\cdot}E, T')$, where $T'{\cdot}\hat{c}'$ is a prefix of T. Clearly

$$s_{\hat{c}'} = (t, \underline{C\langle \lambda x.\langle\cdot\rangle\rangle}, L, [x{\leftarrow}\hat{c}]{\cdot}E, T') \to_{\bullet 4} (\lambda x.t, \underline{C}, L, E, \hat{c}{\cdot}T') = s'_{\hat{c}'}$$

  and $s'_{\hat{c}'}$ is a tape test for s'. Thus, by *i.h.* $s'_{\hat{c}'}$ is positive, and so is $s_{c'}$.

3. *Transition* $\to_{\mathsf{varJ}}$.

$$(\underline{x}, C\langle \lambda x.D_n\rangle, L_n{\cdot}L, E'[x{\leftarrow}\hat{c}]E, T) \to_{\mathsf{varJ}} (\lambda x.D_n\langle x\rangle, \underline{C}, L, E, \hat{l}{\cdot}T)$$

   where $\hat{l} := (x, C\langle \lambda x.D\rangle, L_n{\cdot}L)^{E'[x{\leftarrow}\hat{c}]E}$.

   - *Environment tests*. It follows by the *i.h.*, since all the environment tests of s are environment tests of s'.
   - *Tape tests*. It follows by the *i.h.*, since the tape tests of s are the same of those of s'.

4. *Transition* $\to_{\mathsf{hop/varK}}$.

$$s' = (\underline{x}, C, L, E, T) \to_{\mathsf{hop/varK}} (\underline{u}, D\langle t\langle\cdot\rangle\rangle, (x, C, L)^E{\cdot}L', F, T) = s$$

   where $E = E'{\cdot}[x{\leftarrow}(u, D\langle t\langle\cdot\rangle\rangle, F)^{L'}]{\cdot}E''$.

   - *Environment tests*. By Lemma A.4.4, we have that the run ρ passed through a state $s'' := (\underline{t}, D\langle\langle\cdot\rangle u\rangle, L', F, T')$ for some T'. Note that s'' is a weak shift of s as defined in Lemma 3.5.10.3, and so s'' and s have the same environment tests, which are then positive by *i.h.*
   - *Tape tests*. Note that for each prefix $T'{\cdot}\hat{c}$ of T we have

$$s_{\hat{c}} = (u, \underline{D\langle t\langle\cdot\rangle\rangle}, (x, C, L)^E{\cdot}L', F, T') \to_{\mathsf{jmp}} (x, \underline{C}, L, E, T') = s'_{\hat{c}}$$

     and by *i.h.* $s'_{\hat{c}}$ is positive. Then $s_{\hat{c}}$ is positive.

Then, suppose that $d = \uparrow$. Cases of $s' \to_{\mathrm{HAM}} s$:

1. Transition $\to_{\bullet 3}$:

$$(t, \underline{C\langle\langle\cdot\rangle u\rangle}, L, E, \hat{c}{\cdot}T) \to_{\bullet 3} (tu, \underline{C}, L, E, T)$$

   - *Environment tests*. It follows by the *i.h.*, because all the environment tests of s are environment tests of s' by Lemma 3.5.10.3.
   - *Tape tests*. By *i.h.* since the tape tests of s are the same of those of s'.

2. Transition $\to_{\bullet 4}$:

$$(t, \underline{C\langle \lambda x.\langle\cdot\rangle\rangle}, L, [x{\leftarrow}\hat{c}]{\cdot}E, T) \to_{\bullet 4} (\lambda x.t, \underline{C}, L, E, \hat{c}{\cdot}T)$$

   - *Environment tests*. It follows by the *i.h.*, because all the environment tests of s are environment tests of s'.
   - *Tape tests*. By *i.h.* since the tape tests of s are the same of those of s' ($\hat{c}$ appears on the left of the enriched logged position in the tape, and so needs not to be tested).

3. Transition $\to_{\mathsf{arg}}$:

$$s' = (t, \underline{C\langle\langle\cdot\rangle u\rangle}, L, E, l{\cdot}T) \to_{\mathsf{arg}} (\underline{u}, C\langle t\langle\cdot\rangle\rangle, l{\cdot}L, E, T) = s$$

   - *Environment tests*. By *i.h.* since the environment tests of s are the same of those of s' by Lemma 3.5.10.3.

- *Tape tests*. Since the direction of s is $\downarrow$, by the tape and direction invariant (Lemma A.4.1) there are no closed position on T, and the tape tests of s are in the form $s_{\hat{c}} := (u, \underline{C\langle t\langle\cdot\rangle\rangle}, \hat{l}\cdot L, E, T')$, where $T'\cdot\hat{c}$ is a prefix of T. If $\hat{l} = (x, D, L')^{E'}$, then

$$s_{\hat{c}} = (u, \underline{C\langle t\langle\cdot\rangle\rangle}, (x, D, L')^{E'}\cdot L, E, T') \to_{\mathsf{jmp}} (x, \underline{D}, L', E', T').$$

  Those states in the form $(x, \underline{D}, L', E', T')$ are exactly the tape tests $s'_{\hat{c}}$ of s'. Thus, by *i.h.* they are positive, and so are the tests $s_{\hat{c}}$.

4. *Jumping*.

$$s' = (t, \underline{C\langle u\langle\cdot\rangle\rangle}, \hat{l}\cdot L, E, T) \to_{\mathsf{jmp}} (x, \underline{D}, L', E', T) = s$$

   where $\hat{l} = (x, D, L')^{E'}$.

   - *Environment tests*. By Lemma A.4.4, the run ρ passes through a state $s'' := (\underline{x}, D, L', E', T')$ for some T'. Note that s'' and s differ only for direction and tape, and so by Lemma A.4.3 they have the same environment tests, which are positive by the *i.h.*
   - *Tape tests*. It follows by the *i.h.*, since the tape tests of s are the same of those of s'.

□

Appendix B

Proofs of Bisimulations

B.1 Proof of the SIAM/λIAM Bisimulation

Proposition B.1.1 (SIAM-λIAM bisimulation)**.** *Let t a closed and $\to_{wh}$-normalizable term, and $\pi \triangleright \vdash t : \star$ a type derivation. Then $\simeq_{\mathsf{ext}}$ is a strong bisimulation between S-exhaustible SIAM states on π and λIAM states on t. Moreover, if $s_\pi \simeq_{\mathsf{ext}} s_\lambda$ then s_π is SIAM reachable if and only if s_λ is λIAM reachable.*

Proof. Assuming the bisimulation part of the statement, the moreover part follows from a trivial induction on the length of the initial run, since initial state are bisimilar and the bisimulation is exactly the fact that $\simeq_{\mathsf{ext}}$ is stable by transitions.

For the bisimulation part, we consider each possible transitions. We focus on the half of the proof showing that SIAM transitions are simulated by the λIAM, the other half is essentially identical.

- Case $\to_{\bullet 1}$.

$$\begin{array}{ccccc} & & \dfrac{\vdash t : S \to A \quad [\vdash]}{\vdash tu : \mathbb{A}\langle\star_\uparrow\rangle (= A)} & & \dfrac{\vdash t : S \to \mathbb{A}\langle\star_\uparrow\rangle \quad [\vdash]}{\vdash tu : A} \\ s' = & & & \to_{\bullet 1} & = s \\ & & \simeq_{\mathsf{ext}} & & \end{array}$$

$$s_{\mathsf{ext}}(s) = (\underline{tu}, C_{s'}, L_{\mathsf{ext}}(s'), T_{\mathsf{ext}}(s')) \quad \to_{\bullet 1} \quad (\underline{t}, C\langle\langle\cdot\rangle r\rangle, L_{\mathsf{ext}}(s'), \bullet \cdot T_{\mathsf{ext}}(s')) = s_\lambda$$

  Note that $C_s = C_{s'}\langle\langle\cdot\rangle r\rangle$, $L_{\mathsf{ext}}(s) = L_{\mathsf{ext}}(s')$, and $T_{\mathsf{ext}}(s') = \bullet \cdot T_{\mathsf{ext}}(s)$. Then, $s_\lambda = s_{\mathsf{ext}}(s)$, that is, $s \simeq_{\mathsf{ext}} s_\lambda$.

- Case $\to_{\bullet 2}$. Identical to the previous one.

- Case $\to_{\mathsf{var}}$.

$$\begin{array}{ccccc} & & \dfrac{}{\vdash x : \mathbb{A}\langle\star_\uparrow\rangle_i (= A_i)}\, i & & \dfrac{}{\vdash x : A_i}\, i \\ & & \vdots & & \vdots \\ s' = & & \dfrac{}{\vdash \lambda x.D_n\langle x\rangle : [\ldots A_i \ldots] \to B} & \to_{\mathsf{var}} & \dfrac{}{\vdash \lambda x.D_n\langle x\rangle : [\ldots \mathbb{A}\langle\star_\downarrow\rangle_i \ldots] \to B} = s \\ & & \simeq_{\mathsf{ext}} & & \end{array}$$

$$s_{\mathsf{ext}}(s) = (\underline{x}, \underbrace{C\langle\lambda x.D_n\rangle}_{=C_{s'}}, \underbrace{L_n \cdot L}_{=L_{\mathsf{ext}}(s')}, T_{\mathsf{ext}}(s')) \quad \to_{\mathsf{var}} \quad (\lambda x.D_n\langle x\rangle, \underline{C}, L, (x, \lambda x.D_n, L_n) \cdot T_{\mathsf{ext}}(s')) = s_\lambda$$

  First of all, C_s has shape $C\langle\lambda x.D_n\rangle$ for some n has the descending path from the focused judgment to the final judgment passes through the showed T-λ rule. Then $C_{s'} = C$.

  About the log, by Lemma 4.3.11 there is a correspondance between the level of term contexts and the length of the extracted log, so that $L_{\mathsf{ext}}(s)$ has at least length n, that is, $L_{\mathsf{ext}}(s') = L_n \cdot L$, and $L_{\mathsf{ext}}(s) = L$.

  About the tape, note that $T_{\mathsf{ext}}(s) = l_{\mathsf{ext}}(s_f^1) \cdot T^s_{\mathsf{ext}}(\mathbb{A}, 1)$ where s_f^1 is the first type test of s. To show that $s_{\mathsf{ext}}(s) = (x, \lambda x.D_n, L_n) \cdot T_{\mathsf{ext}}(s')$ we have to show two things:

1. $l_{\mathsf{ext}}(s^1_f) = (x, \lambda x.D_n, L_n)$. Note that s^1_f is $\vdash \lambda x.C\langle x\rangle : \mathbb{A}\langle\star\rangle_{i\uparrow}, [\ldots\langle\cdot\rangle\ldots] \to B$. Note that $s^1_f \to_{\mathsf{bt2}} \vdash x : \mathbb{A}\langle\star\rangle_{i\downarrow}, \langle\cdot\rangle = s''$, where s'' focusses on the same judgment of s', and that s'' is the state that S-exhausts s^1_f. By definition of extraction, $l_{\mathsf{ext}}(s^1_f) = (x, \lambda x.D_n, L_n)$.
2. $T^s_{\mathsf{ext}}(\mathbb{A}, 1) = T_{\mathsf{ext}}(s')$, that is, $T^s_{\mathsf{ext}}(\mathbb{A}, 1) = T^{s'}_{\mathsf{ext}}(\mathbb{A}, 0)$. Note that $T^s_{\mathsf{ext}}(\mathbb{A}, 1)$ and $T^{s'}_{\mathsf{ext}}(\mathbb{A}, 0)$ may differ only in the content of logged positions (obtained by extracting from tape tests), which is the only thing that depends on the direction and the state, the rest being uniquely determined by the type context $\mathbb{A}$. Here one has to repeat the reasoning done in the $\to_{\mathsf{bt2}}$ case of the proof of the S-exhaustible invariant (Lemma 4.3.8), that shows that the tape test of index $i > 1$ for s and the one of index $i - 1$ of s' exhaust on the same state, and thus induce the same logged position. Then $T^s_{\mathsf{ext}}(\mathbb{A}, 1) = T_{\mathsf{ext}}(s')$.

Then $s_{\mathsf{ext}}(s) = (x, \lambda x.D_n, L_n) \cdot T_{\mathsf{ext}}(s')$, and so $s_\lambda = s_{\mathsf{ext}}(s)$, that is, $s \simeq_{\mathsf{ext}} s_\lambda$.

- Case $\to_{\mathsf{bt2}}$.

$$s' = \dfrac{\dfrac{}{\vdash x : A_i (= \mathbb{A}\langle\star\rangle_i)}\, i \quad \vdots}{\vdash \lambda x.C\langle x\rangle : [\ldots \mathbb{A}\langle\star_\uparrow\rangle_i \ldots] \to B} \quad \to_{\mathsf{bt2}} \quad \dfrac{\dfrac{}{\vdash x : \mathbb{A}\langle\star_\downarrow\rangle_i}\, i \quad \vdots}{\vdash \lambda x.C\langle x\rangle : [\ldots A_i \ldots] \to B} = s$$

$$\simeq_{\mathsf{ext}}$$

$$s_{\mathsf{ext}}(s') = (\underline{\lambda x.D_n\langle x\rangle}, C_{s'}, L_{\mathsf{ext}}(s'), \underbrace{(x, \lambda x.D_n, L_n)\cdot T^{s'}_{\mathsf{ext}}(\mathbb{A}, 1)}_{=T_{\mathsf{ext}}(s')}) \quad \to_{\mathsf{bt2}} \quad (x, \underline{C_{s'}\langle\lambda x.D_n\rangle}, L_n \cdot L_{\mathsf{ext}}(s'), T^{s'}_{\mathsf{ext}}(\mathbb{A}, 1)) = s'_\lambda$$

About the tape of $s_{\mathsf{ext}}(s')$, note that $T_{\mathsf{ext}}(s') = l_{\mathsf{ext}}(s'^1_f)\cdot T^{s'}_{\mathsf{ext}}(\mathbb{A}, 1)$ where s^1_f is the first type test of s'. We have to show that s^1_f exhausts on x, so that $l_{\mathsf{ext}}(s^1_f) = (x, \lambda x.D_n, L_n)$ for some L_n. Note that s^1_f is $\vdash \lambda x.C\langle x\rangle : \mathbb{A}\langle\star\rangle_{i\uparrow}, [\ldots\langle\cdot\rangle\ldots] \to B$. Note that $s^1_f \to_{\mathsf{bt2}} \vdash x : \mathbb{A}\langle\star\rangle_{i\downarrow}, \langle\cdot\rangle = s''$, where s'' focusses on the same judgment of s, and that s'' is the state that S-exhausts s^1_f. By definition of extraction, $l_{\mathsf{ext}}(s^1_f) = (x, \lambda x.D_n, L_n)$ where L_n is the extraction of the first n judgment tests of s. Then $C_s = C_{s'}\langle\lambda x.D_n\rangle$ and $L_{\mathsf{ext}}(s) = L_n \cdot L_{\mathsf{ext}}(s')$.

About the tape, for s we have to prove that $T^{s'}_{\mathsf{ext}}(\mathbb{A}, 1) = T_{\mathsf{ext}}(s) = T^s_{\mathsf{ext}}(\mathbb{A}, 0)$. This is done as for $\to_{\mathsf{var}}$, mimicking the reasoning in the proof of the S-exhaustible invariant (Lemma 4.3.8). Then, $s_\lambda = s_{\mathsf{ext}}(s)$, that is, $s \simeq_{\mathsf{ext}} s_\lambda$.

- Cases $\to_{\bullet 3}$ and $\to_{\bullet 4}$. They are identical to case $\to_{\bullet 1}$.
- Case $\to_{\mathsf{arg}}$.

$$s' = \dfrac{\vdash t : [\ldots \mathbb{A}\langle\star_\downarrow\rangle_i \ldots] \to A \quad \vdash_i u : B_i (= \mathbb{A}\langle\star_\downarrow\rangle_i)}{\vdash tu : A} \quad \to_{\mathsf{arg}} \quad \dfrac{\vdash t : [\ldots B_i \ldots] \to A \quad \vdash_i u : \mathbb{A}\langle\star_\uparrow\rangle_i}{\vdash tu : A} = s$$

$$\simeq_{\mathsf{ext}}$$

$$s_{\mathsf{ext}}(s') = (t, \underbrace{\underline{D\langle\langle\cdot\rangle u\rangle}}_{=\underline{C_{s'}}}, L_{\mathsf{ext}}(s'), \underbrace{l_{\mathsf{ext}}(s'^1_f)\cdot T^{s'}_{\mathsf{ext}}(\mathbb{A}, 1)}_{=T_{\mathsf{ext}}(s')}) \quad \to_{\mathsf{arg}} \quad (\underline{u}, D\langle t\langle\cdot\rangle\rangle, l_{\mathsf{ext}}(s'^1_f)\cdot L_{\mathsf{ext}}(s'), T^{s'}_{\mathsf{ext}}(\mathbb{A}, 1)) = s'_\lambda$$

where s'^1_f is the first type test of s'. Obviously, $C_s = D\langle t\langle\cdot\rangle\rangle$. For the log we have to show that $L_{\mathsf{ext}}(s)$ is equal to $l_{\mathsf{ext}}(s'^1_f)\cdot L_{\mathsf{ext}}(s')$, which amounts to showing that the first judgment test s^1 of s exhausts on the same state as the first tape test s'^1_f of s'. This is exactly the reasoning done in the proof of the S-exhaustible invariant. Similarly, one obtains that $T^{s'}_{\mathsf{ext}}(\mathbb{A}, 1) = T_{\mathsf{ext}}(s) = T^s_{\mathsf{ext}}(\mathbb{A}, 0)$.

- Case $\to_{\mathsf{bt1}}$.

$$s' = \dfrac{\vdash t : [\dots B_i \dots] \to A \quad \vdash_i u : \mathbb{A}\langle\star_{\downarrow}\rangle_i (= B_i)}{\vdash tu : A} \to_{\mathsf{bt1}} \dfrac{\vdash t : [\dots \mathbb{A}\langle\star_{\uparrow}\rangle_i \dots] \to A \quad \vdash_i u : B_i}{\vdash tu : A} = s$$

$$\simeq_{\mathsf{ext}}$$

$$s_{\mathsf{ext}}(s') = (u, \underbrace{D\langle t\langle\cdot\rangle\rangle}_{=C_{s'}}, \underbrace{l_{\mathsf{ext}}(s'^1_f) \cdot L}_{=L_{\mathsf{ext}}(s')}, T_{\mathsf{ext}}(s')) \quad \to_{\mathsf{bt1}} \quad (t, D\langle\langle\cdot\rangle u\rangle, L, l_{\mathsf{ext}}(s'^1_f) \cdot T_{\mathsf{ext}}(s')) = s'_\lambda$$

where s'^1_f is the first judgment test of s'. Obviously, $C_s = D\langle\langle\cdot\rangle u\rangle$. For the log, there is nothing to prove. For the tape, we have to show that $T_{\mathsf{ext}}(s)$ is equal to $l_{\mathsf{ext}}(s^1_f) \cdot T_{\mathsf{ext}}(s')$, which amounts to showing two things. First, that the first tape test s^1 of s exhausts on the same state as the first judgment test s'^1_f of s'. Second, that $T^s_{\mathsf{ext}}(\mathbb{A}, 1) = T_{\mathsf{ext}}(s') = T^{s'}_{\mathsf{ext}}(\mathbb{A}, 0)$. Both points follow exactly the reasoning done in the proof of the S-exhaustible invariant.

□

B.2 Proof of the Loop Preserving Bisimulation

Proposition B.2.1. ▶ *is a loop-preserving improvement between SIAM states.*

Proof. We inspect the 4 cases of the definition of ▶.

- Rule rdx :$\vdash (H\langle(\lambda x.t)u\rangle, K) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\mathsf{rdx}} \vdash (H\langle t\{x\leftarrow u\}\rangle, K) : \mathbb{A}\langle\star\rangle$. Cases for ↑ (by cases of H):
  - $H = \langle\cdot\rangle$. The diagram is closed by rule body:

$$\begin{array}{ccccc} \vdash ((\lambda x.t)u, K) : \mathbb{A}\langle\star\rangle & \to_{\mathsf{SIAM}} & \vdash (\lambda x.t, K\langle\langle\cdot\rangle u\rangle) : S \to \mathbb{A}\langle\star\rangle & \to_{\mathsf{SIAM}} & \vdash (t, K\langle(\lambda x.\langle\cdot\rangle)u\rangle) : \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\mathsf{rdx}} & & & & \blacktriangleright_{\mathsf{body}} \\ \vdash (t\{x\leftarrow u\}, K) : \mathbb{A}\langle\star\rangle & & = & & \vdash (t\{x\leftarrow u\}, K) : \mathbb{A}\langle\star\rangle \end{array}$$

  - $H = Gs$. The diagram is closed by rule $\blacktriangleright_{\mathsf{rdx}}$:

$$\begin{array}{ccc} \vdash (G\langle r\rangle s, K) : \mathbb{A}\langle\star\rangle & \to_{\mathsf{SIAM}} & \vdash (G\langle r\rangle, K\langle\langle\cdot\rangle s\rangle) : S \to \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\mathsf{rdx}} & & \blacktriangleright_{\mathsf{rdx}} \\ \vdash (G\langle w\rangle s, K) : \mathbb{A}\langle\star\rangle & \to_{\mathsf{SIAM}} & \vdash (G\langle w\rangle, K\langle\langle\cdot\rangle s\rangle) : S \to \mathbb{A}\langle\star\rangle \end{array}$$

  Cases for ↓ (by cases of K):
  - $K = \langle\cdot\rangle$. Both machines are stuck.

$$\begin{array}{c} \vdash (r, \langle\cdot\rangle) : \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\mathsf{rdx}} \\ \vdash (w, \langle\cdot\rangle) : \mathbb{A}\langle\star\rangle \end{array}$$

  - $K = G\langle\langle\cdot\rangle s\rangle$. Two subcases depending on the type context. If the focus is on the right of the arrow the diagram is closed by rule rdx.

$$\begin{array}{ccc} \vdash (r, G\langle\langle\cdot\rangle s\rangle) : S \to \mathbb{A}\langle\star\rangle & \to_{\mathsf{SIAM}} & \vdash (rs, G) : \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\mathsf{rdx}} & & \blacktriangleright_{\mathsf{rdx}} \\ \vdash (w, G\langle\langle\cdot\rangle s\rangle) : S \to \mathbb{A}\langle\star\rangle & \to_{\mathsf{SIAM}} & \vdash (ws, G) : \mathbb{A}\langle\star\rangle \end{array}$$

If the focus is on the left of the arrow the diagram is closed by rule ext.

$$\begin{array}{ccc} \vdash (r, G\langle\langle\cdot\rangle s\rangle) : [\dots \mathbb{A}\langle\star\rangle \dots] \to A & \to_{\text{SIAM}} & \vdash (s, G\langle r\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\text{rdx}} & & \blacktriangleright_{\text{ext}} \\ \vdash (w, G\langle\langle\cdot\rangle s\rangle) : [\dots \mathbb{A}\langle\star\rangle \dots] \to A & \to_{\text{SIAM}} & \vdash (s, G\langle w\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle \end{array}$$

- Rule body: $\vdash (t, H\langle(\lambda x.D)u\rangle) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\text{body}} \vdash (t\{x{\leftarrow}u\}, H\langle D\{x{\leftarrow}u\}\rangle) : \mathbb{A}\langle\star\rangle$. Cases of $\uparrow$ (by cases of t):
  - $t = rw$. Trivially closed by rule body.
  - $t = \lambda y.r$. If $t : \star$ both machines are stuck. If $t : S \to A$, the diagram is trivially closed by rule body.
  - $t = x$. Diagram closed by rule arg.

$$\begin{array}{ccccc} \vdash (x, H\langle(\lambda x.D)u\rangle : \mathbb{A}\langle\star\rangle & \to & \vdash (\lambda x.D\langle x\rangle, H\langle\langle\cdot\rangle u\rangle) : [\dots\mathbb{A}\langle\star\rangle\dots] \to B & \to & \vdash (u, H\langle(\lambda x.D\langle x\rangle)\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\text{body}} & & & & \blacktriangleright_{\text{arg}} \\ \vdash (u, H\langle D\{x{\leftarrow}u\}\rangle) : \mathbb{A}\langle\star\rangle & & = & & \vdash (u, H\langle D\{x{\leftarrow}u\}\rangle) : \mathbb{A}\langle\star\rangle \end{array}$$

  Cases of $\downarrow$ (by cases of D):
  - $D = \langle\cdot\rangle$. The diagram is closed by rule rdx

$$\begin{array}{ccccc} \vdash (t, H\langle(\lambda x.\langle\cdot\rangle)u\rangle) : \mathbb{A}\langle\star\rangle & \to_{\text{SIAM}} & \vdash (\lambda x.t, H\langle\langle\cdot\rangle u\rangle) : S \to \mathbb{A}\langle\star\rangle & \to_{\text{SIAM}} & \vdash ((\lambda x.t)u, H) : \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\text{body}} & & & & \blacktriangleright_{\text{rdx}} \\ \vdash (t\{x{\leftarrow}u\}, H) : \mathbb{A}\langle\star\rangle & & = & & \vdash (t\{x{\leftarrow}u\}, H) : \mathbb{A}\langle\star\rangle \end{array}$$

  - $D = E\langle\lambda y.\langle\cdot\rangle\rangle$, $D = E\langle\langle\cdot\rangle r\rangle$ and $D = E\langle r\langle\cdot\rangle\rangle$. The diagram is trivially closed by rule body.

- Rule $\blacktriangleright_{\text{arg}}$: $\vdash (t, H\langle(\lambda x.D\langle x\rangle)E\rangle) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\text{arg}} \vdash (t, H\langle D\{x{\leftarrow}E\langle t\rangle\}\langle E\rangle\rangle) : \mathbb{A}\langle\star\rangle$. Cases of $\uparrow$ (by cases of t) are all trivial: they are closed by rule $\blacktriangleright_{\text{arg}}$ itself. The only non trivial case for $\downarrow$ (by cases of E) is when $E = \langle\cdot\rangle$.

$$\begin{array}{ccccc} \vdash (t, H\langle(\lambda x.D\langle x\rangle)\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle & \to & \vdash (\lambda x.D\langle x\rangle, H\langle\langle\cdot\rangle t\rangle) : [\dots\mathbb{A}\langle\star\rangle\dots] \to B & \to & \vdash (x, H\langle(\lambda x.D)t\rangle : \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\text{arg}} & & & & \blacktriangleright_{\text{body}} \\ \vdash (t, H\langle D\{x{\leftarrow}t\}\rangle) : \mathbb{A}\langle\star\rangle & & = & & \vdash (t, H\langle D\{x{\leftarrow}t\}\rangle) : \mathbb{A}\langle\star\rangle \end{array}$$

- Rule $\blacktriangleright_{\text{ext}}$: $\vdash (t, K\langle H\langle(\lambda x.r)u\rangle D\rangle) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\text{ext}} \vdash (t, K\langle H\langle r\{x{\leftarrow}u\}\rangle D\rangle) : \mathbb{A}\langle\star\rangle$. Cases of $\uparrow$ (by cases of t) are all trivial: they are closed by rule $\blacktriangleright_{\text{ext}}$ itself. The only non trivial case for $\downarrow$ (by cases of D) is when $D = \langle\cdot\rangle$. We put $s := H\langle(\lambda x.r)u\rangle$ and $w := H\langle r\{x{\leftarrow}u\}\rangle$.

$$\begin{array}{ccc} \vdash (t, K\langle s\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle & \to_{\text{SIAM}} & \vdash (s, K\langle\langle\cdot\rangle t\rangle) : [\dots \mathbb{A}\langle\star\rangle \dots] \to A \\ \blacktriangleright_{\text{ext}} & & \blacktriangleright_{\text{rdx}} \\ \vdash (t, K\langle w\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle & \to_{\text{SIAM}} & \vdash (w, K\langle\langle\cdot\rangle t\rangle) : [\dots \mathbb{A}\langle\star\rangle \dots] \to A \end{array}$$

□

B.3 Proof of the TIAM/λIAM Bisimulation

The aim of this section is to explain the strong bisimulation between the TIAM and the λIAM, that is essentially the same between the SIAM and the λIAM. This is the reason why we have chosen to put this chapter in the Appendix. However, for the sake of readability we report all the explanations, very similar to those of the SIAM.

A striking point of the TIAM is that it does not have the log nor the tape. They are encoded in the position of the judgment occurrence J and in the type context $\mathbb{A}$ of its states, as we shall show.

Relating Logs and Tapes with Typed Positions. In the λIAM, the log $L = l_1 \cdot \ldots \cdot l_n$ has a logged position for every argument $u_1, \ldots, u_n$ in which the position of the current state is contained. The argument u_i is the answer to the query of an argument for the variable in the logged position l_i. The TIAM does not keep a trace of the variables for which it completed a query, but the answers to those (forgotten) queries are simply given by the sub-derivations for $u_1, \ldots, u_n$ in which the current judgment occurrence J is contained—the way in which l_k identifies a copy of u_k in the λIAM corresponds on the type derivation π to the index i of the leaf (in the tree of sub-derivations) typing u_k in which J is located. Note that the λIAM manipulates the log only via transitions $\to_{\mathsf{arg}}$ and $\to_{\mathsf{bt1}}$, that on the TIAM correspond exactly to entering/exiting derivations for arguments. The tape, instead, contains logged positions for which the λIAM either has not yet found the associated argument, or it is backtracking to. Note that the λIAM puts logged positions on the tape via transitions $\to_{\mathsf{var}}$ and $\to_{\mathsf{bt1}}$, and removes them using $\to_{\mathsf{arg}}$ and $\to_{\mathsf{bt2}}$. By looking at Fig. 5.2, it is clear that there is a logged position on the λIAM tape for every type sequence of the flattening of T in which it lies the hole $\langle\cdot\rangle$ of the current type context $\mathbb{A}$ of the TIAM.

Extracting λIAM States. These ideas are used to extract from every TIAM state s a λIAM state $\mathsf{ext}(s)$ in a quite technical way. In particular, the extraction process retrieves a log $L_{\mathsf{ext}}(s)$ from the judgment J of s and a tape $T_{\mathsf{ext}}(s)$ from the type context $\mathbb{A}$ of s, using a sophisticated *T-exhaustible invariant* of the TIAM to retrieve the exact shape of the logged positions in $L_{\mathsf{ext}}(s)$ and $T_{\mathsf{ext}}(s)$.

Let us give a high-level description of how extraction works. The invariant is based on the pairing of every TIAM state s with a set of *test states*, some coming from the judgment J of s, called *judgment tests*, and some coming from the type context $\mathbb{A}$, called *type (context) tests*. The invariant guarantees a certain recursive property of each test state. The extraction process uses this property to extract a logged position $l_{s'}$ from each test state s' of s.

Given a TIAM state $s = (\pi, J, \mathbb{A}, d)$, its judgment tests are associated to the T-@ rules having J in their right sub-derivation. Their extractions give logged positions $l_1 \cdot \ldots \cdot l_n$ forming the extracted log $L_{\mathsf{ext}}(s)$, following the correspondence described above.

Type tests are associated to the leaf contexts surrounding the hole of $\mathbb{A}$. The extraction of the tape $T_{\mathsf{ext}}(s)$ from $\mathbb{A}$ is done according to the following schema:

$$\begin{array}{lcl} T_{\mathsf{ext}}(\langle\cdot\rangle) & := & \epsilon \\ T_{\mathsf{ext}}(T \to \mathbb{A}) & := & \bullet \cdot T_{\mathsf{ext}}(\mathbb{A}) \\ T_{\mathsf{ext}}(\mathbb{L}\langle\mathbb{A}\rangle \to B) & := & l_{\mathsf{ext}}(s_{\mathbb{L}}) \cdot T_{\mathsf{ext}}(\mathbb{A}) \end{array}$$

where $s_{\mathbb{L}}$ is the state test associated to the leaf context $\mathbb{L}$.

The extraction process induces a relation $s \simeq_{\mathsf{ext}} \mathsf{ext}(s)$ that is easily proved to be a strong bisimulation between the TIAM and the λIAM.

T-Exhaustible Invariant.

We start by defining the notions of typed tests used to define T-exhaustible states.

Type Positions and Generalized States. To define tests, we have to consider a slightly more general notion of TIAM state. In Sect. 5.2, a state is a quadruple $(\pi, J, \mathbb{A}, d)$ where J is an occurrence of a judgment $\Gamma \vdash u : A$ in π, d is a direction, and $\mathbb{A}$ is a linear type context isolating an occurrence of $\star$ in A. The generalization simply is to consider linear type contexts $\mathbb{A}$ such that $\mathbb{A}\langle B\rangle = A$ for some B, that is, not necessarily isolating $\star$. A pair $(B, \mathbb{A})$ such that $\mathbb{A}\langle B\rangle = A$ is called a position in A.

Note that the TIAM can be naturally adapted to this more general notion of state, that follows an arbitrary formula B, not necessarily $\star$ —it can be found in Fig. B.1, and it amounts to simply replace $\star$ with B.

To easily manage TIAM states we also use a concise notations, writing $\vdash t : A, \mathbb{A}$ for a state $s = (\pi, J, (A, \mathbb{A}), d)$ where J is $\Gamma \vdash t : \mathbb{A}\langle A\rangle$ for some Γ, potentially specifying the direction via colors and under/over-lining.

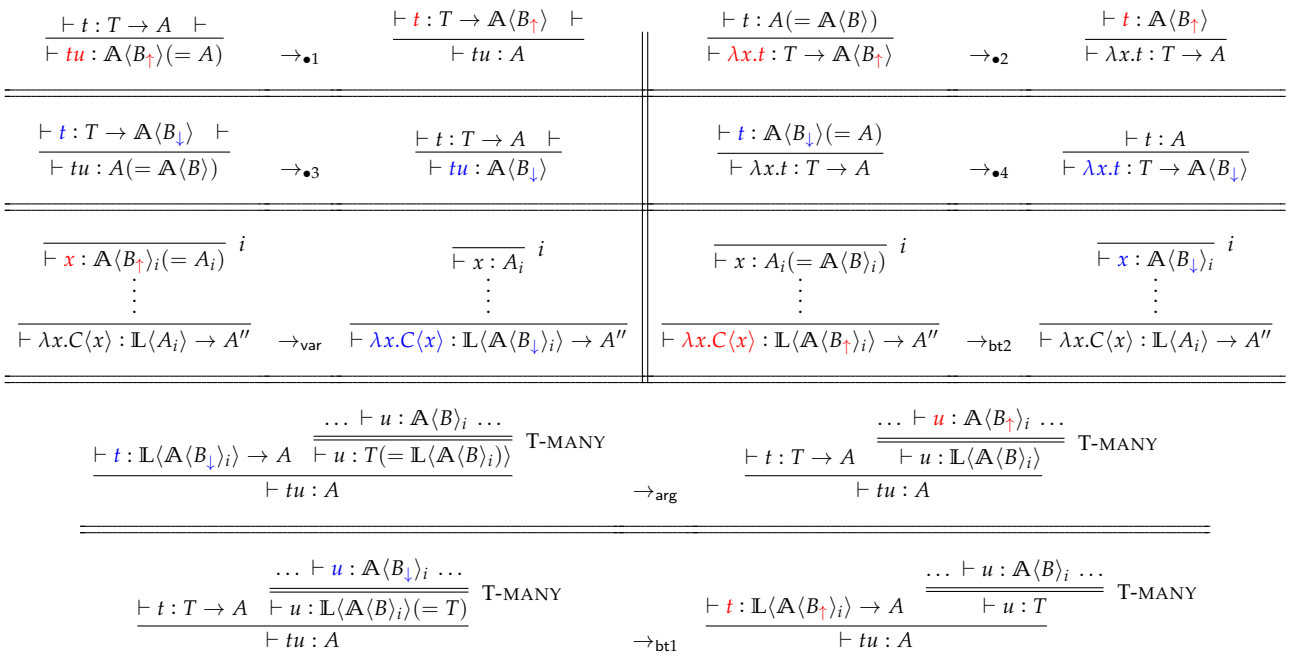


FIGURE B.1: The transitions of the (Generalized) Tree IAM (TIAM).

TIAM Tests. Given a TIAM state $s = (\pi, J, (A, \mathbb{A}), d)$, the underlying idea is that the judgment occurrence J encodes the log of the λIAM, while the type context $\mathbb{A}$ encodes the tape. It is then natural to define two kinds of test, one for judgments and one for type contexts.

The intuition is that a test focuses on (the occurrence of) a leaf B of a tree T related to s, and that these leaf elements play the role of logged positions in the λIAM. These leaf elements are of two kinds:

1. *Elements containing J*: those in which the focused judgment J itself is contained, corresponding to the logged positions in the log of the λIAM. Note that the positions on the log are those for which the λIAM has previously found the corresponding arguments. In the TIAM these arguments are exactly those in which the focused judgment is contained.
2. *Elements appearing in $\mathbb{A}$*: those in the right-hand type of s in which the focused type A is contained, corresponding to the logged positions on the tape of the λIAM. They correspond to λIAM queries for which the argument has not yet been found, or positions to which the λIAM is backtracking to.

Each one of these elements is then identified by a judgment occurrence J' and a position $(B, \mathbb{A}')$ in the right-hand type of J'.

Definition B.3.1 (Focus). *A* focus f *in a derivation π is a pair $f = (J, (A, \mathbb{A}))$ of a judgment occurrence J and of a type position $(A, \mathbb{A})$ in the right-hand type $\mathbb{A}\langle A\rangle$ of J.*

The intuition is that exhausting a test $s_{J,(A,\mathbb{A})}$ in π shall amount to retrieve the axiom of π of type A that would be substituted by that sequence element of type A by reducing π via cut-elimination—the definition of exhaustible tests is given below, after the definition of tests.

Definition B.3.2 (judgment tests). *Let $s = (\pi, J, (A, \mathbb{A}), d)$ be a TIAM state. Let r_i be i-th* T-MANY *rule tree found traversing π by descending from the focused judgment J towards the final judgment of π. Let J_i be the topmost traversed judgment of r_i in such a descent. Let J_i be $\Gamma \vdash t : B$. Then $s^i_f = (\pi, J_i, (B, \langle\cdot\rangle), \downarrow)$ is the i-th judgment test of s, having as focus $f := (J_i, (B, \langle\cdot\rangle))$.*

We often omit the judgment from the focus, writing simply $s_{(B,\langle\cdot\rangle)}$, and even concisely note s_f as $\vdash t : B, \langle\cdot\rangle_{\downarrow}$.

Note that judgment tests always have type context $\langle\cdot\rangle$. According to the intended correspondence judgment/ log and type context/tape between the TIAM and the λIAM, having type context $\langle\cdot\rangle$ corresponds to the fact that the log tests of the λIAM have an empty tape.

Type (Context) Tests. While judgment tests depend only on the judgment occurrence J of a state $s = (\pi, J, (A, \mathbb{A}), d)$, type context tests—dually—fix J and depend only on the type context $\mathbb{A}$ of s, that is, they all focus on sequence elements of the form $(J, (B, \mathbb{A}'))$ where $\mathbb{A}'\langle B\rangle = \mathbb{A}\langle A\rangle$ and $\mathbb{A} = \mathbb{A}'\langle\mathbb{A}''\rangle$ for some type context $\mathbb{A}''$. Namely, there is one type context test (shortened to *type test*) for every flattened tree (*i.e.* sequence) in which the hole of $\mathbb{A}$ is contained. We need some notions about type contexts, in particular a notion of level analogous to the one for term contexts.

Terminology About Type Contexts. Define type contexts $\mathbb{A}_n$ of level $n \in \mathbb{N}$ as follows:

$$\begin{array}{lcl} \mathbb{A}_0 & := & \langle\cdot\rangle \mid T \to \mathbb{A}_0 \\ \mathbb{A}_{n+1} & := & \mathbb{L}\langle\mathbb{A}_n\rangle \to A \mid T \to \mathbb{A}_{n+1} \end{array}$$

Clearly, every type context $\mathbb{A}$ can be seen as a type context $\mathbb{A}_n$ for a unique n, and viceversa a type context of level n is also simply a type context—the level is then sometimes omitted. A *prefix* of a context $\mathbb{A}$ is a context $\mathbb{A}'$ such that $\mathbb{A}'\langle\mathbb{A}''\rangle = \mathbb{A}$ for some $\mathbb{A}''$. Given $\mathbb{A}$ of level $n > 0$, there is a smallest prefix context $\mathbb{A}|_i$ of level $0 < i \leq n$, and it has the form $\mathbb{A}'\langle\mathbb{L} \to A\rangle$ for a type context $\mathbb{A}'$ of level $i - 1$.

Definition B.3.3 (Type tests). *Let $s = (\pi, J, (A, \mathbb{A}), d)$ be a TIAM state and n be the level of $\mathbb{A}$. The sequence of directed prefixes* $\mathsf{DiPref}(\mathbb{A})$ *of $\mathbb{A}$ is the sequence of pairs $(\mathbb{A}', d')$, where $\mathbb{A}'$ is a prefix of $\mathbb{A}$, defined as follows:*

$$\begin{array}{lcll} \mathsf{DiPref}(\mathbb{A}) & := & [\cdot] & \text{if } n = 0 \\ \mathsf{DiPref}(\mathbb{A}) & := & [(\mathbb{A}|_1, \uparrow^0), \ldots, (\mathbb{A}|_n, \uparrow^{n-1})] & \text{if } n > 0 \end{array}$$

The i-th directed prefix (from left to right) $(\mathbb{A}', d')$ in $\mathsf{DiPref}(\mathbb{A})$ *induces the type test $s^i_f := (\pi, J, (\mathbb{A}''\langle A\rangle, \mathbb{A}'), d')$ of s and focus $f := (J, (\mathbb{A}''\langle B\rangle, \mathbb{A}'))$, where $\mathbb{A}''$ is the unique type context such that $\mathbb{A} = \mathbb{A}'\langle\mathbb{A}''\rangle$.*

According to the idea that type tests correspond to the tape tests of the λIAM, note that the first element (on the left) of the sequence $\mathsf{DiPref}(\mathbb{A})$ has $\uparrow$ direction, and that the direction alternates along the sequence. This is the analogous to the fact that the tape test associated to the first logged position on the tape (from left to right) has always direction $\downarrow$, and passing to the test of the next logged position on the tape switches the direction.

Definition B.3.4 (State respecting a focus). *Let $f = (J, (A, \mathbb{A}))$ be a focus. A TIAM state s respects f if it is an axiom $\vdash x : \langle A\rangle_\downarrow$ for some variable x (the typing context of s, which is omitted by convention, is $x : [A]$).*

Definition B.3.5 (T-Exhaustible states). *The set $\mathcal{E}_T$ of T-exhaustible states is the smallest set such that if $s \in \mathcal{E}_T$, then for each type or judgment test of s_f of focus f there exists a run $\rho : s_f \to^*_{\mathrm{TIAM}} \to_{\mathsf{bt2}} s'$ where s' respects f and for the shortest such run $s' \in \mathcal{E}_T$.*

Lemma B.3.6 (T-exhaustible invariant). *Let t be a closed term, $\pi \triangleright \Gamma \vdash t : A$ a tree type derivation for it, and $\sigma : \vdash t : \langle A\rangle_\uparrow \to^k_{\mathrm{TIAM}} s$ an initial TIAM run. Then s is T-exhaustible.*

Proof. By induction on k. For $k = 0$ there is nothing to prove because the initial state $s_0 = \vdash t : \langle A\rangle_\uparrow$ has no judgment nor type tests. Then suppose $\sigma' : s_0 \to^{k-1}_{\mathrm{TIAM}} s'$ and that the run continues with $s' \to_{\mathrm{TIAM}} s$. By *i.h.*, s' is T-exhaustible.

Terminology: when a test state satisfies the clause in the definition of T-exhaustible states we say that it is *positive*.

Cases of $s' \to_{\mathrm{TIAM}} s$:

- Case $\to_{\bullet 1}$.

$$s' = \dfrac{\vdash t : T \to A \quad \vdash}{\vdash t\,u : \mathbb{A}\langle\star_\uparrow\rangle(= A)} \to_{\bullet 1} \dfrac{\vdash t : T \to \mathbb{A}\langle\star_\uparrow\rangle \quad \vdash}{\vdash t\,u : A} = s$$

– *judgment tests.* Note that s has the same judgment tests of s', which are positive by the *i.h.*

– *Type tests.* We first consider the type tests of direction $\uparrow$. Let us s_f be one of them. We observe that there is a corresponding type test ${s'}_f$ of s', that by *i.h.* it is positive, and that ${s'}_f \to_{\text{TIAM}} s_f$. Since the machine is deterministic also s_f is positive. Let us now consider a type test s_f of direction $\downarrow$. We observe that there is a corresponding type test ${s'}_f$ of s', that it is positive by *i.h.*, and that $s_f \to {s'}_f$. Then s_f is positive.

- Case $\to_{\bullet 2}$. Identical to the previous one.
- Case $\to_{\mathsf{var}}$.

$$s' = \dfrac{\dfrac{}{\vdash x : \mathbb{A}\langle\star_\uparrow\rangle_i (= A_i)}\, i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle A_i\rangle \to B}{} \quad \to_{\mathsf{var}} \quad \dfrac{\dfrac{}{\vdash x : A_i}\, i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle \mathbb{A}\langle\star_\downarrow\rangle_i\rangle \to B}{} = s$$

  – *judgment tests.* judgment tests of s are a subset of judgment tests of s' and thus positive by *i.h.*

  – *Type tests.* Let n be the level of $\mathbb{A}$. Let s^j be the type test of s associated to the j-th triple in $\mathsf{DiPref}(\mathbb{L}\langle\mathbb{A}_i\rangle \to B)$. Three cases, depending on the index j of s^j:

    1. $j = 1$: then s^1 is $\vdash \lambda x.C\langle x\rangle : \mathbb{A}\langle\star\rangle_{i\uparrow}, \mathbb{L} \to B$. Note that $s^1 \to_{\mathsf{bt2}} \vdash x : \mathbb{A}\langle\star\rangle_{i\downarrow}, \langle\cdot\rangle$, which has no type tests and has the same judgment tests of s', which by *i.h.* are positive. Hence, s^1 is T-exhaustible.
    2. *j is even*: for s^j (of direction $\downarrow$) there is a corresponding type test ${s'}^{j-1}$ of odd index of s', having direction $\uparrow$ and such that ${s'}^{j-1} \to_{\mathsf{var}} s^j$. Thus one can conclude by *i.h.* and determinism of the TIAM.
    3. *$j \neq 1$ is odd*: for s^j (of direction $\uparrow$) there is a corresponding type test ${s'}^{j-1}$ of even index of s', having direction $\downarrow$ and such that $s^j \to_{\mathsf{bt2}} {s'}^{j-1}$. Thus one can conclude by *i.h.*

- Case $\to_{\mathsf{bt2}}$.

$$s' = \dfrac{\dfrac{}{\vdash x : A_i (= \mathbb{A}\langle\star\rangle_i)}\, i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle \mathbb{A}\langle\star_\uparrow\rangle_i\rangle \to B}{} \quad \to_{\mathsf{bt2}} \quad \dfrac{\dfrac{}{\vdash x : \mathbb{A}\langle\star_\downarrow\rangle_i}\, i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle A_i\rangle \to B}{} = s$$

  – *judgment tests.* The first type test of s' is ${s'}^1 :=\vdash \lambda x.C\langle x\rangle : \mathbb{A}\langle\star\rangle_{i\uparrow}, \mathbb{L} \to B$. Note that ${s'}^1 \to_{\mathsf{bt2}} \vdash x : \mathbb{A}\langle\star\rangle_{i\downarrow}, \langle\cdot\rangle =: s''$ and that s'' exhausts ${s'}^1$, and it is the first such state. Since ${s'}^1$ is positive, s'' is T-exhaustible. Note that s'' has the same judgment tests of s, which are then positive.

  – *Type tests.* For each odd type test s^i of s (whose direction is $\uparrow$), the corresponding even type test ${s'}^{i+1}$ of s' has direction $\downarrow$, is positive by *i.h.*, and such that $s^i \to_{\mathsf{var}} {s'}^{i+1}$. Then s^i is positive. For each even type test s^i of s (whose direction is $\downarrow$), the corresponding odd type test ${s'}^{i+1}$ of s' has direction $\uparrow$, is positive by *i.h.*, and such that ${s'}^{i+1} \to_{\mathsf{bt2}} s^i$. Then s^i is positive by determinism of the TIAM.

- Cases $\to_{\bullet 3}$ and $\to_{\bullet 4}$. They are identical to case $\to_{\bullet 1}$.
- Case $\to_{\mathsf{arg}}$.

$$s' = \dfrac{\vdash t : \mathbb{L}\langle\mathbb{A}\langle\star_\downarrow\rangle_i\rangle \to A \qquad \dfrac{.. \vdash u : \mathbb{A}\langle\star\rangle_i ..}{\vdash u : T}\ \text{T-M}}{\vdash tu : A} \quad \to_{\mathsf{arg}} \quad \dfrac{\vdash t : T \to A \qquad \dfrac{.. \vdash u : \mathbb{A}\langle\star_\uparrow\rangle_i ..}{\vdash u : \mathbb{L}\langle\mathbb{A}\langle\star\rangle_i\rangle}\ \text{T-M}}{\vdash tu : A} = s$$

  - *judgment tests.* judgment tests of s are those of s', which are positive by *i.h.*, plus $s^u :=\vdash u : \mathbb{A}\langle\star\rangle_{i\downarrow}, \langle\cdot\rangle$. Please note that $s^u \to_{\mathsf{bt1}} \vdash t : \mathbb{A}\langle\star\rangle_{i\uparrow}, \mathbb{L} \to A =: s'^t$. Now, s'^t is a type test of s' and by *i.h.* is positive. Then s^u is positive.
  - *Type tests.* For each odd type test s^i of s (whose direction is $\uparrow$), the corresponding even type test s'^{i+1} of s' has direction $\downarrow$, is positive by *i.h.*, and such that $s'^{i+1} \to_{\mathsf{arg}} s^i$. Then s^i is positive by determinism of the TIAM. For each even type test s^i of s' (whose direction is $\downarrow$), the corresponding odd type test s'^{i+1} of s' has direction $\uparrow$, is positive by *i.h.*, and such that $s^i \to_{\mathsf{bt1}} s'^{i+1}$. Then s^i is positive.

- Case $\to_{\mathsf{bt1}}$.

$$s' = \dfrac{\vdash t : T \to B \qquad \dfrac{.. \vdash u : \mathbb{A}\langle\star_{\downarrow}\rangle_i ..}{\vdash u : \mathbb{L}\langle\mathbb{A}\langle\star\rangle_i\rangle}\ \text{T-M}}{\vdash tu : B} \quad \to_{\mathsf{bt1}} \quad \dfrac{\vdash t : \mathbb{L}\langle\mathbb{A}\langle\star_{\uparrow}\rangle_i\rangle \to B \qquad \dfrac{.. \vdash u : \mathbb{A}\langle\star\rangle_i ..}{\vdash u : T}\ \text{T-M}}{\vdash tu : B} = s$$

  - *judgment tests.* All judgment tests of s are judgment test of s', which are this way positive by *i.h.*
  - *Type tests.* The first type test of s is $s^1 :=\vdash t : \mathbb{A}\langle\star\rangle_{i\uparrow}, [\ldots\langle\cdot\rangle\ldots] \to A$. Please note that $s'^u :=\vdash u : \mathbb{A}\langle\star\rangle_{i\downarrow}, \langle\cdot\rangle$ is a judgment test of s' such that $s'^u \to_{\mathsf{bt1}} s^1$. By *i.h.*, s'^u is positive. By determinism of the TIAM, s^1 is positive.
    For each odd type test s^i of s (whose direction is $\uparrow$), the corresponding even type test s'^{i-1} of s' has direction $\downarrow$, is positive by *i.h.*, and such that $s'^{i-1} \to_{\mathsf{bt1}} s^i$. Then s^i is positive by determinism of the TIAM. For each even type test s^i of s' (whose direction is $\downarrow$), the corresponding odd type test s'^{i-1} of s' has direction $\uparrow$, is positive by *i.h.*, and such that $s^i \to_{\mathsf{arg}} s'^{i-1}$. Then s^i is positive.

□

Extracting λIAM States from TIAM T-Exhaustible States, and the λIAM/TIAM Strong Bisimulation

From T-exhaustible states one is able to *extract* λIAM states, as the following definition shows. Please note that the definition is well-founded, precisely because the objects are T-exhaustible states. Indeed, the induction principle used to define T-exhaustability allows recursive definition on T-exhaustible states to be well-behaved.

Definition B.3.7 (Extraction of logged positions)**.** *Let s be an T-exhaustible TIAM state in a derivation π, t be the final term in π, and s_f be a judgment or type test of s. Since s is T-exhaustible, there is an exhausting run $s_f \to^+_{\mathrm{TIAM}} s' \in \mathcal{E}_T$. Let x be the variable of s'. Then the logged position extracted from s_f is $l_{\mathsf{ext}}(s_f) := (x, \lambda x.D_n, l_{\mathsf{ext}}(s'^1) \cdot \ldots \cdot l_{\mathsf{ext}}(s'^n))$, where D_n is the context (of level n) retrieved traversing π from s' to the binder of λx of x in t and s'^i is the i-th judgment test of s'.*

Definition B.3.8 (Extraction of logs, tapes, and states)**.** *Let $s = (\pi, J, (A, \mathbb{A}), d)$ be an T-exhaustible TIAM state where t is the final term in π, and J is $\Gamma \vdash u : \mathbb{A}\langle A\rangle$. The λIAM state extracted from s is $s_{\mathsf{ext}}(s) := (u, C_s, L_{\mathsf{ext}}(s), T_{\mathsf{ext}}(s), d)$[1] where*

- Context*: C_s is the only term context such that $t = C_s\langle u\rangle$;*
- Log*: $L_{\mathsf{ext}}(s) := l_1 \cdots l_i \cdots l_n$ where $l_i = l_{\mathsf{ext}}(s^i_f)$ where s^i_f is the i-th judgment test of s.*
- Tape*: $T_{\mathsf{ext}}(s) = T^s_{\mathsf{ext}}(\mathbb{A}, 0)$ where $T^s_{\mathsf{ext}}(\mathbb{A}, i)$ is the auxiliary function defined by induction on $\mathbb{A}$ as follows.*

$$\begin{array}{lcl} T^s_{\mathsf{ext}}(\langle\cdot\rangle, i) & := & \epsilon \\ T^s_{\mathsf{ext}}(T \to \mathbb{A}, i) & := & \bullet \cdot T^s_{\mathsf{ext}}(\mathbb{A}, i) \\ T^s_{\mathsf{ext}}(\mathbb{L}\langle\mathbb{A}\rangle \to B, i) & := & l_{\mathsf{ext}}(s^i_f) \cdot T^s_{\mathsf{ext}}(\mathbb{A}, i+1) \end{array}$$

[1] We leave the color of d unchanged, in the sense that $s_{\mathsf{ext}}(s)$ is red/blue if s is red/blue, *i.e.* $\downarrow$ becomes $\uparrow$ and $\uparrow$ becomes $\downarrow$.

where s_f^i is the i-th type test of s.

We use $\simeq_{\mathsf{ext}}$ for the extraction relation between T-exhaustible TIAM states and λIAM states defined as $(s, s_{\mathsf{ext}}(s)) \in \simeq_{\mathsf{ext}}$.

First of all, we show that the extracted state respects the λIAM invariant about the length of the log.

Lemma B.3.9. *Let s be an T-exhaustible TIAM state and $s_{\mathsf{ext}}(s) = (t, C_s, L_{\mathsf{ext}}(s), T_{\mathsf{ext}}(s), d)$ the λIAM state extracted from it. Then the level of C_s is exactly the length of $L_{\mathsf{ext}}(s)$, that is, $(t, C_s, L_{\mathsf{ext}}(s))$ is a logged position.*

Proof. The length of $L_{\mathsf{ext}}(s)$ is the number of judgment tests of s, which is the number of T-MANY rule trees, and thus of T-@ rules, traversed descending from the focused judgment J of s to the final judgment of π. The level of C_s is the number of arguments in which the hole of C_s is contained, which are exactly the number of T-@ rules traversed descending from J to the final judgment of π. □

Proposition B.3.10 (TIAM-λIAM bisimulation)**.** *Let t a closed and $\to_{wh}$-normalizable term, and $\pi \triangleright \vdash t : \star$ a type derivation. Then $\simeq_{\mathsf{ext}}$ is a strong bisimulation between T-exhaustible TIAM states on π and λIAM states on t. Moreover, if $s_\pi \simeq_{\mathsf{ext}} s_\lambda$ then s_π is TIAM reachable if and only if s_λ is λIAM reachable.*

Proof. Assuming the bisimulation part of the statement, the moreover part follows from a trivial induction on the length of the initial run, since initial state are bisimilar and the bisimulation is exactly the fact that $\simeq_{\mathsf{ext}}$ is stable by transitions.

For the bisimulation part, we consider each possible transitions. We focus on the half of the proof showing that TIAM transitions are simulated by the λIAM, the other half is essentially identical.

- Case $\to_{\bullet 1}$.

$$s' = \dfrac{\vdash t : T \to A \quad \vdash}{\vdash tu : \mathbb{A}\langle\star_\uparrow\rangle(= A)} \quad \to_{\bullet 1} \quad \dfrac{\vdash t : T \to \mathbb{A}\langle\star_\uparrow\rangle \quad \vdash}{\vdash tu : A} = s$$

$$\simeq_{\mathsf{ext}}$$

$$s_{\mathsf{ext}}(s') = (\underline{tu}, C_{s'}, L_{\mathsf{ext}}(s'), T_{\mathsf{ext}}(s')) \quad \to_{\bullet 1} \quad (\underline{t}, C\langle\langle\cdot\rangle r\rangle, L_{\mathsf{ext}}(s'), \bullet \cdot T_{\mathsf{ext}}(s')) = s_\lambda$$

Note that $C_s = C_{s'}\langle\langle\cdot\rangle r\rangle$, $L_{\mathsf{ext}}(s) = L_{\mathsf{ext}}(s')$, and $T_{\mathsf{ext}}(s) = \bullet \cdot T_{\mathsf{ext}}(s')$. Then, $s_\lambda = s_{\mathsf{ext}}(s)$, that is, $s \simeq_{\mathsf{ext}} s_\lambda$.

- Case $\to_{\bullet 2}$. Identical to the previous one.

- Case $\to_{\mathsf{var}}$.

$$s' = \dfrac{\dfrac{}{\vdash x : \mathbb{A}\langle\star_\uparrow\rangle_i(= A_i)}\, i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle A_i\rangle \to B} \quad \to_{\mathsf{var}} \quad \dfrac{\dfrac{}{\vdash x : A_i}\, i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle\mathbb{A}\langle\star_\downarrow\rangle_i\rangle \to B} = s$$

$$\simeq_{\mathsf{ext}}$$

$$s_{\mathsf{ext}}(s') = (\underline{x}, \underbrace{C\langle\lambda x.D_n\rangle}_{=C_{s'}}, \underbrace{L_n \cdot L}_{=L_{\mathsf{ext}}(s')}, T_{\mathsf{ext}}(s')) \quad \to_{\mathsf{var}} \quad (\lambda x.D_n\langle x\rangle, \underline{C}, L, (x, \lambda x.D_n, L_n) \cdot T_{\mathsf{ext}}(s')) = s_\lambda$$

First of all, $C_{s'}$ has shape $C\langle\lambda x.D_n\rangle$ for some n, as the descending path from the focused judgment to the final judgment passes through the showed T-λ rule. Then $C_s = C$.

About the log, by Lemma 4.3.11 there is a correspondence between the level of term contexts and the length of the extracted log, so that $L_{\mathsf{ext}}(s')$ is at least of length n, that is, $L_{\mathsf{ext}}(s') = L_n \cdot L$, and $L_{\mathsf{ext}}(s) = L$.

About the tape, note that $T_{\mathsf{ext}}(s) = l_{\mathsf{ext}}(s^1_f) \cdot T^s_{\mathsf{ext}}(\mathbb{A}, 1)$ where s^1_f is the first type test of s. To show that $s_{\mathsf{ext}}(s) = (x, \lambda x.D_n, L_n) \cdot T_{\mathsf{ext}}(s')$ we have to show two things:

1. $l_{\mathsf{ext}}(s^1_f) = (x, \lambda x.D_n, L_n)$. Note that s^1_f is $\vdash \lambda x.C\langle x\rangle : \mathbb{A}\langle\star\rangle_{i\uparrow}, \mathbb{L} \to B$. Note that $s^1_f \to_{\mathsf{bt2}} \vdash x : \mathbb{A}\langle\star\rangle_{i\downarrow}, \langle\cdot\rangle = s''$, where s'' focusses on the same judgment of s', and that s'' is the state that T-exhausts s^1_f. By definition of extraction, $l_{\mathsf{ext}}(s^1_f) = (x, \lambda x.D_n, L_n)$.
2. $T^s_{\mathsf{ext}}(\mathbb{A}, 1) = T_{\mathsf{ext}}(s')$, that is, $T^s_{\mathsf{ext}}(\mathbb{A}, 1) = T^{s'}_{\mathsf{ext}}(\mathbb{A}, 0)$. Note that $T^s_{\mathsf{ext}}(\mathbb{A}, 1)$ and $T^{s'}_{\mathsf{ext}}(\mathbb{A}, 0)$ may differ only in the content of logged positions (obtained by extracting from tape tests), which is the only thing that depends on the direction and the state, the rest being uniquely determined by the type context $\mathbb{A}$. Here one has to repeat the reasoning done in the $\to_{\mathsf{bt2}}$ case of the proof of the T-exhaustible invariant (Lemma 4.3.8), that shows that the tape test of index $i > 1$ for s and the one of index $i - 1$ of s' exhaust on the same state, and thus induce the same logged position. Then $T^s_{\mathsf{ext}}(\mathbb{A}, 1) = T_{\mathsf{ext}}(s')$.

Then $s_{\mathsf{ext}}(s) = (x, \lambda x.D_n, L_n) \cdot T_{\mathsf{ext}}(s')$, and so $s_\lambda = s_{\mathsf{ext}}(s)$, that is, $s \simeq_{\mathsf{ext}} s_\lambda$.

- Case $\to_{\mathsf{bt2}}$.

$$s' = \dfrac{\dfrac{}{\vdash x : A_i (= \mathbb{A}\langle\star\rangle_i)}\, i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle\mathbb{A}\langle\star_\uparrow\rangle_i\rangle \to B}$$

$$\simeq_{\mathsf{ext}}$$

$$s_{\mathsf{ext}}(s') = (\underline{\lambda x.D_n\langle x\rangle}, C_{s'}, L_{\mathsf{ext}}(s'), \underbrace{(x, \lambda x.D_n, L_n) \cdot T^{s'}_{\mathsf{ext}}(\mathbb{A}, 1)}_{= T_{\mathsf{ext}}(s')})$$

$$\to_{\mathsf{bt2}} \quad (x, \underline{C_{s'}\langle\lambda x.D_n\rangle}, L_n \cdot L_{\mathsf{ext}}(s'), T^{s'}_{\mathsf{ext}}(\mathbb{A}, 1)) = s'_\lambda$$

$$\to_{\mathsf{bt2}} \quad \dfrac{\dfrac{}{\vdash x : \mathbb{A}\langle\star_\downarrow\rangle_i}\, i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle A_i\rangle \to B = s}$$

About the tape of $s_{\mathsf{ext}}(s')$, note that $T_{\mathsf{ext}}(s') = l_{\mathsf{ext}}(s'^1_f) \cdot T^{s'}_{\mathsf{ext}}(\mathbb{A}, 1)$ where s^1_f is the first type test of s'. We have to show that s^1_f exhausts on x, so that $l_{\mathsf{ext}}(s^1_f) = (x, \lambda x.D_n, L_n)$ for some L_n. Note that s^1_f is $\vdash \lambda x.C\langle x\rangle : \mathbb{A}\langle\star\rangle_{i\uparrow}, \mathbb{L} \to B$. Note that $s^1_f \to_{\mathsf{bt2}} \vdash x : \mathbb{A}\langle\star\rangle_{i\downarrow}, \langle\cdot\rangle = s''$, where s'' focusses on the same judgment of s, and that s'' is the state that S-exhausts s^1_f. By definition of extraction, $l_{\mathsf{ext}}(s^1_f) = (x, \lambda x.D_n, L_n)$ where L_n is the extraction of the first n judgment tests of s. Then $C_s = C_{s'}\langle\lambda x.D_n\rangle$ and $L_{\mathsf{ext}}(s) = L_n \cdot L_{\mathsf{ext}}(s')$.

About the tape, for s we have to prove that $T^{s'}_{\mathsf{ext}}(\mathbb{A}, 1) = T_{\mathsf{ext}}(s) = T^s_{\mathsf{ext}}(\mathbb{A}, 0)$. This is done as for $\to_{\mathsf{var}}$, mimicking the reasoning in the proof of the T-exhaustible invariant (Lemma 4.3.8).

Then, $s_\lambda = s_{\mathsf{ext}}(s)$, that is, $s \simeq_{\mathsf{ext}} s_\lambda$.

- Cases $\to_{\bullet 3}$ and $\to_{\bullet 4}$. They are identical to case $\to_{\bullet 1}$.
- Case $\to_{\mathsf{arg}}$.

$$s' = \dfrac{\vdash t : \mathbb{L}\langle\mathbb{A}\langle\star_\downarrow\rangle_i\rangle \to A \qquad \dfrac{.. \vdash u : \mathbb{A}\langle\star\rangle_i ..}{\vdash u : T}\ \text{T-M}}{\vdash tu : A}$$

$$\simeq_{\mathsf{ext}}$$

$$s_{\mathsf{ext}}(s') = (t, \underbrace{\underline{D\langle\langle\cdot\rangle u\rangle}}_{\underline{= C_{s'}}}, L_{\mathsf{ext}}(s'), \underbrace{l_{\mathsf{ext}}(s'^1_f) \cdot T^{s'}_{\mathsf{ext}}(\mathbb{A}, 1)}_{= T_{\mathsf{ext}}(s')})$$

$$\to_{\mathsf{arg}} \quad (\underline{u}, D\langle t\langle\cdot\rangle\rangle, l_{\mathsf{ext}}(s'^1_f) \cdot L_{\mathsf{ext}}(s'), T^{s'}_{\mathsf{ext}}(\mathbb{A}, 1)) = s'_\lambda$$

$$\to_{\mathsf{arg}} \quad \dfrac{\vdash t : T \to A \qquad \dfrac{.. \vdash u : \mathbb{A}\langle\star_\uparrow\rangle_i ..}{\vdash u : \mathbb{L}\langle\mathbb{A}\langle\star\rangle_i\rangle}\ \text{T-M}}{\vdash tu : A} = s$$

where s'^1_f is the first type test of s'. Obviously, $C_s = D\langle t\langle\cdot\rangle\rangle$. For the log we have to show that $L_{\mathsf{ext}}(s)$ is equal to $l_{\mathsf{ext}}(s'^1_f) \cdot L_{\mathsf{ext}}(s')$, which amounts to show that the first judgment test s^1 of s exhausts on the same state as the first tape test s'^1_f of s'. This is exactly the reasoning done in the proof of the T-exhaustible invariant. Similarly, one obtains that $T^{s'}_{\mathsf{ext}}(\mathbb{A}, 1) = T_{\mathsf{ext}}(s) = T^{s}_{\mathsf{ext}}(\mathbb{A}, 0)$.

- Case $\to_{\mathsf{bt1}}$.

$$s' = \dfrac{\vdash t : T \to A \qquad \dfrac{\dots \vdash u : \mathbb{A}\langle\star_{\downarrow}\rangle_i \dots}{\vdash u : \mathbb{L}\langle\mathbb{A}\langle\star\rangle_i\rangle(= T)}\ \text{T-M}}{\vdash tu : A} \quad \to_{\mathsf{bt1}} \quad \dfrac{\vdash t : \mathbb{L}\langle\mathbb{A}\langle\star_{\uparrow}\rangle_i\rangle \to A \qquad \dfrac{\dots \vdash u : \mathbb{A}\langle\star\rangle_i \dots}{\vdash u : T}\ \text{T-M}}{\vdash tu : A} = s$$

$$\simeq_{\mathsf{ext}}$$

$$s_{\mathsf{ext}}(s') = (u, \underbrace{D\langle t\langle\cdot\rangle\rangle}_{=C_{s'}}, \underbrace{l_{\mathsf{ext}}(s^1_f) \cdot L}_{=L_{\mathsf{ext}}(s')}, T_{\mathsf{ext}}(s')) \quad \to_{\mathsf{bt1}} \quad (t, D\langle\langle\cdot\rangle u\rangle, L, l_{\mathsf{ext}}(s^1_f) \cdot T_{\mathsf{ext}}(s')) = s'_{\lambda}$$

where s'^1_f is the first judgment test of s'. Obviously, $C_s = D\langle\langle\cdot\rangle u\rangle$. For the log, there is nothing to prove. For the tape, we have to show that $T_{\mathsf{ext}}(s)$ is equal to $l_{\mathsf{ext}}(s^1_f) \cdot T_{\mathsf{ext}}(s')$, which amounts to show two things. First, that the first tape test s^1 of s exhausts on the same state as the first judgment test s'^1_f of s'. Second, that $T^{s}_{\mathsf{ext}}(\mathbb{A}, 1) = T_{\mathsf{ext}}(s') = T^{s'}_{\mathsf{ext}}(\mathbb{A}, 0)$. Both points follow exactly the reasoning done in the proof of the T-exhaustible invariant.

□

The *moreover* part of the above statement hints at a bijection between *all* the states in π and reachable λIAM states. However, there still could be the possibility that some of the states in π are not reachable. This is actually *not* the case.

B.3.1 The TIAM is acyclic

In order to prove that the TIAM is acyclic, we need to show that if $t \to_{wh} u$, then cycles are preserved between the tree type derivation π for t and the sequence type derivation π' for u. One way to show this fact is building a (non-)termination-preserving bisimulation between states of π and states of π'.

Explaining the Bisimulation. Let us give an intuitive explanation of the improvement $\blacktriangleright$ that we are going to build next. Given two type derivations $\pi \triangleright \vdash H\langle(\lambda x.r)w\rangle : \star$ and $\pi' \triangleright \vdash H\langle r\{x{\leftarrow}w\}\rangle : \star$, it is possible to define a relation $\blacktriangleright$ between states of the former and of the latter. The key points are:

1. each axiom for x in π is $\blacktriangleright$-related with the judgment for the argument w that replaces it in π'.
2. Both the judgment for r and the one for $(\lambda x.r)w$ are $\blacktriangleright$-related to $r\{x{\leftarrow}w\}$.
3. The judgment for $\lambda x.r$ is not $\blacktriangleright$-related to any judgment of π'.

Defining $\blacktriangleright$. In order to define $\blacktriangleright$ formally, we enrich each type judgment (occurrence) $\vdash t : \mathbb{A}\langle\star\rangle$ with a context C such that $C\langle t\rangle$ is the term in the final judgment of the derivation π, obtaining $\vdash (t, C) : \mathbb{A}\langle\star\rangle$.

Definition B.3.11 (Bisimulation $\blacktriangleright$). *The definition of $\blacktriangleright$ for $\vdash (t, C) : \mathbb{A}\langle\star\rangle$ has 4 clauses:*

- rdx*: the redex is in t, that is, $t = H\langle(\lambda x.u)r\rangle$, and so C is a head context K:*

$$\vdash (H\langle(\lambda x.u)r\rangle, K) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\mathsf{rdx}} \vdash (H\langle u\{x{\leftarrow}r\}\rangle, K) : \mathbb{A}\langle\star\rangle$$

- body: *the term t is part of the body of the abstraction involved in the redex:*

$$\vdash (t, H\langle(\lambda x.D)u\rangle) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\mathsf{body}} \vdash (t\{x{\leftarrow}u\}, H\langle D\{x{\leftarrow}u\}\rangle) : \mathbb{A}\langle\star\rangle$$

- arg: *the term t is part of the argument of the redex:*

$$\vdash (t, H\langle(\lambda x.D\langle x\rangle)E\rangle) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\mathsf{arg}} \vdash (t, H\langle D\{x{\leftarrow}E\langle t\rangle\}\langle E\rangle\rangle) : \mathbb{A}\langle\star\rangle$$

- ext: *The term t is disjoint form the redex, that then takes place only in C:*

$$\vdash (t, K\langle H\langle(\lambda x.r)u\rangle D\rangle) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\mathsf{ext}} \vdash (t, K\langle H\langle r\{x{\leftarrow}u\}\rangle D\rangle) : \mathbb{A}\langle\star\rangle$$

Please note that the only states of π which are not mapped to any state of π' are those relative to the judgment $\vdash \lambda x.r : [G'_1 ... G'_n] \to A$.

Proposition B.3.12. $\blacktriangleright$ *is an improvement between TIAM states.*

Proof. [2] We inspect the 4 cases of the definition of $\blacktriangleright$.

- Rule rdx :$\vdash (H\langle(\lambda x.t)u\rangle, K) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\mathsf{rdx}} \vdash (H\langle t\{x{\leftarrow}u\}\rangle, K) : \mathbb{A}\langle\star\rangle$. Cases for $\uparrow$ (by cases of H):
  - $H = \langle\cdot\rangle$. The diagram is closed by rule body:

$$\begin{array}{ccccc} \vdash ((\lambda x.t)u, K) : \mathbb{A}\langle\star\rangle & \to_{\mathrm{TIAM}} & \vdash (\lambda x.t, K\langle\langle\cdot\rangle u\rangle) : T \to \mathbb{A}\langle\star\rangle & \to_{\mathrm{TIAM}} & \vdash (t, K\langle(\lambda x.\langle\cdot\rangle)u\rangle) : \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\mathsf{rdx}} & & & & \blacktriangleright_{\mathsf{body}} \\ \vdash (t\{x{\leftarrow}u\}, K) : \mathbb{A}\langle\star\rangle & & = & & \vdash (t\{x{\leftarrow}u\}, K) : \mathbb{A}\langle\star\rangle \end{array}$$

  - $H = Gs$. The diagram is closed by rule $\blacktriangleright_{\mathsf{rdx}}$:

$$\begin{array}{ccc} \vdash (G\langle r\rangle s, K) : \mathbb{A}\langle\star\rangle & \to_{\mathrm{TIAM}} & \vdash (G\langle r\rangle, K\langle\langle\cdot\rangle s\rangle) : T \to \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\mathsf{rdx}} & & \blacktriangleright_{\mathsf{rdx}} \\ \vdash (G\langle w\rangle s, K) : \mathbb{A}\langle\star\rangle & \to_{\mathrm{TIAM}} & \vdash (G\langle w\rangle, K\langle\langle\cdot\rangle s\rangle) : T \to \mathbb{A}\langle\star\rangle \end{array}$$

  Cases for $\downarrow$ (by cases of K):
  - $K = \langle\cdot\rangle$. Both machines are stuck.

$$\begin{array}{c} \vdash (r, \langle\cdot\rangle) : \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\mathsf{rdx}} \\ \vdash (w, \langle\cdot\rangle) : \mathbb{A}\langle\star\rangle \end{array}$$

  - $K = G\langle\langle\cdot\rangle s\rangle$. Two subcases depending on the type context. If the focus is on the right of the arrow the diagram is closed by rule rdx.

$$\begin{array}{ccc} \vdash (r, G\langle\langle\cdot\rangle s\rangle) : T \to \mathbb{A}\langle\star\rangle & \to_{\mathrm{TIAM}} & \vdash (rs, G) : \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\mathsf{rdx}} & & \blacktriangleright_{\mathsf{rdx}} \\ \vdash (w, G\langle\langle\cdot\rangle s\rangle) : T \to \mathbb{A}\langle\star\rangle & \to_{\mathrm{TIAM}} & \vdash (ws, G) : \mathbb{A}\langle\star\rangle \end{array}$$

    If the focus is on the left of the arrow the diagram is closed by rule ext.

$$\begin{array}{ccc} \vdash (r, G\langle\langle\cdot\rangle s\rangle) : \mathbb{L}\langle\mathbb{A}\langle\star\rangle\rangle \to A & \to_{\mathrm{TIAM}} & \vdash (s, G\langle r\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\mathsf{rdx}} & & \blacktriangleright_{\mathsf{ext}} \\ \vdash (w, G\langle\langle\cdot\rangle s\rangle) : \mathbb{L}\langle\mathbb{A}\langle\star\rangle\rangle \to A & \to_{\mathrm{TIAM}} & \vdash (s, G\langle w\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle \end{array}$$

[2] Also this proof requires colors.

- Rule body: $\vdash (t, H\langle(\lambda x.D)u\rangle) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\mathsf{body}} \vdash (t\{x{\leftarrow}u\}, H\langle D\{x{\leftarrow}u\}\rangle) : \mathbb{A}\langle\star\rangle$. Cases of $\uparrow$ (by cases of t):
  - $t = rw$. Trivially closed by rule body.
  - $t = \lambda y.r$. If $t : \star$ both machines are stuck. If $t : T \to A$, the diagram is trivially closed by rule body.
  - $t = x$. Diagram closed by rule arg.

$$\begin{array}{ccccc} \vdash (x, H\langle(\lambda x.D)u\rangle : \mathbb{A}\langle\star\rangle & \to & \vdash (\lambda x.D\langle x\rangle, H\langle\langle\cdot\rangle u\rangle) : \mathbb{L}\langle\mathbb{A}\langle\star\rangle\rangle \to B & \to & \vdash (u, H\langle(\lambda x.D\langle x\rangle)\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\mathsf{body}} & & & & \blacktriangleright_{\mathsf{arg}} \\ \vdash (u, H\langle D\{x{\leftarrow}u\}\rangle) : \mathbb{A}\langle\star\rangle & & = & & \vdash (u, H\langle D\{x{\leftarrow}u\}\rangle) : \mathbb{A}\langle\star\rangle \end{array}$$

  Cases of $\downarrow$ (by cases of D):
  - $D = \langle\cdot\rangle$. The diagram is closed by rule rdx

$$\begin{array}{ccccc} \vdash (t, H\langle(\lambda x.\langle\cdot\rangle)u\rangle) : \mathbb{A}\langle\star\rangle & \to_{\mathrm{TIAM}} & \vdash (\lambda x.t, H\langle\langle\cdot\rangle u\rangle) : T \to \mathbb{A}\langle\star\rangle & \to_{\mathrm{TIAM}} & \vdash ((\lambda x.t)u, H) : \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\mathsf{body}} & & & & \blacktriangleright_{\mathsf{rdx}} \\ \vdash (t\{x{\leftarrow}u\}, H) : \mathbb{A}\langle\star\rangle & & = & & \vdash (t\{x{\leftarrow}u\}, H) : \mathbb{A}\langle\star\rangle \end{array}$$

  - $D = E\langle\lambda y.\langle\cdot\rangle\rangle$, $D = E\langle\langle\cdot\rangle r\rangle$ and $D = E\langle r\langle\cdot\rangle\rangle$. The diagram is trivially closed by rule body.
- Rule $\blacktriangleright_{\mathsf{arg}}$: $\vdash (t, H\langle(\lambda x.D\langle x\rangle)E\rangle) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\mathsf{arg}} \vdash (t, H\langle D\{x{\leftarrow}E\langle t\rangle\}\langle E\rangle\rangle) : \mathbb{A}\langle\star\rangle$. Cases of $\uparrow$ (by cases of t) are all trivial: they are closed by rule $\blacktriangleright_{\mathsf{arg}}$ itself. The only non trivial case for $\downarrow$ (by cases of E) is when $E = \langle\cdot\rangle$.

$$\begin{array}{ccccc} \vdash (t, H\langle(\lambda x.D\langle x\rangle)\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle & \to & \vdash (\lambda x.D\langle x\rangle, H\langle\langle\cdot\rangle t\rangle) : \mathbb{L}\langle\mathbb{A}\langle\star\rangle\rangle \to B & \to & \vdash (x, H\langle(\lambda x.D)t\rangle : \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\mathsf{arg}} & & & & \blacktriangleright_{\mathsf{body}} \\ \vdash (t, H\langle D\{x{\leftarrow}t\}\rangle) : \mathbb{A}\langle\star\rangle & & = & & \vdash (t, H\langle D\{x{\leftarrow}t\}\rangle) : \mathbb{A}\langle\star\rangle \end{array}$$

- Rule $\blacktriangleright_{\mathsf{ext}}$: $\vdash (t, K\langle H\langle(\lambda x.r)u\rangle D\rangle) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\mathsf{ext}} \vdash (t, K\langle H\langle r\{x{\leftarrow}u\}\rangle D\rangle) : \mathbb{A}\langle\star\rangle$. Cases of $\uparrow$ (by cases of t) are all trivial: they are closed by rule $\blacktriangleright_{\mathsf{ext}}$ itself. The only non trivial case for $\downarrow$ (by cases of D) is when $D = \langle\cdot\rangle$. We put $s := H\langle(\lambda x.r)u\rangle$ and $w := H\langle r\{x{\leftarrow}u\}\rangle$.

$$\begin{array}{ccc} \vdash (t, K\langle s\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle & \to_{\mathrm{TIAM}} & \vdash (s, K\langle\langle\cdot\rangle t\rangle) : \mathbb{L}\langle\mathbb{A}\langle\star\rangle\rangle \to A \\ \blacktriangleright_{\mathsf{ext}} & & \blacktriangleright_{\mathsf{rdx}} \\ \vdash (t, K\langle w\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle & \to_{\mathrm{TIAM}} & \vdash (w, K\langle\langle\cdot\rangle t\rangle) : \mathbb{L}\langle\mathbb{A}\langle\star\rangle\rangle \to A \end{array}$$

□

Corollary B.3.13. *If $\pi \triangleright \vdash H\langle(\lambda x.r)w\rangle : \star$ contains a cycle, the also $\pi' \triangleright \vdash H\langle r\{x{\leftarrow}w\}\rangle : \star$ contains a cycle.*

Proof. If the run of the TIAM on $\pi \triangleright \vdash H\langle(\lambda x.r)w\rangle : \star$ loops then there exists a state s_π such that a computation starting from s_π diverges. Every state but $\vdash (\lambda x.r, H\langle\langle\cdot\rangle w\rangle) : \mathbb{A}\langle\star\rangle$, which however is not final, is related by $\blacktriangleright$ to a state $s_{\pi'}$ of $\mathsf{T}_{\pi'}$. Since improvements preserve non-termination (Prop. 3.6.2.1), also $s_{\pi'}$ diverges. Since $s_{\pi'}$ has a finite number of states, there must be a cycle. □

Corollary B.3.14. *For each type derivation $\pi \triangleright \vdash t : \star$, T_π has no cycles.*

Proof. Since t is typable, then it has normal form, call it u. Clearly the type derivation for u has no cycles. By the previous corollary, also π cannot have any of them. □

Proposition B.3.15. *Let t a closed term and $\pi \triangleright \vdash t : \star$ a tree type derivation. Then every state of π is reached exactly once.*

Proof. Immediately by Lemma 4.3.13. □

Appendix C

Correctness and Completeness of the Tree Type System

The size $|\pi|$ of a tree types derivation π is the number of its rules that are not $\mathrm{T}_{\mathrm{MANY}}$. It is the quantity that is used to prove the termination argument for typed terms.

C.1 Correctness

In order to prove that typability implies termination via a simple combinatorial argument, we need to refine the standard statements of the substitution lemma and of subject reduction with quantitative information.

The next lemma is used in the substitution lemma (namely, the implication from left to right), and shall also be used in the anti-substitution lemma (the converse implication).

Lemma C.1.1 (Tree splitting and merging). *Let $T = T_1 \uplus .. \uplus T_k$. Then there exists $\pi \triangleright [\Gamma] \vdash t : T$ if and only if there exist $\pi_i \triangleright [\Gamma_i] \vdash t : T_i$ for $i \in \{1, .., k\}$. Moreover, $[\Gamma] = [\Gamma_1 \uplus .. \uplus \Gamma_k]$ and $|\pi| = \sum_{i=1}^{n} |\pi_i|$.*

Proof. We prove the statement by first examining the rule T-MANY, which is the last rule used in π, as t is typed with a tree type.

$$\frac{\Gamma_i \vdash t : G_i \quad 1 \leq i \leq n}{[\uplus_{i=1}^{n} \Gamma_i] \vdash t : [G_1, .., G_n] = T_1 \uplus .. \uplus T_k} \text{ T-MANY}$$

We can prove the statement considering k derivations, each of them deriving the judgment $t : T_j$.

$$\frac{\Gamma_i \vdash t : G_i \quad 1 \leq i \leq |T_j|}{[\uplus_{i=1}^{|T_j|} \Gamma_i] \vdash t : [G_1, .., G_{|T_j|}] = T_j} \text{ T-MANY}$$

□

Lemma C.1.2 (Quantitative substitution). *Let $\pi_t \triangleright \Gamma, x : T \vdash t : G$ and $\pi_u \triangleright \vdash u : T$. Then there exists $\pi_{t\{x \leftarrow u\}} \triangleright \Gamma \vdash t\{x \leftarrow u\} : G$ such that $|\pi_{t\{x \leftarrow u\}}| = |\pi_t| + |\pi_u| - |T^{\ell}|$.*

Proof. By induction on the derivation π_t.

- *Rule* T-VAR. Two sub-cases:
  1. $t = x$: then $t\{x \leftarrow u\} = u$, $G = A$, and $T = [A]$ is a singleton. Then the hypothesis $\pi_u \triangleright \vdash u : T$ is necessarily obtained by applying a unary T-MANY rule to a derivation of the form $\pi'_u \triangleright \vdash u : A$. The typing derivation $\pi_{t\{x \leftarrow u\}} := \pi'_u$ satisfies the statement because $|\pi_{t\{x \leftarrow u\}}| = |\pi'_u| = 1 + |\pi'_u| - 1 = |\pi_t| + |\pi_u| - |T^{\ell}|$.
  2. $t = y$: then $t\{x \leftarrow u\} = y$ and $T = [\cdot]$. Then the hypothesis $\pi_u \triangleright \vdash u : T$ is necessarily obtained by applying a T-NONE rule, and $|\pi_u| = 0$. The typing derivation $\pi_{t\{x \leftarrow u\}} := \pi_t$ satisfies the statement because $|\pi_{t\{x \leftarrow u\}}| = |\pi_t| = |\pi_t| + 0 - 0 = |\pi_t| + |\pi_u| - |T^{\ell}|$.

- *Rule* T-$\lambda_\star$. Then $t = \lambda y.r$, $G = \star$, and $T = [\cdot]$. It goes as for the second variable case ($t = y$). Namely, the hypothesis $\pi_u \triangleright \vdash u : T$ is necessarily obtained by applying a T-NONE rule, and $|\pi_u| = 0$. The typing derivation $\pi_{t\{x\leftarrow u\}}$ is also a single T-$\lambda_\star$ rule, because $t\{x\leftarrow u\} = (\lambda y.r)\{x\leftarrow u\} = \lambda y.r\{x\leftarrow u\}$ is also an abstraction. Note that $\pi_{t\{x\leftarrow u\}}$ satisfies the statement because $|\pi_{t\{x\leftarrow u\}}| = |\pi_t| = |\pi_t| + 0 - 0 = |\pi_t| + |\pi_u| - |T^\ell|$.

- *Rule* T-λ. Then π_t has the following shape:

$$\dfrac{\begin{array}{c}\vdots\\ \pi_r \ \triangleright \Gamma, x : T, y : T' \vdash r : B\end{array}}{\Gamma, x : T \vdash \lambda y.r : T' \to B}\ \text{T-}\lambda$$

  with $t = \lambda y.r$ and $G = T' \to B$. By *i.h.*, there exists a derivation $\pi_{r\{x\leftarrow u\}} \triangleright \Gamma, y : T' \vdash r\{x\leftarrow u\} : B$ such that $|\pi_{r\{x\leftarrow u\}}| = |\pi_r| + |\pi_u| - |T^\ell|$. Applying back rule T-$\lambda$ we obtain $\pi_{(\lambda y.r)\{x\leftarrow u\}}$

$$\dfrac{\begin{array}{c}\vdots\\ \pi_{r\{x\leftarrow u\}} \ \triangleright \Gamma, y : T' \vdash r\{x\leftarrow u\} : B\end{array}}{\Gamma \vdash \lambda y.r\{x\leftarrow u\} : T' \to B}\ \text{T-}\lambda$$

  which satisfies $|\pi_{(\lambda y.r)\{x\leftarrow u\}}| = |\pi_{r\{x\leftarrow u\}}| + 1 =_{i.h.} |\pi_r| + |\pi_u| - |T^\ell| + 1 = |\pi_{\lambda y.r}| + |\pi_u| - |T^\ell|$.

- *Rule* T-@. Then π_t has the following shape:

$$\dfrac{\begin{array}{c}\vdots\\ \pi_r \ \triangleright x : T_1, \Gamma \vdash r : T' \to A\end{array} \quad \begin{array}{c}\vdots\\ \pi_w \ \triangleright x : T_2, \Delta \vdash w : T'\end{array}}{x : T_1 \uplus T_2, \Gamma \uplus \Delta \vdash rw : A}\ \text{T-@}$$

  with $t = rw$, $G = A$, and $T = T_1 \uplus T_2$. By Lemma C.1.1, the hypothesis $\pi_u \triangleright \vdash u : T$ splits into two derivations $\pi_u^1 \triangleright \vdash u : T_1$ and $\pi_u^2 \triangleright \vdash u : T_2$ such that $|\pi_u| = |\pi_u^1| + |\pi_u^2|$. By *i.h.*, there exist:

  1. $\pi_{r\{x\leftarrow u\}} \triangleright \Gamma \vdash r\{x\leftarrow u\} : T' \to A$ such that $|\pi_{r\{x\leftarrow u\}}| = |\pi_r| + |\pi_u^1| - |T_1^\ell|$, and
  2. $\pi_{w\{x\leftarrow u\}} \triangleright \Delta \vdash w\{x\leftarrow u\} : T'$ such that $|\pi_{w\{x\leftarrow u\}}| = |\pi_w| + |\pi_u^2| - |T_2^\ell|$.

  Note that $(rw)\{x\leftarrow u\} = r\{x\leftarrow u\}w\{x\leftarrow u\}$. Then the derivation $\pi_{(rw)\{x\leftarrow u\}}$ is defined as follows:

$$\dfrac{\pi_{r\{x\leftarrow u\}} \ \triangleright \Gamma \vdash r\{x\leftarrow u\} : T' \to A \qquad \pi_{w\{x\leftarrow u\}} \ \triangleright \Delta \vdash w\{x\leftarrow u\} : T'}{\Gamma \uplus \Delta \vdash r\{x\leftarrow u\}w\{x\leftarrow u\} : A}\ \text{T-@}$$

  for which

$$\begin{array}{rcl} |\pi_{(rw)\{x\leftarrow u\}}| & = & |\pi_{r\{x\leftarrow u\}}| + |\pi_{w\{x\leftarrow u\}}| + 1 \\ & =_{i.h.} & |\pi_r| + |\pi_u^1| - |T_1^\ell| + |\pi_w| + |\pi_u^2| - |T_2^\ell| + 1 \\ & = & (|\pi_r| + |\pi_w| + 1) + (|\pi_u^1| + |\pi_u^2|) - (|T_1^\ell| + |T_2^\ell|) \\ & = & |\pi_{rw}| + |\pi_u| - |T^\ell| \end{array}$$

- *Rule* T-NONE. Then $G = [\cdot]$, and $T = [\cdot]$. It goes as for the second variable case ($t = y$). Namely, the hypothesis $\pi_u \triangleright \vdash u : T$ is necessarily obtained by applying another T-NONE rule, and $|\pi_u| = 0$. The typing derivation $\pi_{t\{x\leftarrow u\}}$ is also a single T-NONE rule for the term $t\{x\leftarrow u\}$. Note that $\pi_{t\{x\leftarrow u\}}$ satisfies the statement because $|\pi_{t\{x\leftarrow u\}}| = |\pi_t| = |\pi_t| + 0 - 0 = |\pi_t| + |\pi_u| - |T^\ell|$.

- *Rule* T-MANY. Then π_t has the following shape:

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma_i, x : T_i \vdash t : G_i \quad 1 \leq i \leq n\end{array}}{[\uplus_{i=1}^n \Gamma_i], x : [\uplus_{i=1}^n T_i] \vdash t : [G_1, .., G_n]}\ \text{T-MANY}$$

By Lemma C.1.1, the hypothesis $\pi_u \triangleright \vdash u : T$ splits into n derivations $\pi_u^i \triangleright \vdash u : T_i$ with $i \in \{1, .., n\}$ such that $|\pi_u| = \sum_{i=1}^n |\pi_u^i|$. By *i.h.*, there exist n derivations $\pi^i_{t\{x\leftarrow u\}} \triangleright \Gamma_i \vdash t\{x\leftarrow u\} : G_i$ such that $|\pi_{t\{x\leftarrow u\}}|^i = |\pi_t|^i + |\pi_u^i| - |T_i^\ell|$. Then the derivation $\pi_{t\{x\leftarrow u\}}$ is defined as follows:

$$\dfrac{\begin{array}{c}\vdots\\ \pi^i_{t\{x\leftarrow u\}} \triangleright \Gamma_i \vdash t\{x\leftarrow u\} : G_i \quad 1 \leq i \leq n\end{array}}{[\uplus_{i=1}^n \Gamma_i] \vdash t\{x\leftarrow u\} : [G_1, .., G_n]}\ \text{T-MANY}$$

for which

$$\begin{array}{rcl} |\pi_{t\{x\leftarrow u\}}| & = & \sum_{i=1}^n |\pi_{t\{x\leftarrow u\}}|^i \\ & =_{i.h.} & \sum_{i=1}^n (|\pi_t|^i + |\pi_u^i| - |T_i^\ell|) \\ & = & \sum_{i=1}^n |\pi_t|^i + \sum_{i=1}^n |\pi_u^i| - \sum_{i=1}^n |T_i^\ell| \\ & = & |\pi_t| + |\pi_u| - |T^\ell| \end{array}$$

□

Proposition C.1.3 (Quantitative Subject Reduction). *If t is closed, $\pi \triangleright \vdash t : A$, and $t \to_{wh} u$ then there exists $\pi' \triangleright \vdash u : A$ such that $|\pi| > |\pi'|$.*

Proof. By induction on $t \to_{wh} u$.

- *Base case, step at top level*: $t = (\lambda x.r)w \to_{wh} r\{x\leftarrow w\} = u$. Note that w is closed because t is closed by hypothesis. Then π has the following shape.

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \pi_r \triangleright x : T \vdash r : A\end{array}}{\vdash \lambda x.r : T \to A}\ \text{T-}\lambda \qquad \begin{array}{c}\vdots\\ \pi_w \triangleright \vdash w : T\end{array}}{\vdash (\lambda x.r)w : A}\ \text{T-@}$$

We can apply the quantitative substitution lemma (Lemma C.1.2) to the sub-derivations π_r and π_w obtaining a derivation $\pi_{r\{x\leftarrow w\}} \triangleright \vdash r\{x\leftarrow u\} : A$ such that $|\pi_{r\{x\leftarrow w\}}| = \pi_r + \pi_w - |T^\ell| < \pi_r + \pi_w + 2 = |\pi|$.

- *Inductive case, step on the left of the root application*: $t = rw \to_{wh} r'w = u$ with $r \to_{wh} r'$. Then π has the following shape.

$$\dfrac{\begin{array}{c}\vdots\\ \pi_r \triangleright \vdash r : T \to A\end{array} \quad \begin{array}{c}\vdots\\ \pi_w \triangleright \vdash w : T\end{array}}{\vdash rw : A}\ \text{T-@}$$

Applying the *i.h.* to the left sub-derivation π_r, we obtain $\pi'_{r'} \triangleright \vdash r' : T \to A$ such that $|\pi_r| > |\pi'_{r'}|$. Then π' is defined as follows.

$$\pi' := \dfrac{\begin{array}{c}\vdots\\ \pi'_{r'} \triangleright \vdash r' : T \to A\end{array} \quad \begin{array}{c}\vdots\\ \pi_w \triangleright \vdash w : T\end{array}}{\vdash r'w : A}\ \text{T-@}$$

for which $|\pi| = |\pi_r| + 1 + |\pi_w| >_{i.h.} |\pi'_{r'}| + 1 + |\pi_w| = |\pi'|$, as required.

□

Theorem C.1.4 (Correctness of tree types for Closed CbN). *If $\pi \triangleright \vdash t : A$ then t is Closed CbN terminating.*

Proof. By induction on $|\pi|$ and case analysis of whether $t \to_{wh}$-reduces. Cases:

1. *t does not reduce.* Then it is $\to_{wh}$-normal.
2. *$t \to_{wh} u$ for some u.* By quantitative subject reduction (Prop. C.1.3), there exists $\pi' \triangleright \vdash u : A$ such that $|\pi| > |\pi'|$. Then we can apply the *i.h.* to π', obtaining that u is Closed CbN normalizing. Therefore, so is t.

□

C.2 Completeness

The completeness of the type system is easier to prove, because there is no need to develop the quantitative analysis, not having to show termination of a relation.

Lemma C.2.1 (Anti-substitution). *Let $\pi \triangleright \Gamma \vdash t\{x \leftarrow u\} : G$ with u closed. Then there exist*

- *a tree type T,*
- *a derivation $\pi_t \triangleright \Gamma, x : T \vdash t : G$, and*
- *a derivation $\pi_u \triangleright \vdash u : T$.*

Proof. By lexycographic induction on (t, G). We first deal with the case in which G is a tree type T. We look at the last rule of π. Cases:

- *Rule* T-NONE. Then $G = [\cdot]$. The statement holds with respect to $T := [\cdot]$, $\pi_t := \pi$ and π_u being another T-NONE rule of term u.
- *Rule* T-MANY. Then π has the following shape:

$$\dfrac{\begin{array}{c}\vdots\\ \pi^i \triangleright \Gamma_i \vdash t\{x \leftarrow u\} : G_i \quad 1 \leq i \leq n\end{array}}{[\uplus_{i=1}^n \Gamma_i] \vdash t\{x \leftarrow u\} : [G_1, .., G_n]}\ \text{T-MANY}$$

  By *i.h.* (2nd component), for $1 \leq i \leq n$ there exist T_i and derivations $\pi_t^i \triangleright \Gamma_i, x : T_i \vdash t : G_i$ and $\pi_u^i \triangleright \vdash u : T_i$. By Lemma C.1.1, the derivations π_u^i merge into a derivation $\pi_u \triangleright \vdash u : T$ where $T := T_1 \uplus .. \uplus T_n$. The derivation π_t is instead obtained as follows.

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma_i, x : T_i \vdash t : G_i \quad 1 \leq i \leq n\end{array}}{[\uplus_{i=1}^n \Gamma_i], x : [\uplus_{i=1}^n T_i] \vdash t : [G_1, .., G_n]}\ \text{T-MANY}$$

  Now, we assume G to be a linear type A, and look at the cases for the last rule of π.

- *Variable*. Two sub-cases:
  1. $t = x$: then $t\{x \leftarrow u\} = u$. The statements holds by taking
     - $T := [A]$
     - π_t as an axiom assigning type A to x, and
     - $\pi_u := \pi$.
  2. $t = y$: then $t\{x \leftarrow u\} = y$. The statements holds by taking
     - $T := [\cdot]$
     - π_t as an axiom assigning type A to y, and

– π_u as a T-NONE rule of term u.

- *Rule* T-$\lambda_\star$. Then $t = \lambda y.r$ and $G = \star$. The statements holds by taking
  - $T := [\cdot]$
  - π_t as a T-$\lambda_\star$ rule of term $\lambda y.r$, and
  - π_u as a T-NONE rule of term u.
- *Rule* T-λ. Then π has the following shape:

$$\dfrac{\begin{array}{c}\vdots\\ \pi_{r\{x\leftarrow u\}} \ \triangleright \Gamma, y : T' \vdash r\{x\leftarrow u\} : B\end{array}}{\Gamma \vdash \lambda y.r\{x\leftarrow u\} : T' \to B}\ \text{T-}\lambda$$

  with $t = \lambda y.r$ and $A = T' \to B$. By *i.h.* (1st component), there exists T and derivations $\pi_r \ \triangleright \Gamma, x : T, y : T' \vdash r : B$ and $\pi_u \triangleright \vdash u : T$. Applying back rule T-$\lambda$ to π_r, we obtain $\pi_{\lambda y.r}$ as follows.

$$\dfrac{\begin{array}{c}\vdots\\ \pi_r \ \triangleright \Gamma, x : T, y : T' \vdash r : B\end{array}}{\Gamma, x : T \vdash \lambda y.r : T' \to B}\ \text{T-}\lambda$$

- *Rule* T-@. Then π has the following shape:

$$\dfrac{\pi_{r\{x\leftarrow u\}} \ \triangleright \Gamma \vdash r\{x\leftarrow u\} : T' \to A \qquad \pi_{w\{x\leftarrow u\}} \ \triangleright \Delta \vdash w\{x\leftarrow u\} : T'}{\Gamma \uplus \Delta \vdash r\{x\leftarrow u\}w\{x\leftarrow u\} : A}\ \text{T-@}$$

  with $t = rw$ and $T = T_1 \uplus T_2$. By *i.h.* (1st component), there exist:

  1. a tree type T_1 and derivations $\pi_r \ \triangleright \Gamma, x : T_1 \vdash r : T' \to A$ and $\pi_u^1 \triangleright \vdash u : T_1$;
  2. a tree type T_2 and derivations $\pi_w \ \triangleright \Delta, x : T_2 \vdash w : T'$ and $\pi_u^2 \triangleright \vdash u : T_2$.

  By Lemma C.1.1, the derivations π_u^1 and π_u^2 merge into a derivation $\pi_u \ \triangleright \vdash u : T$ with $T := T_1 \uplus T_2$. The derivation π_t is instead obtained as follows.

$$\dfrac{\begin{array}{c}\vdots\\ \pi_r \ \triangleright x : T_1, \Gamma \vdash r : T' \to A\end{array} \qquad \begin{array}{c}\vdots\\ \pi_w \ \triangleright x : T_2, \Delta \vdash w : T'\end{array}}{x : \underbrace{T_1 \uplus T_2}_{T}, \Gamma \uplus \Delta \vdash rw : A}\ \text{T-@}$$

□

Proposition C.2.2 (Subject expansion)**.** *If t is closed, $\pi \ \triangleright \vdash u : A$, and $t \to_{wh} u$ then there exists $\pi' \ \triangleright \vdash t : A$.*

Proof. By induction on $t \to_{wh} u$.

- *Base case, step at top level*: $t = (\lambda x.r)w \to_{wh} r\{x\leftarrow w\} = u$. Note that w is closed because t is closed. The derivation in the hypothesis is $\pi \ \triangleright \vdash r\{x\leftarrow w\} : A$. Then by the anti-substitution lemma (Lemma C.2.1) we obtain a tree type T and two derivations $\pi_r \ \triangleright x : T \vdash r : A$ and $\pi_w \ \triangleright \vdash w : T$. The derivation π' of the statement is then defined as follows:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \pi_r \ \triangleright x : T \vdash r : A\end{array}}{\vdash \lambda x.r : T \to A}\ \text{T-}\lambda \qquad \begin{array}{c}\vdots\\ \pi_w \ \triangleright \vdash w : T\end{array}}{\vdash (\lambda x.r)w : A}\ \text{T-@}$$

- *Inductive case, step on the left of the root application*: $t = rw \to_{wh} r'w = u$ with $r \to_{wh} r'$. Then π has the following shape.

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \pi_{r'} \ \triangleright \vdash r' : T \to A & \pi_w \ \triangleright \vdash w : T\end{array}}{\vdash r'w : A}\ \text{T-@}$$

  Applying the *i.h.* to the left sub-derivation $\pi_{r'}$, we obtain $\pi'_r \ \triangleright \vdash r : T \to A$. Then π' is defined as follows.

$$\pi' := \dfrac{\begin{array}{cc}\vdots & \vdots \\ \pi'_r \ \triangleright \vdash r : T \to A & \pi_w \ \triangleright \vdash w : T\end{array}}{\vdash rw : A}\ \text{T-@}$$

□

Theorem C.2.3 (Completeness of tree types for Closed CbN)**.** *If t is Closed CbN terminating then there exists a tree type derivation $\pi \ \triangleright \vdash t : A$.*

Proof. Let $t \to_{wh}^{n} u$ the reduction of t to weak head normal form. By induction on n. Cases:

1. If $n = 0$ then $t = u$ is a weak head normal form, that is, an abstraction. Then it is typable with rule T-$\lambda_\star$.
2. If $n > 0$ then $t \to_{wh} r \to_{wh}^{n-1} u$. By *i.h.*, there exists $\pi'' \ \triangleright \vdash r : A$. By subject expansion Prop. C.2.2, there exists $\pi' \ \triangleright \vdash t : A$.

□

Appendix D

Encoding Space Sensitive TMs into the λ-Calculus

D.1 Preliminaries

Deterministic λ-Calculus. The language and the evaluation contexts of the deterministic λ-calculus $\Lambda_{\mathtt{det}}$ are given by:

$$\begin{array}{rrcl} \text{Terms} & t,u,r,w & ::= & v \mid tv \\ \text{Values} & v,w,v' & ::= & \lambda x.t \mid x \\ \\ \text{Evaluation Contexts} & E & ::= & \langle\cdot\rangle \mid Ev \end{array}$$

Note that

- *Arguments are values*: the right subterm of an application has to be a value, in contrast to what happens in the ordinary λ-calculus.
- *Weak evaluation*: evaluation contexts are *weak*, *i.e.* they do not enter inside abstractions.

Evaluation is then defined by:

$$\begin{array}{ll} \textsc{Rule at top level} & \textsc{Contextual closure} \\ (\lambda x.t)u \mapsto_\beta t\{x \leftarrow u\} & E\langle t\rangle \to_{det} E\langle u\rangle \quad \text{if } t \mapsto_\beta u \end{array}$$

Convention: to improve readability we omit some parenthesis, giving precedence to application with respect to abstraction. Therefore $\lambda x.tu$ stands for $\lambda x.(tu)$ and not for $(\lambda x.t)u$, that instead requires parenthesis.

The name of this calculus is motivated by the following immediate lemma.

Lemma D.1.1. *Let $t \in \Lambda_{\mathtt{det}}$. There is at most one $u \in \Lambda_{\mathtt{det}}$ such that $t \to_\beta u$, and in that case t is an application.*

Proof. By induction on t. If t is a value then it does not reduce. Then assume that t is an application $t = uv$. Let's apply the *i.h.* to u. Two cases:

1. *u reduces and it is an application*: then t has one redex, the one given by u (because $\langle\cdot\rangle v$ is an evaluation context), and no other one, because v does not reduce and u is not an abstraction by the *i.h.*
2. *u does not reduce*: if u is not an abstraction then t is normal, otherwise $u = \lambda x.r$ and $t = (\lambda x.r)v$ has exactly one redex.

□

Fixpoint. Let fix be the term $\theta\theta$, where

$$\theta := \lambda x.\lambda y.y(\lambda z.xxyz).$$

Now, given a term u let us show that $\mathsf{fix}\, u$ is a fixpoint of u up to η-equivalence.

$$\begin{array}{rcl} \mathsf{fix}\, u & = & (\lambda x.\lambda y.y(\lambda z.xxyz))\theta u \\ & \to_{det} & (\lambda y.y(\lambda z.\theta\theta yz))u \\ & \to_{det} & u(\lambda z.\theta\theta uz) \\ & = & u(\lambda z.\mathsf{fix}\, uz) \\ & =_{\eta} & u(\mathsf{fix}\, u) \end{array}$$

Encoding alphabets. Let $\Sigma = \{a_1, \ldots, a_n\}$ be a finite alphabet. Elements of Σ are encoded as follows:

$$\lceil a_i \rceil^{\Sigma} := \lambda x_1 \ldots . \lambda x_n . x_i \ .$$

When the alphabet will be clear from the context we will simply write $\lceil a \rceil_i$. Note that

1. the representation fixes a total order on Σ such that $a_i \leq a_j$ iff $i \leq j$;
2. the representation of an element $\lceil a_i \rceil^{\Sigma}$ requires space linear (and not logarithmic) in $|\Sigma|$. But, since Σ is fixed, it actually requires constant space.

Encoding strings. A string in $s \in \Sigma^*$ is represented by a term $\overline{s}^{\Sigma^*}$, defined by induction on the structure of s as follows:

$$\begin{array}{rcl} \overline{\varepsilon}^{\Sigma^*} & := & \lambda x_1 \ldots . \lambda x_n . \lambda y . y \ , \\ \overline{a_i r}^{\Sigma^*} & := & \lambda x_1 \ldots . \lambda x_n . \lambda y . x_i \overline{r}^{\Sigma^*} . \end{array}$$

Note that the representation depends on the cardinality of Σ. In other words, if $s \in \Sigma^*$ and $\Sigma \subset \Delta$, $\overline{s}^{\Sigma^*} \neq \overline{s}^{\Delta^*}$. In particular, $|\overline{s}^{\Sigma^*}| = \Theta(|s| \cdot |\Sigma|)$. The size of the alphabet is however considered as a fixed parameter, and so we rather have $|\overline{s}^{\Sigma^*}| = \Theta(|s|)$.

Lemma D.1.2 (Appending a character in constant time)**.** *Let Σ be an alphabet and $a \in \Sigma$ one of its characters. There is a term $\mathtt{append}^a_\Sigma$ such that for every continuation k and every string $s \in \Sigma^*$,*

$$\mathtt{append}^a_\Sigma k\overline{s} \to^{\mathcal{O}(1)}_{det} k\overline{(as)}.$$

Proof. Define the term $\mathtt{append}^a_\Sigma := \lambda k'.\lambda s'.k'(\lambda x_1 \ldots . \lambda x_{|\Sigma|} . \lambda y . x_{i_a} s')$ where i_a is the index of a in the ordering of Σ fixed by its encoding, that appends the character a to the string s' relatively to the alphabet Σ. We have:

$$\begin{array}{rcl} \mathtt{append}^a_\Sigma k\overline{s} & = & (\lambda k'.\lambda s'.k'(\lambda x_1 \ldots . \lambda x_{|\Sigma|} . \lambda y . x_{i_a} s'))k\overline{s} \\ & \to^2_{det} & k(\lambda x_1 \ldots . \lambda x_{|\Sigma|} . \lambda y . x_{i_a} \overline{s}) \\ & = & k\overline{(as)}. \end{array}$$

□

D.2 Binary Arithmetic

In order to navigate the input word, we consider a counter (in binary). Moving the head left (respectively right) amounts to decrement (respectively increment) the counter by one. The starting idea is to see a number as its binary string representation and to use the Scott encoding of strings. Since it is tricky to define the successor and predecessor on such an encoding, we actually define an ad-hoc encoding.

The first unusual aspect of our encoding is that the binary string is represented in reverse order, so that the representation of 2 is 01 and not 10. This is done to ease the definition of the successor and predecessor functions as λ-terms, which have to process strings from left to right, and that with the standard representation would have to go to the end of the string and then potentially back from right to left. With a reversed representation, these functions need to process the string only once from left to right.

The second unusual aspect is that, in order to avoid problems with strings made out of all 0s and strings having many 0s on the right (which are not meaningful), we collapse all suffixes made out of all 0 on to the empty string. A consequence is that the number 0 is then represented with the empty string. Non-rightmost 0 bits are instead represented with the usual Scott encoding.

If $n \in \mathbb{N}$ we write $\lfloor n \rfloor$ for the binary string representing n. Then we have:

$$\begin{array}{lcl} \lfloor 0 \rfloor & := & \varepsilon \\ \lfloor 1 \rfloor & := & 1 \\ \lfloor 2 \rfloor & := & 01 \\ \lfloor 3 \rfloor & := & 11 \\ \lfloor 4 \rfloor & := & 001 \end{array}$$

And so on. Binary strings are then encoded as λ-terms using the Scott encoding, as follows:

$$\begin{array}{lcl} \overline{\varepsilon} & := & \lambda x_0.\lambda x_1.\lambda y.y \\ \overline{0{\cdot}s} & := & \lambda x_0.\lambda x_1.\lambda y.x_0\overline{s} \\ \overline{1{\cdot}s} & := & \lambda x_0.\lambda x_1.\lambda y.x_1\overline{s} \end{array}$$

Successor Function. The successor function SUCC on the reversed binary representation can be defined as follows (in Haskell-like syntax):

$$\begin{array}{llcl} \text{SUCC} & \varepsilon & = & 1 \\ \text{SUCC} & 0{\cdot}s & = & 1{\cdot}s \\ \text{SUCC} & 1{\cdot}s & = & 0{\cdot}(\text{SUCC}\, s) \end{array}$$

For which we have $\text{SUCC}(\lfloor n \rfloor) = \lfloor n+1 \rfloor$.

Lemma D.2.1. *There is a λ-term* succ *such that for every continuation k and every natural number $n \in \mathbb{N}$,*

$$\mathtt{succ}\, k\overline{\lfloor n \rfloor} \to_{det}^{\mathcal{O}(|\lfloor n \rfloor|)} k\,\overline{\text{SUCC}\ \lfloor n \rfloor}.$$

Proof. Define $\mathtt{succ} := \Theta\mathtt{succaux}$ and $\mathtt{succaux} := \lambda f.\lambda k'.\lambda n'.n' N_0 N_1 N_\varepsilon f k'$ where:

- $N_0 := \lambda f'.\lambda s'.\lambda k'.\mathtt{append}^1 k' s'$
- $N_1 := \lambda f'.\lambda s'.\lambda k'.f'(\lambda z.\mathtt{append}^0 k' z)s'$
- $N_\varepsilon := \lambda f'.\lambda k'.k'\,\overline{1{\cdot}\varepsilon}$

The first steps of the evaluation of $\mathtt{succ}\, k\overline{\lfloor n \rfloor}$ are common to all natural numbers $n \in \mathbb{N}$:

$$\begin{array}{lcl} \mathtt{succ}\, k\overline{\lfloor n \rfloor} & = & \mathsf{fix}\,\mathtt{succaux}\, k\overline{\lfloor n \rfloor} \\ & \to_\beta^2 & \mathtt{succaux}(\lambda z.\mathtt{succ}\, z)k\overline{\lfloor n \rfloor} \\ & = & (\lambda f.\lambda k'.\lambda n'.n' N_0 N_1 N_\varepsilon f k')(\lambda z.\mathtt{succ}\, z)k\overline{\lfloor n \rfloor} \\ & \to_\beta^3 & \overline{\lfloor n \rfloor} N_0 N_1 N_\varepsilon (\lambda z.\mathtt{succ}\, z)k \end{array}$$

Cases of n:

- *Zero*, that is, $n = 0$, $\lfloor n \rfloor = \varepsilon$, and $\overline{\lfloor n \rfloor} = \lambda x_0.\lambda x_1.\lambda y.y$: then

$$\begin{array}{rcl}
\overline{\lfloor n \rfloor} N_0 N_1 N_\varepsilon (\lambda z.\mathtt{succ}\, z) k & = & (\lambda x_0.\lambda x_1.\lambda y.y) N_0 N_1 N_\varepsilon (\lambda z.\mathtt{succ}\, z) k \\
& \to_\beta^3 & N_\varepsilon (\lambda z.\mathtt{succ}\, z) k \\
& = & (\lambda f'.\lambda k'.k'\, \overline{1{\cdot}\varepsilon}) k \\
& \to_\beta & k\, \overline{1{\cdot}\varepsilon} \\
& = & k\, \overline{\mathrm{SUCC}\, \lfloor 0 \rfloor}
\end{array}$$

- *Not zero*. Then there are two sub-cases, depending on the first character of the string $\lfloor n \rfloor$:
  - 0 *character*, *i.e.* $\lfloor n \rfloor = 0{\cdot}s$: then

$$\begin{array}{rcl}
\overline{0{\cdot}r} N_0 N_1 N_\varepsilon (\lambda z.\mathtt{succ}\, z) k & = & (\lambda x_0.\lambda x_1.\lambda y.x_0 \bar{s}) N_0 N_1 N_\varepsilon (\lambda z.\mathtt{succ}\, z) k \\
& \to_\beta^3 & N_0 \bar{s} (\lambda z.\mathtt{succ}\, z) k \\
& = & (\lambda f'.\lambda s'.\lambda k'.\mathtt{append}^1 k' s') \bar{s} (\lambda z.\mathtt{succ}\, z) k \\
& \to_\beta^3 & \mathtt{append}^1 k \bar{s} \\
(L.\ D.1.2) & \to_\beta^{\mathcal{O}(1)} & k\, \overline{1{\cdot}s} \\
& = & k\, \overline{\mathrm{SUCC}\, \lfloor n \rfloor}
\end{array}$$

  - 1 *character*, *i.e.* $\lfloor n \rfloor = 1{\cdot}s$: then

$$\begin{array}{rcl}
\overline{1{\cdot}r} N_0 N_1 N_\varepsilon (\lambda z.\mathtt{succ}\, z) k & = & (\lambda x_0.\lambda x_1.\lambda y.x_1 \bar{s}) N_0 N_1 N_\varepsilon (\lambda z.\mathtt{succ}\, z) k \\
& \to_\beta^3 & N_1 \bar{s} (\lambda z.\mathtt{succ}\, z) k \\
& = & (\lambda f'.\lambda s'.\lambda k'.f' (\lambda z.\mathtt{append}^0 k' z) s') \bar{s} (\lambda z.\mathtt{succ}\, z) k \\
& \to_\beta^3 & (\lambda z.\mathtt{succ}\, z)\, (\lambda z.\mathtt{append}^0 k z) \bar{s} \\
& \to_\beta & \mathtt{succ}\, (\lambda z.\mathtt{append}^0 k z) \bar{s} \\
(i.h.) & \to_\beta^{\mathcal{O}(|s|)} & (\lambda z.\mathtt{append}^0 k z) \overline{\mathrm{SUCC}\, s} \\
& \to_{det} & \mathtt{append}^0 k \overline{\mathrm{SUCC}\, s} \\
(L.\ D.1.2) & \to_\beta^{\mathcal{O}(1)} & k\, \overline{0{\cdot}(\mathrm{SUCC}\, s)} \\
& = & k\, \overline{\mathrm{SUCC}\, (1{\cdot}s)} \\
& = & k\, \overline{\mathrm{SUCC}\, \lfloor n \rfloor}
\end{array}$$

□

Predecessor Function. We now define and implement a predecessor function. We define it assuming that it shall only be applied to the enconding $\lfloor n \rfloor$ of a natural number n different from 0, as it shall indeed be the case in the following. Such a predecessor function PRED is defined as follows on the reversed binary representation (in Haskell-like syntax):

$$\begin{array}{llcl}
\mathrm{PRED} & 0{\cdot}s & = & 1{\cdot}(\mathrm{PRED}\, s) \\
\mathrm{PRED} & 1{\cdot}\varepsilon & = & \varepsilon \\
\mathrm{PRED} & 1{\cdot}b{\cdot}s & = & 0{\cdot}b{\cdot}s
\end{array}$$

It is easily seen that $\mathrm{PRED}(\lfloor n \rfloor) = \lfloor n - 1 \rfloor$ for all $0 < n \in \mathbb{N}$. Note that $\mathrm{PRED}(\lfloor n \rfloor)$ does not introduce a rightmost 0 bit when it changes the rightmost bit of $\lfloor n \rfloor$, that is, $\mathrm{PRED}\, 001 = 11$ and not 110.

Lemma D.2.2. *There is a λ-term* `pred` *such that for every continuation k and every natural number* $1 \leq n \in \mathbb{N}$,

$$\mathtt{pred}\, k \overline{\lfloor n \rfloor} \to_{det}^{\mathcal{O}(|\lfloor n \rfloor|)} k\, \overline{\mathrm{PRED}\, \lfloor n \rfloor}.$$

Proof. Define $\mathtt{pred} := \mathrm{fix}\, \mathtt{predaux}$ and $\mathtt{predaux} := \lambda f.\lambda k'.\lambda n'.n' N_0 N_1 N_\varepsilon f k'$ where:

- $N_0 := \lambda r'.\lambda f.\lambda k'.f (\lambda z.\mathtt{append}^1 k' z) r'$;

- $N_1 := \lambda r'.\lambda f.r' M_0 M_1 M_\varepsilon$, where:
  - $M_0 := \lambda v.\lambda k.\mathtt{append}^0(\lambda z.\mathtt{append}^1 k z)v$;
  - $M_1 := \lambda v.\lambda k.\mathtt{append}^1(\lambda z.\mathtt{append}^1 k z)v$;
  - $M_\varepsilon := \lambda k'.k'\overline{\varepsilon}$;
- N_ε is whatever closed term.

The first steps of the evaluation of $\mathtt{pred}\,k\overline{\lfloor n \rfloor}$ are common to all natural numbers $1 \leq n \in \mathbb{N}$:

$$\begin{array}{rcl} \mathtt{pred}\,k\overline{\lfloor n \rfloor} & = & \mathsf{fix}\,\mathtt{predaux}\,k\overline{\lfloor n \rfloor} \\ & \to_\beta^2 & \mathtt{predaux}(\lambda z.\mathtt{pred}\,z)k\overline{\lfloor n \rfloor} \\ & = & (\lambda f.\lambda k'.\lambda n'.n' N_0 N_1 N_\varepsilon f k')(\lambda z.\mathtt{pred}\,z)k\overline{\lfloor n \rfloor} \\ & \to_\beta^3 & \overline{\lfloor n \rfloor} N_0 N_1 N_\varepsilon (\lambda z.\mathtt{pred}\,z)k \end{array}$$

By hypothesis, $n \geq 1$. Then $\overline{n}$ is a non-empty string. Cases of its first character:

- 0 *character*, *i.e.* $\lfloor n \rfloor = 0{\cdot}r$: then

$$\begin{array}{rcl} \overline{\lfloor n \rfloor} N_0 N_1 N_\varepsilon (\lambda z.\mathtt{pred}\,z)k & = & (\lambda x_0.\lambda x_1.\lambda y.x_0\overline{r})N_0 N_1 N_\varepsilon(\lambda z.\mathtt{pred}\,z)k \\ & \to_\beta^3 & N_0\overline{r}(\lambda z.\mathtt{pred}\,z)k \\ & = & (\lambda r'.\lambda f.\lambda k'.f(\lambda z.\mathtt{append}^1 k' z)r')\overline{r}(\lambda z.\mathtt{pred}\,z)k \\ & \to_\beta^3 & (\lambda z.\mathtt{pred}\,z)(\lambda z.\mathtt{append}^1 k' z)\overline{r} \\ & \to_\beta & \mathtt{pred}(\lambda z.\mathtt{append}^1 k z)\overline{r} \\ (\textit{i.h.}) & \to_\beta^{\mathcal{O}(|r|)} & (\lambda z.\mathtt{append}^1 k z)\,\overline{\mathrm{PRED}\,r} \\ & \to_\beta & \mathtt{append}^1 k\,\overline{\mathrm{PRED}\,r} \\ (\textit{L. D.1.2}) & \to_\beta^{\mathcal{O}(1)} & k\,\overline{1{\cdot}(\mathrm{PRED}\,r)} \\ & = & k\,\overline{\mathrm{PRED}\,0{\cdot}r} \\ & = & k\,\overline{\mathrm{PRED}\,\lfloor n \rfloor} \end{array}$$

- 1 *character*, *i.e.* $\lfloor n \rfloor = 1{\cdot}r$: then

$$\begin{array}{rcl} \overline{\lfloor n \rfloor} N_0 N_1 N_\varepsilon (\lambda z.\mathtt{pred}\,z)k & = & (\lambda x_0.\lambda x_1.\lambda y.x_1\overline{r})N_0 N_1 N_\varepsilon(\lambda z.\mathtt{pred}\,z)k \\ & \to_\beta^3 & N_1\overline{r}(\lambda z.\mathtt{pred}\,z)k \\ & = & (\lambda r'.\lambda f.r' M_0 M_1 M_\varepsilon)\overline{r}(\lambda z.\mathtt{pred}\,z)k \\ & \to_\beta & \overline{r} M_0 M_1 M_\varepsilon k \end{array}$$

  There are three sub-cases, depending on the string r:

  - r is empty, *i.e.* $r = \varepsilon$. Then:

$$\begin{array}{rcl} \overline{\varepsilon} M_0 M_1 M_\varepsilon k & = & (\lambda x_0.\lambda x_1.\lambda y.y)M_0 M_1 M_\varepsilon k \\ & \to_\beta^3 & M_\varepsilon k \\ & = & (\lambda k'.k'\overline{\varepsilon})k \\ & \to_\beta & k\overline{\varepsilon} \\ & \to_\beta & k\overline{\lfloor 0 \rfloor} \\ & = & k\overline{\mathrm{PRED}\,\lfloor 1 \rfloor} \end{array}$$

– *r start with* 0, that is, $\lfloor n \rfloor = 1{\cdot}r = 1{\cdot}0{\cdot}p$. Then:

$$\begin{array}{rcl}
\overline{0{\cdot}p}\, M_0 M_1 M_\varepsilon k & = & (\lambda x_0.\lambda x_1.\lambda y.x_0\overline{p})M_0M_1M_\varepsilon k \\
& \to_\beta^3 & M_0\overline{p}k \\
& = & (\lambda v.\lambda k.\mathtt{append}^0(\lambda z.\mathtt{append}^0 kz)v)\overline{p}k \\
& \to_\beta^2 & \mathtt{append}^0(\lambda z.\mathtt{append}^0 kz)\overline{p} \\
(L.\ D.1.2) & \to_\beta^{\mathcal{O}(1)} & (\lambda z.\mathtt{append}^0 kz)\overline{0{\cdot}p} \\
& \to_\beta & \mathtt{append}^0 k\,\overline{0{\cdot}p} \\
(L.\ D.1.2) & \to_\beta^{\mathcal{O}(1)} & k\,\overline{0{\cdot}0{\cdot}p} \\
& = & k\,\overline{0{\cdot}r} \\
& = & k\,\overline{\mathrm{PRED}\ 1{\cdot}r} \\
& = & k\,\overline{\mathrm{PRED}\ \lfloor n \rfloor}
\end{array}$$

– *r start with* 1, that is, $\lfloor n \rfloor = 1{\cdot}r = 1{\cdot}1{\cdot}p$. Then:

$$\begin{array}{rcl}
\overline{0{\cdot}p}\, M_0 M_1 M_\varepsilon k & = & (\lambda x_0.\lambda x_1.\lambda y.x_1\overline{p})M_0M_1M_\varepsilon k \\
& \to_\beta^3 & M_1\overline{p}k \\
& = & (\lambda v.\lambda k.\mathtt{append}^1(\lambda z.\mathtt{append}^0 kz)v)\overline{p}k \\
& \to_\beta^2 & \mathtt{append}^1(\lambda z.\mathtt{append}^0 kz)\overline{p} \\
(L.\ D.1.2) & \to_\beta^{\mathcal{O}(1)} & (\lambda z.\mathtt{append}^0 kz)\overline{1{\cdot}p} \\
& \to_\beta & \mathtt{append}^0 k\,\overline{1{\cdot}p} \\
(L.\ D.1.2) & \to_\beta^{\mathcal{O}(1)} & k\,\overline{0{\cdot}1{\cdot}p} \\
& = & k\,\overline{0{\cdot}r} \\
& = & k\,\overline{\mathrm{PRED}\ 1{\cdot}r} \\
& = & k\,\overline{\mathrm{PRED}\ \lfloor n \rfloor}
\end{array}$$

□

Lookup Function. Given a natural number n, we need to be able to extract the $n+1$-th character from a non-empty string s. The partial function LOOKUP can be defined as follows (in Haskell-like syntax):

$$\begin{array}{llcl}
\mathrm{LOOKUP} & \lfloor 0 \rfloor\ (c{\cdot}s) & = & c \\
\mathrm{LOOKUP} & \lfloor n \rfloor\ (c{\cdot}s) & = & \mathrm{LOOKUP}\ (\mathrm{PRED}\ \lfloor n \rfloor)\ s \quad \text{if } n > 0
\end{array}$$

Lemma D.2.3. *There is a λ-term* `lookup` *such that for every continuation k, every natural number n and every non-empty string* $i \in \mathbb{B}^+$,

$$\mathtt{lookup}\, k\,\overline{\lfloor n \rfloor}\,\overline{i} \to_{det}^{\mathcal{O}(n \log n)} k\,\overline{\lceil \mathrm{LOOKUP}\ \lfloor n \rfloor i \rceil}.$$

Proof. We can now code the function `lookup` := fix `lookupaux` where:

$$\mathtt{lookupaux} := \lambda f.\lambda k'.\lambda n'.\lambda i'.n' N_0 N_1 N_\varepsilon f k' i'$$

where:

- $N_0 := \lambda p'.\lambda f.\lambda k'.\lambda i'.i' M_0 M_0 M_\varepsilon p' f k'$, where
  - $M_0 := \lambda r'.\lambda p'.\lambda f.\lambda k'.\mathtt{append}^0(\lambda z''.\mathtt{pred}(\lambda z'.fk'z')z'')p'r'$;
  - M_ε is whatever closed term.
- $N_1 := \lambda p'.\lambda f.\lambda k'.\lambda i'.i' M_1 M_1 M_\varepsilon p' f k'$, where
  - $M_1 := \lambda r'.\lambda p'.\lambda f.\lambda k'.\mathtt{append}^1(\lambda z''.\mathtt{pred}(\lambda z'.fk'z')z'')p'r'$;
  - M_ε is whatever closed term.

- $N_\varepsilon := \lambda f.\lambda k'.\lambda i'.i' O_0 O_1 O_{\mathsf{L}} O_{\mathsf{R}} O_\varepsilon k'$, where
  - $O_b := \lambda s'.\lambda k'.k' \lceil b \rceil$;
  - O_ε is whatever closed term.

The first steps of the evaluation of $\mathtt{lookup}\, k \overline{\lfloor n \rfloor} \bar{i}$ are common to all strings $i \in \mathbb{B}^+$ and natural numbers $n \in \mathbb{N}$:

$$\begin{array}{lll} \mathtt{lookup}\, k \overline{\lfloor n \rfloor} \bar{i} & = & \mathsf{fix}\, \mathtt{lookupaux}\, k \overline{\lfloor n \rfloor} \bar{i} \\ & \to_\beta^2 & \mathtt{lookupaux} (\lambda z.\mathtt{lookup}\, z) k \overline{\lfloor n \rfloor} \bar{i} \\ & = & (\lambda f.\lambda k'.\lambda n'.\lambda i'.n' N_0 N_1 N_\varepsilon f k' i') (\lambda z.\mathtt{lookup}\, z) k \overline{\lfloor n \rfloor} \bar{i} \\ & \to_\beta^4 & \overline{\lfloor n \rfloor} N_0 N_1 N_\varepsilon (\lambda z.\mathtt{lookup}\, z) k \bar{i} \end{array}$$

Cases of n:

- $n = 0$, and so $\lfloor n \rfloor = \varepsilon$: then

$$\begin{array}{lll} & \bar{\varepsilon} N_0 N_1 N_\varepsilon (\lambda z.\mathtt{lookup}\, z) k \bar{i} \\ = & (\lambda x_0.\lambda x_1.\lambda y.y) N_0 N_1 N_\varepsilon (\lambda z.\mathtt{lookup}\, z) k \bar{i} \\ \to_\beta^3 & N_\varepsilon (\lambda z.\mathtt{lookup}\, z) k \bar{i} \\ = & (\lambda f.\lambda k'.\lambda i'.i' O_0 O_1 O_{\mathsf{L}} O_{\mathsf{R}} O_\varepsilon k') (\lambda z.\mathtt{lookup}\, z) k \bar{i} \\ \to_{det}^3 & \bar{i} O_0 O_1 O_{\mathsf{L}} O_{\mathsf{R}} O_\varepsilon k \end{array}$$

  Let i start with $b \in \mathbb{B}$, that is, $i = b{\cdot}s$:

$$\begin{array}{lll} \overline{b{\cdot}s}\, O_0 O_1 O_{\mathsf{L}} O_{\mathsf{R}} O_\varepsilon k & = & (\lambda x_0.\lambda x_1.\lambda x_{._\varepsilon} x_b \bar{s}) O_0 O_1 O_{\mathsf{L}} O_{\mathsf{R}} O_\varepsilon k \\ & \to_{det}^3 & O_b \bar{s} k \\ & = & (\lambda s'.\lambda k'.k' \lceil b \rceil) \bar{s} k \\ & \to_\beta^2 & k \lceil b \rceil \\ & = & k \lceil \textsc{lookup}\, \varepsilon (b{\cdot}s) \rceil \\ & = & k \lceil \textsc{lookup}\, \lfloor 0 \rfloor i \rceil \end{array}$$

- *Non-empty string starting with* 0, that is, $n > 0$ and $\lfloor n \rfloor = 0{\cdot}p$: then

$$\begin{array}{lll} & \overline{\lfloor n \rfloor} N_0 N_1 N_\varepsilon (\lambda z.\mathtt{lookup}\, z) k \bar{i} \\ = & (\lambda x_0.\lambda x_1.\lambda y.x_0 \overline{p}) N_0 N_1 N_\varepsilon (\lambda z.\mathtt{lookup}\, z) k \bar{i} \\ \to_\beta^3 & N_0 \overline{p} (\lambda z.\mathtt{lookup}\, z) k \bar{i} \\ = & (\lambda p'.\lambda f.\lambda k'.\lambda i'.i' M_0 M_0 M_\varepsilon p' f k') \overline{p} (\lambda z.\mathtt{lookup}\, z) k \bar{i} \\ \to_\beta^4 & \bar{i} M_0 M_0 M_\varepsilon \overline{p} (\lambda z.\mathtt{lookup}\, z) k \end{array}$$

  Let i start with $b \in \mathbb{B}$, that is, $i = b{\cdot}r$:

$$
\begin{array}{lll}
\overline{b\cdot r}M_0M_0M_\varepsilon k & = & (\lambda x_0.\lambda x_1.\lambda x_\varepsilon.x_b\bar{r})M_0M_0M_\varepsilon\bar{p}(\lambda z.\texttt{lookup}\,z)k \\
 & \to_{det}^{|\Sigma|} & M_0\bar{r}\,\bar{p}(\lambda z.\texttt{lookup}\,z)k \\
 & = & (\lambda r'.\lambda p'.\lambda f.\lambda k'.\texttt{append}^0(\lambda z''.\texttt{pred}(\lambda z'.fk'z')z'')p'r')\bar{r}\,\bar{p}(\lambda z.\texttt{lookup}\,z)k \\
 & \to_{det}^{4} & \texttt{append}^0(\lambda z''.\texttt{pred}(\lambda z'.(\lambda z.\texttt{lookup}\,z)kz')z'')\bar{p}\,\bar{r} \\
(L.\ D.1.2 & \to_{det}^{\mathcal{O}(1)} & (\lambda z''.\texttt{pred}(\lambda z'.(\lambda z.\texttt{lookup}\,z)kz')z'')\overline{0\cdot p}\,\bar{r} \\
 & \to_{det} & \texttt{pred}(\lambda z'.(\lambda z.\texttt{lookup}\,z)kz')\overline{0\cdot p}\,\bar{r} \\
 & = & \texttt{pred}(\lambda z'.(\lambda z.\texttt{lookup}\,z)kz')\overline{\lfloor n\rfloor}\,\bar{r} \\
(L.\ D.2.2) & \to_{det}^{\mathcal{O}(|\lfloor n\rfloor|)} & (\lambda z'.(\lambda z.\texttt{lookup}\,z)kz')\overline{\text{PRED}\,\lfloor n\rfloor}\,\bar{r} \\
 & \to_{det}^{2} & \texttt{lookup}\,k\,\overline{\text{PRED}\,\lfloor n\rfloor}\,\bar{r} \\
 & = & \texttt{lookup}\,k\,\overline{\lfloor n-1\rfloor}\,\bar{r} \\
(i.h.) & \to_{det}^{\mathcal{O}((n-1)\cdot\log(n-1))} & k(\lceil\text{LOOKUP}\,\lfloor n-1\rfloor r\rceil) \\
 & = & k(\lceil\text{LOOKUP}\,\lfloor n\rfloor s\rceil)
\end{array}
$$

The number of β steps then is $\mathcal{O}(|\lfloor n\rfloor|)+\mathcal{O}((n-1)\cdot\log(n-1))+h$ for a certain constant h. Since $|\lfloor n\rfloor| = \mathcal{O}(\log n)$, we obtain that the number of steps is bound by $\mathcal{O}(n\cdot\log n)$, as required.

- *Non-empty string starting with* 1, that is, $n > 0$ and $\lfloor n\rfloor = 1\cdot p$: same as the previous one, simply replacing N_0 with N_1, and thus M_0 with M_1. In particular, it takes the same number of steps.

□

D.3 Encoding Turing Machines

Turing Machines. Let $\mathbb{B}_\mathsf{I} := \{0,1,\mathsf{L},\mathsf{R}\}$ and $\mathbb{B}_\mathsf{W} := \{0,1,\square\}$ where L and R delimit the input (binary) string, and $\square$ is our notation for the blank symbol. A deterministic binary Turing machine $\mathcal{M}$ *with input* is a tuple $(S, s_{in}, s_T, s_F, \delta)$ consisting of:

- A finite set $Q = \{s_1,\ldots,s_m\}$ of *states*;
- A distinguished state $s_{in} \in Q$, called the *initial state*;
- Two distinguished states $S_{fin} := \{s_T, s_F\} \subseteq S$, called the *final states*;
- A partial *transition function* $\delta : \mathbb{B}_\mathsf{I} \times \mathbb{B}_\mathsf{W} \times S \rightharpoonup \{-1,+1,0\} \times \mathbb{B}_\mathsf{W} \times \{\leftarrow,\rightarrow,\downarrow\} \times S$ such that $\delta(b,a,s)$ is defined only if $s \notin S_{fin}$.

A configuration for $\mathcal{M}$ is a tuple

$$(i, n, w_l, a, w_r, s) \in \mathbb{B}_\mathsf{I}^* \times \mathbb{N} \times \mathbb{B}_\mathsf{W}^* \times \mathbb{B}_\mathsf{W} \times \mathbb{B}_\mathsf{W}^* \times S$$

where:

- i is the immutable input string and is formed as $i = \mathsf{L}\cdot s\cdot\mathsf{R}$, $s \in \mathbb{B}^*$;
- $n \in \mathbb{N}$ represents the position of the input head. It is meant to be represented in binary (that is, as an element of $\mathbb{B}^*$), to take space $\log n$, but for ease of reading we keep referring to it as a number rather than as a string;
- $w_l \in \mathbb{B}_\mathsf{W}^*$ is the work tape on the left of the work head;
- $a \in \mathbb{B}_\mathsf{W}$ is the element on the cell of the work tape read by the work head;
- $w_r \in \mathbb{B}_\mathsf{W}^*$ is the work tape on the right of the work head;

- $s \in S$ is the state of the machine.

For readability, we usually write a configuration (i, n, w_l, a, w_r, s) as $(i, n \mid w_l, a, w_r \mid s)$, separating the input components, the working components, and the current state.

Given an input string $i \in \mathbb{B}^*_{\mathsf{I}}$ (where $i = \mathsf{L}{\cdot}s{\cdot}\mathsf{R}$ and $s \in \mathbb{B}^*$) we define:

- the *initial configuration* $C_{\mathtt{in}}(i)$ for i is $C_{\mathtt{in}}(i) := (i, 0 \mid \varepsilon, \square, \varepsilon \mid s_{in})$,
- the *final configuration* $C_{\mathtt{fin}} := (s, n \mid w_l, a, w_r \mid s)$, where $s \in S_{fin}$.

For readability, a transition, say, $\delta(i_n, a, s) = (-1, a', \leftarrow, s')$, is usually written as $(-1 \mid a', \leftarrow \mid s')$ to stress the three components corresponding to those of configurations (input, work, state).

As in Goldreich, we assume that the machine never scans the input beyond the boundaries of the input. This does not affects space complexity.

An example of transition: if $\delta(i_n, a, s) = (-1 \mid a', \leftarrow \mid s')$, then $\mathcal{M}$ evolves from $C = (i, n \mid w_l a'', a, w_r \mid s)$, where the nth character of i is i_n, to $D = (i, n-1 \mid w_l, a'', a' w_r \mid s')$ and if the tape on the left of the work head is empty, *i.e.* if $C = (i, n \mid \varepsilon, a, w_r \mid s)$, then the content of the new head cell is a blank symbol, that is, $D := (i, n-1 \mid \varepsilon, \square, a' w_r \mid s')$. The same happens if the tape on the right of the work head is empty. If $\mathcal{M}$ has a transition from C to D we write $C \to_{\mathcal{M}} D$. A configuration having as state a final state $s \in S_{fin}$ is *final* and cannot evolve.

A Turing machine $(S, s_{in}, s_T, s_F, \delta)$ computes the function $f : \mathbb{B}^* \to \mathbb{B}$ in time $T : \mathbb{N} \to \mathbb{N}$ and space $S : \mathbb{N} \to \mathbb{N}$ if for every $i \in \mathbb{B}^+$, the initial configuration for i evolves to a final configuration of state $s_{f(i)}$ in $T(|i|)$ steps and using at most $S(|i|)$ cells on the work tape.

Encoding configurations. A configuration $(i, n \mid s, a, r \mid s)$ of a machine $\mathcal{M} = (S, s_{in}, s_T, s_F, \delta)$ is represented by the term

$$\overline{(i, n \mid w_l, a, w_r \mid s)}^{\mathcal{M}} := \lambda x.(x \bar{i}^{\mathbb{B}^+} \overline{\lfloor n \rfloor}^{\mathbb{B}} \overline{w_l^{\mathsf{R}}}^{\mathbb{B}^*_{\mathsf{W}}} \lceil a \rceil^{\mathbb{B}_{\mathsf{W}}} \overline{w_r}^{\mathbb{B}^*_{\mathsf{W}}} \lceil s \rceil^{S}).$$

where w_l^{R} is the string w_l with the elements in reverse order. We shall often rather write

$$\overline{(i, n \mid w_l, a, w_r \mid s)} := \lambda x.(x\ \bar{i}\ \overline{\lfloor n \rfloor}\ \overline{w_l^{\mathsf{R}}}\ \lceil a \rceil\ \overline{w_r}\ \lceil s \rceil).$$

letting the superscripts implicit. To ease the reading, we sometimes use the following notation for tuples $\langle s, q \mid t, u, r \mid w \rangle := \lambda x.(xsqturw)$, so that $\overline{(i, n \mid w_l, a, w_r \mid s)} = \langle \bar{i}, \overline{\lfloor n \rfloor} \mid \overline{w_l^{\mathsf{R}}}, \lceil a \rceil, \overline{w_r} \mid \lceil s \rceil \rangle$.

Turning the input string into the initial configuration. The following lemma provides the term `init` that builds the initial configuration.

Lemma D.3.1 (Turning the input string into the initial configuration)**.** *Let $\mathcal{M} = (S, s_{in}, s_T, s_F, \delta)$ be a Turing machine. There is a term $\mathtt{init}^{\mathcal{M}}$, or simply `init`, such that for every input string $i \in \mathbb{B}^*_{\mathsf{I}}$ (where $i = \mathsf{L}{\cdot}s{\cdot}\mathsf{R}$ and $s \in \mathbb{B}^*$):*

$$\mathtt{init}\, k\, \bar{i} \quad \to^{\Theta(1)}_{det} \quad k\, \overline{C_{\mathtt{in}}(i)}$$

where $C_{\mathtt{in}}(i)$ is the initial configuration of $\mathcal{M}$ for i.

Proof. Define

$$\mathtt{init} := (\lambda d.\lambda e.\lambda f.\lambda k'.\lambda i'.k' \langle i', d \mid e, \lceil \square \rceil^{\mathbb{B}_{\mathsf{W}}}, f \mid \lceil s_{in} \rceil^{S} \rangle) \overline{\lfloor 0 \rfloor} \bar{\varepsilon}^{\mathbb{B}^*_{\mathsf{W}}} \bar{\varepsilon}^{\mathbb{B}^*_{\mathsf{W}}}$$

Please note that the term is not in normal form. This is for technical reasons that will be clear next. Then

$$\begin{array}{rcl}
\mathtt{init}\, k\, \bar{i}^{\mathbb{B}^*_{\mathsf{I}}} & = & (\lambda d.\lambda e.\lambda f.\lambda k'.\lambda i'.k' \langle i', d \mid e, \lceil \square \rceil^{\mathbb{B}_{\mathsf{W}}}, f \mid \lceil s_{in} \rceil^{S} \rangle) \overline{\lfloor 0 \rfloor} \bar{\varepsilon}^{\mathbb{B}^*_{\mathsf{W}}} \bar{\varepsilon}^{\mathbb{B}^*_{\mathsf{W}}} k \bar{i}^{\mathbb{B}^*_{\mathsf{I}}} \\
& \to^{5}_{det} & k \langle \bar{i}^{\mathbb{B}^*_{\mathsf{I}}}, \overline{\lfloor 0 \rfloor} \mid \bar{\varepsilon}^{\mathbb{B}^*_{\mathsf{W}}}, \lceil \square \rceil^{\mathbb{B}_{\mathsf{W}}}, \bar{\varepsilon}^{\mathbb{B}^*_{\mathsf{W}}} \mid \lceil s_{in} \rceil^{S} \rangle \\
& = & k\, \overline{(i, 0 \mid \varepsilon, \square, \varepsilon \mid s_{in})} \\
& = & k\, \overline{C_{\mathtt{in}}(i)}
\end{array}$$

□

Extracting the output from the final configuration.

Lemma D.3.2 (Extracting the output from the final configuration)**.** *Let $\mathcal{M} = (S, s_{in}, s_T, s_F, \delta)$ be a Turing machine. There is a term $\texttt{final}^{\mathcal{M}}$, or simply $\texttt{final}$, such that for every final configuration C of state $s \in S_{fin}$*

$$\texttt{final}\,k\,\overline{C} \to_{det}^{\Theta(|S|)} \begin{cases} k(\lambda x.\lambda y.x) & \text{if } s = s_T \\ k(\lambda x.\lambda y.y) & \text{if } s = s_F \end{cases}$$

Proof. Define

$$\texttt{final} := \lambda k'.\lambda C'.C'(\lambda i'.\lambda n'.\lambda w_l'.\lambda a'.\lambda w_r'.\lambda s'.s' N_1 \dots N_{|S|} k')$$

where:

$$N_i := \begin{cases} \lambda k'.k'(\lambda x.\lambda y.x) & \text{if } q_i = s_T \\ \lambda k'.k'(\lambda x.\lambda y.y) & \text{if } q_i = s_F \\ \text{whatever closed term (say, the identity)} & \text{otherwise} \end{cases}$$

Then:

$$\begin{array}{rcl} \texttt{final}\,k\,\overline{C} & = & (\lambda k'.\lambda C'.C'(\lambda i'.\lambda n'.\lambda w_l'.\lambda a'.\lambda w_r'.\lambda s'.s' N_1 \dots N_{|S|} k'))k\overline{C} \\ & \to_{det}^{2} & \overline{C}(\lambda i'.\lambda n'.\lambda w_l'.\lambda a'.\lambda w_r'.\lambda s'.s' N_1 \dots N_{|S|} k) \\ & = & \overline{(i, n \mid w_l, a, w_r \mid s)}(\lambda i'.\lambda n'.\lambda w_l'.\lambda a'.\lambda w_r'.\lambda s'.s' N_1 \dots N_{|S|} k) \\ & = & (\lambda x.x \overline{i}^{\mathbb{B}^*} \overline{\lfloor n \rfloor}\ \overline{w_l^{\mathrm{R}}}^{\mathbb{B}_{\mathsf{W}}^*}\ \lceil a \rceil^{\mathbb{B}_{\mathsf{W}}}\ \overline{w_r}^{\mathbb{B}_{\mathsf{W}}^*}\ \lceil s \rceil^{S})(\lambda i'.\lambda n'.\lambda w_l'.\lambda a'.\lambda w_r'.\lambda s'.s' N_1 \dots N_{|S|} k) \\ & \to_{det} & (\lambda i'.\lambda n'.\lambda w_l'.\lambda a'.\lambda w_r'.\lambda s'.s' N_1 \dots N_{|S|} k) \overline{i}^{\mathbb{B}^*} \overline{\lfloor n \rfloor}\ \overline{w_l^{\mathrm{R}}}^{\mathbb{B}_{\mathsf{W}}^*}\ \lceil a \rceil^{\mathbb{B}_{\mathsf{W}}}\ \overline{w_r}^{\mathbb{B}_{\mathsf{W}}^*}\ \lceil s \rceil^{S} \\ & \to_{det}^{6} & \lceil s \rceil^{S} N_1 \dots N_{|S|} k \\ & = & (\lambda x_1 \dots x_{|S|}.x_j) N_1 \dots N_{|S|} k \\ & \to_{det}^{|S|} & N_j k \end{array}$$

If $s = s_T$, then:

$$\begin{array}{rcl} N_j k & = & (\lambda k'.k'(\lambda x.\lambda y.x))k \\ & \to_{det} & k(\lambda x.\lambda y.x) \end{array}$$

If $s = s_F$, then:

$$\begin{array}{rcl} N_j k & = & (\lambda k'.k'(\lambda x.\lambda y.y))k \\ & \to_{det} & k(\lambda x.\lambda y.y) \end{array}$$

□

Simulation of a machine transition. Now we show how to encode the transition function δ of a Turing machine as a λ-term in such a way to simulate every single transition in constant? time. This is the heart of the encoding, and the most involved proof.

Lemma D.3.3 (Simulation of a machine transition)**.** *Let $\mathcal{M} = (S, s_{in}, s_T, s_F, \delta)$ be a Turing machine. There is a term $\texttt{trans}^{\mathcal{M}}$, or simply $\texttt{trans}$, such that for every configuration C of input string $i \in \mathbb{B}^+$:*

- Final configuration*: if C is a final configuration then* $\texttt{trans}\,k\,\overline{C} \to_{det}^{\mathcal{O}(|i| \log |i|)} k\,\overline{C}$;
- Non-final configuration*: if $C \to_{\mathcal{M}} D$ then* $\texttt{trans}\,k\,\overline{C} \to_{det}^{\mathcal{O}(|i| \log |i|)} \texttt{trans}\,k\,\overline{D}$.

Proof. The transition function $\delta(b, a, s)$ is a 3-dimensional table having for coordinates:

- the current bit b on the input tape, which is actually retrieved from the input tape i and the counter n of the current input position,
- the current character a on the work tape, and
- the current state s,

The transition function is encoded as a recursive λ-term `trans` taking as argument the encodings of i, and n—to retrieve b—and a and s. It works as follows:

- It first retrieves b from n and i by applying the `lookup` function;
- It has a subterm A_b for the four values of b. The right sub-term is selected by applying the encoding $\lceil b \rceil$ of b to A_0, A_1, A_{L} and A_{R}.
- Each A_b in turn has a sub-term $B_{b,a}$ for every character $a \in \mathbb{B}_{\mathsf{W}}$, corresponding to the working tape coordinates. The right sub-term is selected by applying the encoding $\lceil a \rceil$ of the current character a on the work tape to $B_{b,0}, B_{b,1}, B_{b,\square}$.
- Each $B_{b,a}$ in turn has a subterm $C_{b,a,s}$ for every character s in S. The right sub-term is selected by applying the encoding $\lceil s \rceil$ of the current state s to $C_{b,a,s_1}, \ldots, B_{b,a,s_{|S|}}$.
- The subterm $C_{b,a,s}$ produces the (encoding of the) next configuration according to the transition function δ. If δ decreases (resp. increases) the counter for the input tape then $C_{b,a,s}$ applies `pred` (resp. `succ`) to the input counter and then applies a term corresponding to the required action on the work tape, namely:
  - S (for *stay*) if the head does not move. This case is easy, S simply produces the next configuration.
  - L if it moves left. Let $w_l = wa''$, a' the element that the transition has to write and s' the new state. Then L has a subterm $L_{a''}^{a',s'}$ for each $a'' \in \mathbb{B}_{\mathsf{W}}$ the task of which is to add a' to the right part of the work tape, remove a'' from the left part of the work tape (which becomes w), and make a'' the character in the work head position.
  - R if it moves right. Its structure is similar to the one of L.

  In order to be as modular as possible we use the definition of S, L, and R for the cases when the input head moves also for the cases where it does not move, even if this requires a useless (but harmless) additional update of the counter n.

Define

$$\texttt{transaux} := \lambda x.\lambda k'.\lambda C'.C'(\lambda i'.\lambda n'.\lambda w_l'.\lambda a'.\lambda w_r'.\lambda s'.\texttt{lookup}\, K i' n')$$

$$\texttt{trans} := \mathsf{fix}\,\texttt{transaux},$$

where:

$$
\begin{array}{rcl}
K & := & \lambda b'.b'\ A_0 A_1 A_{\mathsf{L}} A_{\mathsf{R}} a' s' x k' i' n' w_l' w_r' \\
A_b & := & \lambda a'.a' B_{b,0} B_{b,1} B_{b,\square} \\
\\
B_{b,a} & := & \lambda s'.s' C_{b,a,s_1} \dots C_{b,a,s_{|S|}} \\
C_{b,a,s} & := & \lambda x.\lambda k'.\lambda i'.\lambda n'.\lambda w_l'.\lambda w_r'. \begin{cases} k'\langle i', n' \mid w_l', \lceil a \rceil, w_r' \mid \lceil s \rceil \rangle & \text{if } s \in S_{fin} \\ Sn' & \text{if } \delta(b,a,s) = (0 \mid a', \downarrow \mid s') \\ Ln' & \text{if } \delta(b,a,s) = (0 \mid a', \leftarrow \mid s') \\ Rn' & \text{if } \delta(b,a,s) = (0 \mid a', \rightarrow \mid s') \\ \texttt{pred}Sn' & \text{if } \delta(b,a,s) = (-1 \mid a', \downarrow \mid s') \\ \texttt{pred}Ln' & \text{if } \delta(b,a,s) = (-1 \mid a', \leftarrow \mid s') \\ \texttt{pred}Rn' & \text{if } \delta(b,a,s) = (-1 \mid a', \rightarrow \mid s') \\ \texttt{succ}Sn' & \text{if } \delta(b,a,s) = (+1 \mid a', \downarrow \mid s') \\ \texttt{succ}Ln' & \text{if } \delta(b,a,s) = (+1 \mid a', \leftarrow \mid s') \\ \texttt{succ}Rn' & \text{if } \delta(b,a,s) = (+1 \mid a', \rightarrow \mid s') \end{cases} \\
S & := & \lambda n''.x k' \langle i', n'' \mid w_l', \lceil a' \rceil, w_r' \mid \lceil s' \rceil \rangle \\
L & := & \lambda n''.w_l' L_0^{s',a'} L_1^{s',a'} L_\square^{s',a'} L_\varepsilon^{s',a'} x k' i' n'' w_r' \\
R & := & \lambda n''.w_r' R_0^{s',a'} R_1^{s',a'} R_\square^{s',a'} R_\varepsilon^{s',a'} x k' i' n'' w_l' \\
L_{a''}^{s',a'} & := & \lambda w_l'.\lambda x.\lambda k'.\lambda i'.\lambda n'.\texttt{append}^{a'}(\lambda w_r'.x k' \langle i', n' \mid w_l', \lceil a'' \rceil, w_r' \mid \lceil s' \rceil \rangle) \\
L_\varepsilon^{s',a'} & := & \lambda x.\lambda k'.\lambda i'.\lambda n'.\texttt{append}^{a'}((\lambda d.\lambda w_r'.x k' \langle i', n' \mid d, \lceil \square \rceil, w_r' \mid \lceil s' \rceil \rangle)\bar{\varepsilon}) \\
R_{a''}^{s',a'} & := & \lambda w_r'.\lambda x.\lambda k'.\lambda i'.\lambda n'.\texttt{append}^{a'}(\lambda w_l'.x k' \langle i', n' \mid w_l', \lceil a'' \rceil, w_r' \mid \lceil s' \rceil \rangle) \\
R_\varepsilon^{s',a'} & := & \lambda x.\lambda k'.\lambda i'.\lambda n'.\texttt{append}^{a'}((\lambda d.\lambda w_l'.x k' \langle i', n' \mid w_l', \lceil \square \rceil, d \mid \lceil s' \rceil \rangle)\bar{\varepsilon})
\end{array}
$$

Let $C = (i, n \mid w_l, a, w_r \mid s)$. We are now going to show the details of how the λ-calculus simulates the transition function. At the level of the number of steps, the main cost is payed at the beginning, by the `lookup` function that looks up the n-th character of the input string i. The cost of one such call is $\mathcal{O}(n \log n)$, but since n can vary and $n \leq |i|$, such a cost is bound by $\mathcal{O}(|i| \log |i|)$. The cases of transition where the position on the input tape does not change have a constant cost. Those where the input position changes require to change the counter n via `pred` or `succ`, which requires $\mathcal{O}(\log n)$, itself bound by the cost $\mathcal{O}(|i| \log |i|)$ of the previous look-up.

Now, if LOOKUP $\lfloor n \rfloor i = b$ then:

$$
\begin{array}{lcl}
\mathtt{trans}\, k\, \overline{C} & = & \mathrm{fix}\, \mathtt{transaux} k \overline{C} \\
& \to^{2}_{det} & \mathtt{transaux}(\lambda z.\mathrm{fix}\, \mathtt{transaux} z) k \overline{C} \\
& = & \mathtt{transaux}(\lambda z.\mathtt{trans}\, z) k \overline{C} \\
& = & (\lambda x.\lambda k'.\lambda C'.C'(\lambda i'.\lambda n'.\lambda w_l'.\lambda a'.\lambda w_r'.\lambda s'.\mathtt{lookup}\, K i' n'))(\lambda z.\mathtt{trans}\, z) k \overline{C} \\
& \to^{3}_{det} & \overline{C}(\lambda i'.\lambda n'.\lambda w_l'.\lambda a'.\lambda w_r'.\lambda s'.\mathtt{lookup}\, K\{x \leftarrow \lambda z.\mathtt{trans}\, z\} i' n') \\
& = & \overline{(i, n \mid w_l, a, w_r \mid s)}(\lambda i'.\lambda n'.\lambda w_l'.\lambda a'.\lambda w_r'.\lambda s'.\mathtt{lookup}\, K\{x \leftarrow \lambda z.\mathtt{trans}\, z\} i' n') \\
& = & (\lambda x.x \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}}\ \lceil a \rceil\ \overline{w_r}\ \lceil s \rceil)(\lambda i'.\lambda n'.\lambda w_l'.\lambda a'.\lambda w_r'.\lambda s'.\mathtt{lookup}\, K\{x \leftarrow \lambda z.\mathtt{trans}\, z\} i' n') \\
& = & (\lambda i'.\lambda n'.\lambda w_l'.\lambda a'.\lambda w_r'.\lambda s'.\mathtt{lookup}\, K\{x \leftarrow \lambda z.\mathtt{trans}\, z\} i' n') \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}}\ \lceil a \rceil\ \overline{w_r}\ \lceil s \rceil \\
& \to^{6}_{det} & \mathtt{lookup}\, (\lambda b'.b'\ A_0 A_1 A_{\mathsf{L}} A_{\mathsf{R}} \lceil a \rceil\ \lceil s \rceil (\lambda z.\mathtt{trans}\, z) k' \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}} \overline{w_r}}) \bar{i} \overline{\lfloor n \rfloor} \\
\text{L. D.2.3} & \to^{O(|n| \log |n|)}_{det} & (\lambda b'.b'\ A_0 A_1 A_{\mathsf{L}} A_{\mathsf{R}} \lceil a \rceil\ \lceil s \rceil (\lambda z.\mathtt{trans}\, z) k' \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}} \overline{w_r}}) \lceil b \rceil \\
& \to_{det} & \lceil b \rceil\ A_0 A_1 A_{\mathsf{L}} A_{\mathsf{R}} \lceil a \rceil\ \lceil s \rceil (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}} \overline{w_r}} \\
& \to^{4}_{det} & A_b \lceil a \rceil\ \lceil s \rceil (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}}\ \overline{w_r} \\
& = & (\lambda a'.a' B_{b,0} B_{b,1} B_{b,\square}) \lceil a \rceil\ \lceil s \rceil (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}}\ \overline{w_r} \\
& \to_{det} & \lceil a \rceil B_{b,0} B_{b,1} B_{b,\square} \lceil s \rceil (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
& \to^{3}_{det} & B_{b,a} \lceil s \rceil (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
& = & (\lambda s'.s' C_{b,a,s_1} \dots C_{b,a,s_{|S|}}) \lceil s \rceil (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
& \to_{det} & \lceil s \rceil C_{b,a,s_1} \dots C_{b,a,s_{|S|}} (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
& \to^{|S|}_{det} & C_{b,a,s} (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r}
\end{array}
$$

Now, consider the following four cases, depending on the value of $\delta(b, a, s)$:

1. *Final state*: if $\delta(b, a, s)$ is undefined, then $s \in S_{fin}$ and replacing $C_{b,a,s}$ with the corresponding λ-term we obtain:

$$
\begin{array}{lcl}
& & C_{b,a,s} (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
& = & (\lambda x.\lambda k'.\lambda i'.\lambda n'.\lambda w_l'.\lambda w_r'.k' \langle i', n' \mid w_l', \lceil a \rceil, w_r' \mid \lceil s \rceil \rangle)(\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
& \to^{6}_{det} & k \langle \bar{i}, \overline{\lfloor n \rfloor} \mid \overline{w_l^{\mathrm{R}}}, \lceil a \rceil, \overline{w_r} \mid \lceil s \rceil \rangle \\
& = & k \overline{(i, n \mid w_l, a, w_r \mid s)} \\
& = & k \overline{C}
\end{array}
$$

2. *The heads do not move*: if $\delta(b, a, s) = (0 \mid a', \downarrow \mid s')$, then $D = (i, n \mid w_l, a', w_r, s')$. The simulation continues as follows:

$$
\begin{array}{lcl}
& & C_{b,a,s} (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
& = & (\lambda x.\lambda k'.\lambda i'.\lambda n'.\lambda w_l'.\lambda w_r'.Sn')(\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
& = & (\lambda x.\lambda k'.\lambda i'.\lambda n'.\lambda w_l'.\lambda w_r'.(\lambda n''.xk' \langle i', n'' \mid w_l', \lceil a' \rceil, w_r' \mid \lceil s' \rceil \rangle) n')(\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
& \to^{6}_{det} & (\lambda n''.(\lambda z.\mathtt{trans}\, z) k \langle \bar{i}, n'' \mid \overline{w_l^{\mathrm{R}}}, \lceil a' \rceil, \overline{w_r} \mid \lceil s' \rceil \rangle) \overline{\lfloor n \rfloor} \\
& \to^{2}_{det} & \mathtt{trans}\, k \langle \bar{i}, \overline{\lfloor n \rfloor} \mid \overline{w_l^{\mathrm{R}}}, \lceil a' \rceil, \overline{w_r} \mid \lceil s' \rceil \rangle \\
& = & \mathtt{trans}\, k \overline{(i, n \mid w_l, a', w_r \mid s')} \\
& = & \mathtt{trans}\, k \overline{D}
\end{array}
$$

3. *The input head does not move and the work head moves left*: if $\delta(b, a, s) = (0 \mid a', \leftarrow \mid s')$ and $w_l = wa''$ then:

$$
\begin{array}{lcl}
& & C_{b,a,s} (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
& & (\lambda x.\lambda k'.\lambda i'.\lambda n''.\lambda w_l'.\lambda w_r'.Ln')(\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
& = & (\lambda x.\lambda k'.\lambda i'.\lambda n'.\lambda w_l'.\lambda w_r'.(\lambda n''.w_l' L_0^{s',a'} L_1^{s',a'} L_\square^{s',a'} L_\varepsilon^{s',a'} xk' i' n'' w_r') n')(\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
& \to^{6}_{det} & (\lambda n''.\overline{w_l^{\mathrm{R}}} L_0^{s',a'} L_1^{s',a'} L_\square^{s',a'} L_\varepsilon^{s',a'} (\lambda z.\mathtt{trans}\, z) k \bar{i} n'' \overline{w_r}) \overline{\lfloor n \rfloor} \\
& \to_{det} & \overline{w_l^{\mathrm{R}}} L_0^{s',a'} L_1^{s',a'} L_\square^{s',a'} L_\varepsilon^{s',a'} (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor} \overline{w_r}
\end{array}
$$

Two sub-cases, depending on whether w_l is an empty or a compound string.

(a) *w_l is the compound string wa''.* Then $w_l^{\mathrm{R}} = a''w^{\mathrm{R}}$ and $D = (i, n \mid w_l, a'', a'w_r \mid s')$. The simulation continues as follows:

$$
\begin{array}{lll}
 & = & \overline{a''w^{\mathrm{R}}}L_0^{s',a'}L_1^{s',a'}L_{\square}^{s',a'}L_{\varepsilon}^{s',a'}(\lambda z.\mathtt{trans}\,z)k\bar{i}\overline{\lfloor n\rfloor}\overline{w_r} \\
 & \to_{det}^{4} & L_{a''}^{s',a'}\overline{w^{\mathrm{R}}}(\lambda z.\mathtt{trans}\,z)k\bar{i}\overline{\lfloor n\rfloor}\overline{w_r} \\
 & = & (\lambda w_l'.\lambda x.\lambda k'.\lambda i'.\lambda n'.\mathtt{append}^{a'}(\lambda w_r'.xk'\langle i', n' \mid w_l', \lceil a''\rceil, w_r' \mid \lceil s'\rceil\rangle))\overline{w^{\mathrm{R}}}(\lambda z.\mathtt{trans}\,z)k\bar{i}\overline{\lfloor n\rfloor}\overline{w_r} \\
 & \to_{det}^{5} & \mathtt{append}^{a'}(\lambda w_r'.(\lambda z.\mathtt{trans}\,z)k\langle\bar{i}, \overline{\lfloor n\rfloor} \mid \overline{w^{\mathrm{R}}}, \lceil a''\rceil, w_r' \mid \lceil s'\rceil\rangle)\overline{w_r} \\
\text{L. D.1.2} & \to_{det}^{O(1)} & (\lambda w_r'.(\lambda z.\mathtt{trans}\,z)k\langle\bar{i}, \overline{\lfloor n\rfloor} \mid \overline{w^{\mathrm{R}}}, \lceil a''\rceil, w_r' \mid \lceil s'\rceil\rangle)\overline{a'w_r} \\
 & \to_{det}^{2} & \mathtt{trans}\,k\langle\bar{i}, \overline{\lfloor n\rfloor} \mid \overline{w^{\mathrm{R}}}, \lceil a''\rceil, \overline{a'w_r} \mid \lceil s'\rceil\rangle \\
 & = & \mathtt{trans}\,k\overline{(i, n \mid w, a'', a'w_r \mid s')} \\
 & = & \mathtt{trans}\,k\overline{D}
\end{array}
$$

(b) *w_l is the empty string ε.* Then $D = (i, n \mid \varepsilon, \square, a'w_r \mid s')$. The simulation continues as follows:

$$
\begin{array}{lll}
 & = & \bar{\varepsilon}L_0^{s',a'}L_1^{s',a'}L_{\square}^{s',a'}L_{\varepsilon}^{s',a'}(\lambda z.\mathtt{trans}\,z)k\bar{i}\overline{\lfloor n\rfloor}\overline{w_r} \\
 & \to_{det}^{4} & L_{\varepsilon}^{s',a'}(\lambda z.\mathtt{trans}\,z)k\bar{i}\overline{\lfloor n\rfloor}\overline{w_r} \\
 & = & (\lambda x.\lambda k'.\lambda i'.\lambda n'.\mathtt{append}^{a'}((\lambda d.\lambda w_r.xk'\langle i', n' \mid d, \lceil\square\rceil, w_r \mid \lceil s'\rceil\rangle)\bar{\varepsilon}))(\lambda z.\mathtt{trans}\,z)k\bar{i}\overline{\lfloor n\rfloor}\overline{w_r} \\
 & \to_{det}^{5} & \mathtt{append}^{a'}((\lambda d.\lambda w_r'.(\lambda z.\mathtt{trans}\,z)k\langle\bar{i}, \overline{\lfloor n\rfloor} \mid d, \lceil\square\rceil, w_r' \mid \lceil s'\rceil\rangle)\bar{\varepsilon})\overline{w_r} \\
\text{L. D.1.2} & \to_{det}^{O(1)} & ((\lambda d.\lambda w_r'.(\lambda z.\mathtt{trans}\,z)k\langle\bar{i}, \overline{\lfloor n\rfloor} \mid d, \lceil\square\rceil, w_r' \mid \lceil s'\rceil\rangle)\bar{\varepsilon})\overline{a'w_r} \\
 & \to_{det}^{3} & \mathtt{trans}\,k\langle\bar{i}, \overline{\lfloor n\rfloor} \mid \bar{\varepsilon}, \lceil\square\rceil, \overline{a'w_r} \mid \lceil s'\rceil\rangle \\
 & = & \mathtt{trans}\,k\overline{(i, n \mid \varepsilon, \square, a'w_r \mid s')} \\
 & = & \mathtt{trans}\,k\overline{D}
\end{array}
$$

4. *The input head does not move and the work head moves right*: if $\delta(b, a, s) = (0 \mid a', \to \mid s')$ and $w_r = a''w$ then:

$$
\begin{array}{lll}
 & & C_{b,a,s}(\lambda z.\mathtt{trans}\,z)k\bar{i}\overline{\lfloor n\rfloor w_l^{\mathrm{R}}}\overline{w_r} \\
 & = & (\lambda x.\lambda k'.\lambda i'.\lambda n'.\lambda w_l'.\lambda w_r'.Rn')(\lambda z.\mathtt{trans}\,z)k\bar{i}\overline{\lfloor n\rfloor w_l^{\mathrm{R}}}\overline{w_r} \\
 & = & (\lambda x.\lambda k'.\lambda i'.\lambda n'.\lambda w_l'.\lambda w_r'.(\lambda n''.w_r'R_0^{s',a'}R_1^{s',a'}R_{\square}^{s',a'}R_{\varepsilon}^{s',a'}xk'i'n''w_l')n')(\lambda z.\mathtt{trans}\,z)k\bar{i}\overline{\lfloor n\rfloor w_l^{\mathrm{R}}}\overline{w_r} \\
 & \to_{det}^{6} & (\lambda n''.\overline{w_r}R_0^{s',a'}R_1^{s',a'}R_{\square}^{s',a'}R_{\varepsilon}^{s',a'}(\lambda z.\mathtt{trans}\,z)k\bar{i}n''\overline{w_l^{\mathrm{R}}})\overline{\lfloor n\rfloor} \\
 & \to_{det} & \overline{w_r}R_0^{s',a'}R_1^{s',a'}R_{\square}^{s',a'}R_{\varepsilon}^{s',a'}(\lambda z.\mathtt{trans}\,z)k\bar{i}\overline{\lfloor n\rfloor w_l^{\mathrm{R}}}
\end{array}
$$

Two sub-cases, depending on whether w_l is an empty or a compound string.

(a) *w_r is the compound string $a''w$.* Then $D = (i, n \mid w_l a', a'', w \mid s')$. The simulation continues as follows:

$$
\begin{array}{lll}
 & = & \overline{a''w}R_0^{s',a'}R_1^{s',a'}R_{\square}^{s',a'}R_{\varepsilon}^{s',a'}(\lambda z.\mathtt{trans}\,z)k\bar{i}\overline{\lfloor n\rfloor w_l^{\mathrm{R}}} \\
 & \to_{det}^{4} & R_{a''}^{s',a'}\overline{w}(\lambda z.\mathtt{trans}\,z)k\bar{i}\overline{\lfloor n\rfloor w_l^{\mathrm{R}}} \\
 & = & (\lambda w_r'.\lambda x.\lambda k'.\lambda i'.\lambda n'.\mathtt{append}^{a'}(\lambda w_l'.xk'\langle i', n' \mid w_l', \lceil a''\rceil, w_r' \mid \lceil s'\rceil\rangle))\overline{w}(\lambda z.\mathtt{trans}\,z)k\bar{i}\overline{\lfloor n\rfloor w_l^{\mathrm{R}}} \\
 & \to_{det}^{5} & \mathtt{append}^{a'}(\lambda w_l'.(\lambda z.\mathtt{trans}\,z)k\langle\bar{i}, \overline{\lfloor n\rfloor} \mid w_l', \lceil a''\rceil, w \mid \lceil s'\rceil\rangle)\overline{w_l^{\mathrm{R}}} \\
\text{L. D.1.2} & \to_{det}^{O(1)} & (\lambda w_l'.(\lambda z.\mathtt{trans}\,z)k\langle\bar{i}, \overline{\lfloor n\rfloor} \mid w_l', \lceil a''\rceil, w \mid \lceil s'\rceil\rangle)\overline{a'w_l^{\mathrm{R}}} \\
 & \to_{det}^{2} & \mathtt{trans}\,k\langle\bar{i}, \overline{\lfloor n\rfloor} \mid \overline{a'w_l^{\mathrm{R}}}, \lceil a''\rceil, w \mid \lceil s'\rceil\rangle \\
 & = & \mathtt{trans}\,k\langle\bar{i}, \overline{\lfloor n\rfloor} \mid \overline{(w_l a')^{\mathrm{R}}}, \lceil a''\rceil, w \mid \lceil s'\rceil\rangle \\
 & = & \mathtt{trans}\,k\overline{(i, n \mid w_l a', a'', w \mid s')} \\
 & = & \mathtt{trans}\,k\overline{D}
\end{array}
$$

(b) *w_r is the empty string ε.* Then $D = (i, n \mid w_l a', \square, \varepsilon \mid s')$. The simulation continues as follows:

$$
\begin{array}{lll}
 & = & \bar{\varepsilon} R_0^{s',a'} R_1^{s',a'} R_\square^{s',a'} R_\varepsilon^{s',a'} (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \\
 & \to_{det}^{4} & R_\varepsilon^{s',a'} (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \\
 & = & (\lambda x.\lambda k'.\lambda i'.\lambda n'.\mathtt{append}^{a'}((\lambda d.\lambda w_l'.x k' \langle i', n' \mid w_l', \lceil \square \rceil, d \mid \lceil s' \rceil \rangle)\bar{\varepsilon}))(\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \\
 & \to_{det}^{5} & \mathtt{append}^{a'}((\lambda d.\lambda w_l'.(\lambda z.\mathtt{trans}\, z) k \langle \bar{i}, \overline{\lfloor n \rfloor} \mid w_l', \lceil \square \rceil, d \mid \lceil s' \rceil \rangle)\bar{\varepsilon}) \overline{w_l^{\mathrm{R}}} \\
\text{L. D.1.2} & \to_{det}^{O(1)} & ((\lambda d.\lambda w_l'.(\lambda z.\mathtt{trans}\, z) k \langle \bar{i}, \overline{\lfloor n \rfloor} \mid w_l', \lceil \square \rceil, d \mid \lceil s' \rceil \rangle)\bar{\varepsilon}) \overline{a' w_l^{\mathrm{R}}} \\
 & \to_{det}^{3} & \mathtt{trans}\, k \langle \bar{i}, \overline{\lfloor n \rfloor} \mid \overline{a' w_l^{\mathrm{R}}}, \lceil \square \rceil, \bar{\varepsilon} \mid \lceil s' \rceil \rangle \\
 & = & \mathtt{trans}\, k \langle \bar{i}, \overline{\lfloor n \rfloor} \mid \overline{(w_l a')^{\mathrm{R}}}, \lceil \square \rceil, \bar{\varepsilon} \mid \lceil s' \rceil \rangle \\
 & = & \mathtt{trans}\, k \overline{(i, n \mid w_l a', \square, \varepsilon \mid s')} \\
 & = & \mathtt{trans}\, k \overline{D}
\end{array}
$$

5. *The input head moves left and the work head does not move*: if $\delta(b,a,s) = (-1 \mid a', \downarrow \mid s')$, then $D = (i, n-1 \mid w_l, a', w_r, s')$. The simulation continues as follows:

$$
\begin{array}{lll}
 & & C_{b,a,s}(\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
 & = & (\lambda x.\lambda k'.\lambda i'.\lambda n'.\lambda w_l'.\lambda w_r'.\mathtt{pred}\, S (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
 & = & (\lambda x.\lambda k'.\lambda i'.\lambda n'.\lambda w_l'.\lambda w_r'.\mathtt{pred}\, (\lambda n''.x k' \langle i', n'' \mid w_l', \lceil a' \rceil, w_r' \mid \lceil s' \rceil \rangle) n') (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
 & \to_{det}^{6} & \mathtt{pred}\, (\lambda n''.(\lambda z.\mathtt{trans}\, z) k \langle \bar{i}, n'' \mid \overline{w_l^{\mathrm{R}}}, \lceil a' \rceil, \overline{w_r} \mid \lceil s' \rceil \rangle) \overline{\lfloor n \rfloor} \\
\text{L. D.2.2} & \to_{det}^{O(\log n)} & (\lambda n''.(\lambda z.\mathtt{trans}\, z) k \langle \bar{i}, n'' \mid \overline{w_l^{\mathrm{R}}}, \lceil a' \rceil, \overline{w_r} \mid \lceil s' \rceil \rangle) \overline{\lfloor n-1 \rfloor} \\
 & = & \mathtt{trans}\, k \overline{(i, \lfloor n-1 \rfloor \mid w_l, a', w_r \mid s')} \\
 & = & \mathtt{trans}\, k \overline{(i, \lfloor n-1 \rfloor \mid w_l, a', w_r \mid s')} \\
 & = & \mathtt{trans}\, k \overline{D}
\end{array}
$$

6. *The input head moves left and the work head moves left*: if $\delta(b,a,s) = (-1 \mid a', \leftarrow \mid s')$ and $w_l = wa''$, then $D = (i, n-1 \mid w, a'', a' w_r, s')$. The simulation continues as follows:

$$
\begin{array}{lll}
 & & C_{b,a,s}(\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
 & = & (\lambda x.\lambda k'.\lambda i'.\lambda n'.\lambda w_l'.\lambda w_r'.\mathtt{pred}\, L n') (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
 & = & (\lambda x.\lambda k'.\lambda i'.\lambda n'.\lambda w_l'.\lambda w_r'.\mathtt{pred}\, (\lambda n''.w_l' L_0^{s',a'} L_1^{s',a'} L_\square^{s',a'} L_\varepsilon^{s',a'} x k' i' n'' w_r') n') (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
 & \to_{det}^{6} & \mathtt{pred}\, (\lambda n''.\overline{w_l^{\mathrm{R}}} L_0^{s',a'} L_1^{s',a'} L_\square^{s',a'} L_\varepsilon^{s',a'} (\lambda z.\mathtt{trans}\, z) k \bar{i} n'' \overline{w_r}) \overline{\lfloor n \rfloor} \\
\text{L. D.2.2} & \to_{det}^{O(\log n)} & (\lambda n''.\overline{w_l^{\mathrm{R}}} L_0^{s',a'} L_1^{s',a'} L_\square^{s',a'} L_\varepsilon^{s',a'} (\lambda z.\mathtt{trans}\, z) k \bar{i} n'' \overline{w_r}) \overline{\lfloor n-1 \rfloor} \\
 & \to_{det} & \overline{w_l^{\mathrm{R}}} L_0^{s',a'} L_1^{s',a'} L_\square^{s',a'} L_\varepsilon^{s',a'} (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n-1 \rfloor} \overline{w_r}
\end{array}
$$

And then the case continues with the two sub-cases of case 3 (input head does not move and work head moves left), with the only difference that $\overline{\lfloor n \rfloor}$ is replaced by $\overline{\lfloor n-1 \rfloor}$.

7. *The input head moves left and the work head moves right*: if $\delta(b,a,s) = (-1 \mid a', \leftarrow \mid s')$ and $w_l = wa''$, then $D = (i, n-1 \mid w, a'', a' w_r, s')$. The simulation continues as follows:

$$
\begin{array}{lll}
 & & C_{b,a,s}(\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
 & = & (\lambda x.\lambda k'.\lambda i'.\lambda n'.\lambda w_l'.\lambda w_r'.\mathtt{pred}\, R n') (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
 & = & (\lambda x.\lambda k'.\lambda i'.\lambda n'.\lambda w_l'.\lambda w_r'.\mathtt{pred}(\lambda n''.w_r' R_0^{s',a'} R_1^{s',a'} R_\square^{s',a'} R_\varepsilon^{s',a'} x k' i' n'' w_l') n') (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n \rfloor w_l^{\mathrm{R}}} \overline{w_r} \\
 & \to_{det}^{6} & \mathtt{pred}(\lambda n''.\overline{w_r} R_0^{s',a'} R_1^{s',a'} R_\square^{s',a'} R_\varepsilon^{s',a'} (\lambda z.\mathtt{trans}\, z) k \bar{i} n'' \overline{w_l^{\mathrm{R}}}) \overline{\lfloor n \rfloor} \\
\text{L. D.2.2} & \to_{det}^{O(\log n)} & (\lambda n''.\overline{w_r} R_0^{s',a'} R_1^{s',a'} R_\square^{s',a'} R_\varepsilon^{s',a'} (\lambda z.\mathtt{trans}\, z) k \bar{i} n'' \overline{w_l^{\mathrm{R}}}) \overline{\lfloor n-1 \rfloor} \\
 & \to_{det} & \lambda n''.\overline{w_r} R_0^{s',a'} R_1^{s',a'} R_\square^{s',a'} R_\varepsilon^{s',a'} (\lambda z.\mathtt{trans}\, z) k \bar{i} \overline{\lfloor n-1 \rfloor w_l^{\mathrm{R}}}
\end{array}
$$

And then the case continues with the two sub-cases of case 4 (input head does not move and work head moves right), with the only difference that $\overline{\lfloor n \rfloor}$ is replaced by $\overline{\lfloor n-1 \rfloor}$.

8. *The input head moves right and the work head does not move*: exactly as case 5 (input head *left*, work head does not move) just replacing `pred` with `succ` and using Lemma D.2.1 instead of Lemma D.2.2.

9. *The input head moves right and the work head moves left*: exactly as case 6 (input head *left*, work head left) just replacing `pred` with `succ` and using Lemma D.2.1 instead of Lemma D.2.2.
10. *The input head moves right and the work head moves right*: exactly as case 7 (input head *right*, work head right) just replacing `pred` with `succ` and using Lemma D.2.1 instead of Lemma D.2.2.

□

Straightforward inductions on the length of executions provide the following corollaries.

Corollary D.3.4 (Executions)**.** *Let $\mathcal{M}$ be a Turing machine. Then there exist a term* `trans` *encoding $\mathcal{M}$ as given by Lemma D.3.3 such that for every configuration C of input string $i \in \mathbb{B}^+$*

1. Finite computation*: if D is a final configuration reachable from C in n transition steps then there exists a derivation ρ such that $\rho : \mathtt{trans}\, k\lceil C\rceil \to_{det}^{\mathcal{O}((n+1)|i|\log|i|)} k\lceil D\rceil$;*
2. Diverging computation*: if there is no final configuration reachable from C then $\mathtt{trans}\, k\lceil C\rceil$ diverges.*

The simulation theorem. We now have all the ingredients for the final theorem of this note.

Theorem D.3.5 (Simulation)**.** *Let $f : \mathbb{B}^* \to \mathbb{B}$ a function computed by a Turing machine $\mathcal{M}$ in time $T_{\mathcal{M}}$. Then there is an encoding $\lceil\cdot\rceil$ into $\Lambda_{\mathtt{det}}$ of $\mathbb{B}$, strings, and Turing machines over $\mathbb{B}$ such that for every $i \in \mathbb{B}^+$, there exists ρ such that $\rho : \lceil\mathcal{M}\rceil\lceil i\rceil \to_{det}^{n} \lceil f(i)\rceil$ where $n = \Theta((T_{\mathcal{M}}(|i|)+1)\cdot|i|\cdot\log|i|)$.*

Proof. Morally, the term is simply

$$\overline{\mathcal{M}} := \mathtt{init}(\mathtt{trans}(\mathtt{final}(\lambda w.w))$$

where $\lambda w.w$ plays the role of the initial continuation.

Such a term however does not belong to the deterministic λ-calculus, because the right subterms of applications are not always values. The solution is simple, it is enough to η-expand the arguments. Thus, define

$$\overline{\mathcal{M}} := \mathtt{init}(\lambda y.\mathtt{trans}(\lambda x.\mathtt{final}(\lambda w.w)x)y)$$

Then

$$\begin{array}{lll}
\overline{\mathcal{M}}\lceil i\rceil & = & \\
\mathtt{init}(\lambda y.\mathtt{trans}(\lambda x.\mathtt{final}(\lambda w.w)x)y)\lceil i\rceil & \to_{det}^{\Theta(1)} & \text{(by L. D.3.1)} \\
(\lambda y.\mathtt{trans}(\lambda x.\mathtt{final}(\lambda w.w)x)y)\lceil C_{\mathtt{in}}^{\mathcal{M}}(s)\rceil & \to_{det} & \\
\mathtt{trans}(\lambda x.\mathtt{final}(\lambda w.w)x)\lceil C_{\mathtt{in}}^{\mathcal{M}}(s)\rceil & \to_{det}^{\Theta((T_{\mathcal{M}}(|i|)+1)\cdot|i|\cdot\log|i|)} & \text{(by Cor. D.3.4)} \\
(\lambda x.\mathtt{final}(\lambda w.w)x)\lceil C_{\mathtt{fin}}(f(i))\rceil & \to_{det} & \\
\mathtt{final}(\lambda w.w)\lceil C_{\mathtt{fin}}(f(i))\rceil & \to_{det}^{\Theta(|S|)} & \text{(by L. D.3.2)} \\
(\lambda w.w)\lceil f(i)\rceil & \to_{det} & \\
\lceil f(i)\rceil & &
\end{array}$$

□

Appendix E

Execution of the Encoding of TMs on the Space KAM

This appendix is devoted to the proof of the main theorem of Chapter 8, *i.e.* the space reasonable simulation of TMs into the λ-calculus (better, the Space KAM). It is a boring proof, where we simply execute the image of the encoding of TMs into the λ-calculus with the Space KAM.

First, we need to understand how a TM configuration is represented in the Space KAM, *i.e.* how it is mapped to environments and closures.

Definition E.0.1. *A configuration C of a TM is represented as a KAM closure C^{K} in the following way:*

$$(i,n,s,a,r,s)^{\mathrm{K}} := (\langle f, c, m, \lceil a \rceil, d, \lceil s \rceil \rangle, [f \leftarrow (\bar{i}, \epsilon)], [c \leftarrow n^{\mathrm{K}}], [m \leftarrow s^{\mathrm{K}}], [d \leftarrow r^{\mathrm{K}}])$$

where $s^{\mathrm{K}} = \begin{cases} (\lceil \varepsilon \rceil, \epsilon) & \text{if } s = \varepsilon \\ (\lambda x_1 . \ldots . \lambda x_{|\Sigma|} . \lambda y . x_{i_a} z, [z \leftarrow r^{\mathrm{K}}]) & \text{if } s = a_i r \end{cases}$

We observe that this representation preserves the space consumption, *i.e.* it is reasonable.

Lemma E.0.2. *Let $C := (i,n,s,a,r,s)$ be a configuration of a Turing machine and $|C| := |s| + |r|$ its space consumption. Then $|C^{\mathrm{K}}| = \Theta(|C| + \log(|i|))$.*

In this lemma, we have already considered that the size of pointers inside n^{K}, s^{K}, r^{K} is constant and that $n \leq \log |i|$.

Now we are able to prove the theorem. A series of intermediate lemmata, about the different combinators used in the encoding (`init`, `final`, `trans`), are necessary. They are stated and proved below the main statement . By $\to^*_f$, we mean that the space consumption of that series of transitions is f.

Theorem E.0.3 (TM are simulated by the Space KAM in reasonable space)**.** *There is an encoding $\bar{\cdot}$ of log-sensitive TM into $\Lambda_{\mathtt{det}}$ such that if the run ρ of the TM M on input $i \in \mathbb{B}^*$:*

1. Termination*: ends in s_b with $b \in \mathbb{B}$, then there is a complete sequence $\sigma : \overline{M}\,\bar{i} \to^n_{det} \bar{b}$ where $n = \Theta((T_{TM}(\rho) + 1) \cdot |i| \cdot \log |i|)$.*
2. Divergence*: diverges, then $\overline{M}\,\bar{i}$ is $\to_{det}$-divergent.*
3. Space KAM*: the space used by the Space KAM to simulate the evaluation of point 1 is $\mathcal{O}(S_{TM}(\rho) + \log |i|)$ if $\overline{M}$ and $\bar{i}$ have separate address spaces.*

Proof. The first two points are proved in the section devoted to the encoding of TMs into the λ-calculus. We concentrate on the third point.

We simply evaluate $\overline{M}\,\bar{i}$ with the Space KAM.

Term	Env	Stack	
$\overline{M}\,\bar{i} := \mathtt{init}(\mathtt{trans}(\mathtt{final}(\lambda x.x)))\bar{i}$	ϵ	ϵ	$\to^*_{\mathcal{O}(\log(\lvert i\rvert))}$ (Lemma E.0.5)
$\mathtt{trans}(\mathtt{final}(\lambda x.x))$	ϵ	$C_{\mathtt{in}}(i)^{\mathrm{K}}$	$\to^*_{\mathcal{O}(S_{\mathrm{TM}}(\rho)+\log\lvert i\rvert)}$ (Lemma E.0.9)
$\mathtt{final}(\lambda x.x)$	ϵ	D^{K}	$\to^*_{\mathcal{O}(S_{\mathrm{TM}}(\rho)+\log\lvert i\rvert)}$ (Lemma E.0.6)
$\bar{b}$	ϵ	ϵ	

□

Fix-Point. In the following we will often use the execution of the fix-point combinator. For this reason, we encapsulate its execution by the Space KAM in a lemma.

Lemma E.0.4. *For each term u, $(\theta,\epsilon,(\theta,\epsilon)\cdot(u,\epsilon)\cdot\pi) \to^{\mathcal{O}(1)}_{SpKAM} (u,\epsilon,\mathsf{fix}^{\mathrm{K}}\cdot\pi)$ where $\mathsf{fix}^{\mathrm{K}} := (xxy,[y\leftarrow(u,e)]\cdot[x\leftarrow(\theta,\epsilon)])$ consuming space $\mathcal{O}(|e|+|\pi|+\log(|u|))$.*

Proof.

Term	Env	Stack
$\theta := \lambda x.\lambda y.y(xxy)$	ϵ	$(\theta,\epsilon)\cdot(u,\epsilon)\cdot\pi$
$\lambda y.y(xxy)$	$[x\leftarrow(\theta,\epsilon)]$	$(u,e)\cdot\pi$
$y(xxy)$	$[y\leftarrow(u,e)]\cdot[x\leftarrow(\theta,\epsilon)]$	π
		$\overbrace{(xxy,[y\leftarrow(u,e)]\cdot[x\leftarrow(\theta,\epsilon)])}^{\mathsf{fix}^{\mathrm{K}}}\cdot\pi$
y	$[y\leftarrow(u,e)]$	
u	e	$\mathsf{fix}^{\mathrm{K}}\cdot\pi$

□

Init and Final.

Lemma E.0.5. $(\mathtt{init}\,k\,\bar{i},\epsilon,\epsilon) \to^{\mathcal{O}(1)}_{SpKAM} (k,\epsilon,C_{\mathtt{in}}(i)^{\mathrm{K}})$ *and consumes space* $\Theta(\log(|i|))$.

Proof.

Term	Env	Stack	
$\mathtt{init}\,k\,\bar{i}$	ϵ	ϵ	
$\mathtt{init}\,k$	ϵ	$(\bar{i},\epsilon)$	
$\mathtt{init} := (\lambda d.\lambda e.\lambda f.\lambda k'.\lambda i'.k'\langle i',d\,\vert\,e,\lceil\Box\rceil,f\,\vert\,\lceil s_{in}\rceil\rangle)\overline{[0]}\,\bar{\varepsilon}\bar{\varepsilon}$	ϵ	$\overbrace{(k,\epsilon)}^{k^{\mathrm{K}}}\cdot\overbrace{(\bar{i},\epsilon)}^{\bar{i}^{\mathrm{K}}}$	$\to^3$
$\lambda d.\lambda e.\lambda f.\lambda k'.\lambda i'.k'\langle i',d\,\vert\,e,\lceil\Box\rceil,f\,\vert\,\lceil s_{in}\rceil\rangle$	ϵ	$\overbrace{(\overline{[0]},\epsilon)}^{\overline{[0]}^{\mathrm{K}}}\cdot\overbrace{(\bar{\varepsilon},\epsilon)}^{\bar{\varepsilon}^{\mathrm{K}}}\cdot(\bar{\varepsilon},\epsilon)\cdot k^{\mathrm{K}}\cdot\bar{i}^{\mathrm{K}}$	$\to^5$
$k'\langle i',d\,\vert\,e,\lceil\Box\rceil,f\,\vert\,\lceil s_{in}\rceil\rangle$	$[k'\leftarrow k^{\mathrm{K}}]\cdot\overbrace{[i'\leftarrow\bar{i}^{\mathrm{K}}]\cdot[d\leftarrow\overline{[0]}^{\mathrm{K}}]\cdot[e\leftarrow\bar{\varepsilon}^{\mathrm{K}}]\cdot[f\leftarrow\bar{\varepsilon}^{\mathrm{K}}]}^{C_{\mathtt{in}}(i)^E}$	ϵ	
k'	$[k'\leftarrow k^{\mathrm{K}}]$	$(\langle i',d\,\vert\,e,\lceil\Box\rceil,f\,\vert\,\lceil s_{in}\rceil\rangle,C_{\mathtt{in}}(i)^E)$	
k	ϵ	$\underbrace{(\langle i',d\,\vert\,e,\lceil\Box\rceil,f\,\vert\,\lceil s_{in}\rceil\rangle,C_{\mathtt{in}}(i)^E)}_{C_{\mathtt{in}}(i)^{\mathrm{K}}}$	

The space bound is immediate by inspecting the execution. □

Lemma E.0.6. *Let C be a final configuration, i.e. $C := (i,n,s,a,r,s_{fin})$ where $s_{fin}\in S_{fin}$. Then*

$$(\mathtt{final}(\lambda x.x),\epsilon,C^{\mathrm{K}}) \to^{\mathcal{O}(1)}_{SpKAM} \begin{cases}(\lambda x.\lambda y.x,\epsilon,\epsilon) & \text{if } s_{fin}=s_T\\ (\lambda x.\lambda y.y,\epsilon,\epsilon) & \text{if } s_{fin}=s_F\end{cases}$$

Moreover, the space consumption is $\Theta(|C^{\mathrm{K}}|)$.

Proof. We execute the Space KAM. Let us define $t := \lambda i'.\lambda n'.\lambda w'_l.\lambda a'.\lambda w'_r.\lambda s'.s'N_1\dots N_{|S|}k'$.

Term	Env	Stack	
$\mathtt{final}(\lambda x.x)$	ϵ	C^{K}	
$\mathtt{final} := \lambda k'.\lambda C'.C't$	ϵ	$(\lambda x.x,\epsilon)\cdot C^{\mathrm{K}}$	$\to^2$
$C't$	$[C'\leftarrow C^{\mathrm{K}}]\cdot[k'\leftarrow(\lambda x.x,\epsilon)]$	ϵ	
C'	$[C'\leftarrow C^{\mathrm{K}}]$	$(t,[k'\leftarrow(\lambda x.x,\epsilon)])$	
$C^{\mathrm{K}} := \lambda x.xfcm\lceil a\rceil d\lceil s_{fin}\rceil$	$[f\leftarrow(\bar{i},\epsilon)],[c\leftarrow n^{\mathrm{K}}],[m\leftarrow s^{\mathrm{K}}],[d\leftarrow r^{\mathrm{K}}]$	$(t,[k'\leftarrow(\lambda x.x,\epsilon)])$	
$xfcm\lceil a\rceil d\lceil s_{fin}\rceil$	$[x\leftarrow(t,[k'\leftarrow(I,\epsilon)])]\cdot[f\leftarrow(\bar{i},\epsilon)],[c\leftarrow n^{\mathrm{K}}],[m\leftarrow s^{\mathrm{K}}],[d\leftarrow r^{\mathrm{K}}]$	ϵ	$\to^6$
x	$[x\leftarrow(t,[k'\leftarrow(\lambda x.x,\epsilon)])]$	$(\bar{i},\epsilon)\cdot n^{\mathrm{K}}\cdot s^{\mathrm{K}}\cdot(\lceil a\rceil,\epsilon)\cdot r^{\mathrm{K}}\cdot(\lceil s_{fin}\rceil,\epsilon)$	
$t := \lambda i'.\lambda n'.\lambda w'_l.\lambda a'.\lambda w'_r.\lambda s'.s'N_1\dots N_{\vert S\vert}k'$	$[k'\leftarrow(\lambda x.x,\epsilon)]$	$(\bar{i},\epsilon)\cdot n^{\mathrm{K}}\cdot s^{\mathrm{K}}\cdot(\lceil a\rceil,\epsilon)\cdot r^{\mathrm{K}}\cdot(\lceil s_{fin}\rceil,\epsilon)$	$\to^6$
$q'N_1\dots N_{\vert Q\vert}k'$	$[s'\leftarrow(\lceil s_{fin}\rceil,\epsilon)]\cdot[k'\leftarrow(\lambda x.x,\epsilon)]$	ϵ	$\to^{1+\vert S\vert}$
s'	$[s'\leftarrow(\lceil s_{fin}\rceil,\epsilon)]$	$(N_i,\epsilon)_{1\le i\le\vert S\vert}\cdot(\lambda x.x,\epsilon)$	
$\lceil s_{fin}\rceil := \lambda x_1\dots\lambda x_{\vert S\vert}.x_i$	ϵ	$(N_i,\epsilon)_{1\le i\le\vert S\vert}\cdot(\lambda x.x,\epsilon)$	

Two cases. If $s_{fin} = s_T$, then:

Term	Env	Stack	
$\lceil s_T \rceil := \lambda x_1 \dots \lambda x_{\lvert S \rvert}.x_i$	ϵ	$(N_i, \epsilon)_{1 \le i \le \lvert S \rvert} \cdot (\lambda x.x, \epsilon)$	$\to^{\lvert S \rvert}$
x_i	$[x_i \leftarrow (N_i, \epsilon)]$	$(\lambda x.x, \epsilon)$	
$N_i := \lambda k'.k'(\lambda x.\lambda y.x)$	ϵ	$(\lambda x.x, \epsilon)$	
$k'(\lambda x.\lambda y.x)$	$[k' \leftarrow (\lambda x.x, \epsilon)]$	ϵ	
k'	$[k' \leftarrow (\lambda x.x, \epsilon)]$	$(\lambda x.\lambda y.x, \epsilon)$	
$\lambda x.x$	ϵ	$(\lambda x.\lambda y.x, \epsilon)$	
x	$[x \leftarrow (\lambda x.\lambda y.x, \epsilon)]$	ϵ	
$\lambda x.\lambda y.x$	ϵ	ϵ	

If $s_{fin} = s_F$, then:

Term	Env	Stack	
$\lceil s_F \rceil := \lambda x_1 \dots \lambda x_{\lvert S \rvert}.x_i$	ϵ	$(N_i, \epsilon)_{1 \le i \le \lvert S \rvert} \cdot (\lambda x.x, \epsilon)$	$\to^{\lvert S \rvert}$
x_i	$[x_i \leftarrow (N_i, \epsilon)]$	$(\lambda x.x, \epsilon)$	
$N_i := \lambda k'.k'(\lambda x.\lambda y.y)$	ϵ	$(\lambda x.x, \epsilon)$	
$k'(\lambda x.\lambda y.y)$	$[k' \leftarrow (\lambda x.x, \epsilon)]$	ϵ	
k'	$[k' \leftarrow (\lambda x.x, \epsilon)]$	$(\lambda x.\lambda y.y, \epsilon)$	
$\lambda x.x$	ϵ	$(\lambda x.\lambda y.y, \epsilon)$	
x	$[x \leftarrow (\lambda x.\lambda y.y, \epsilon)]$	ϵ	
$\lambda x.\lambda y.y$	ϵ	ϵ	

The space bound is immediate by inspecting the execution. □

Transition Function.

Lemma E.0.7. $(\mathtt{trans}\, k, \epsilon, C_{\mathtt{in}}(i)^{\mathrm{K}}) \to^{\mathcal{O}(1)}_{SpKAM} (\theta, \epsilon, (\theta, \epsilon) \cdot (\mathtt{transaux}, \epsilon) \cdot k^{\mathrm{K}} \cdot C_{\mathtt{in}}(i)^{\mathrm{K}})$ *in space* $\mathcal{O}(\log(\lvert i \rvert))$.

Proof.

Term	Env	Stack	
$\mathtt{trans} k$	ϵ	$C_{\mathtt{in}}(i)^{\mathrm{K}}$	
$\mathtt{trans} := \mathsf{fix}\, \mathtt{transaux}$	ϵ	$k^{\mathrm{K}} \cdot C_{\mathtt{in}}(i)^{\mathrm{K}}$	Lemma E.0.4
θ	ϵ	$(\theta, \epsilon) \cdot (\mathtt{transaux}, \epsilon) \cdot k^{\mathrm{K}} \cdot C_{\mathtt{in}}(i)^{\mathrm{K}}$	

□

Lemma E.0.8. *Let C be a Turing machine configuration. Then:*

- *if C is a final configuration, then* $(\theta, \epsilon, (\theta, \epsilon) \cdot (\mathtt{transaux}, \epsilon) \cdot k^{\mathrm{K}} \cdot C^{\mathrm{K}}) \to^{\mathcal{O}(1)}_{SpKAM} (k, \epsilon, C^{\mathrm{K}})$ *in space* $\mathcal{O}(\lvert C^{\mathrm{K}} \rvert)$;
- *otherwise if $C \to_{\mathcal{M}} D$, then* $(\theta, \epsilon, (\theta, \epsilon) \cdot (\mathtt{transaux}, \epsilon) \cdot k^{\mathrm{K}} \cdot C^{\mathrm{K}}) \to^{\mathcal{O}(1)}_{SpKAM} (\theta, \epsilon, (\theta, \epsilon) \cdot (\mathtt{transaux}, \epsilon) \cdot k^{\mathrm{K}} \cdot D^{\mathrm{K}})$ *in space* $\mathcal{O}(\lvert C^{\mathrm{K}} \rvert)$.

Proof. The first part of the proof is common to both points.
Let us define $\mathtt{tx} := \mathtt{transaux}$ and $t := \lambda i'.\lambda n'.\lambda w'_l.\lambda a'.\lambda w'_r.\lambda s'.\mathtt{lookup}\, K i' n'$

Term	Env	Stack	
$\theta := \lambda x.\lambda y.y(xxy)$	ϵ	$(\theta,\epsilon)\cdot(\mathtt{tx},\epsilon)\cdot k^{\mathsf{K}}\cdot C^{\mathsf{K}}$	$\to^2$
$y(xxy)$	$[x\leftarrow(\theta,\epsilon)]\cdot[y\leftarrow(\mathtt{tx},\epsilon)]$	$k^{\mathsf{K}}\cdot C^{\mathsf{K}}$	
		$\overbrace{(xxy,[y\leftarrow(\mathtt{tx},\epsilon)]\cdot[x\leftarrow(\theta,\epsilon)])}^{\mathsf{fix}^{\mathsf{K}}}\cdot k^{\mathsf{K}}\cdot C^{\mathsf{K}}$	
y	$[y\leftarrow(\mathtt{tx},\epsilon)]$	$\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot C^{\mathsf{K}}$	$=$
$\mathtt{tx}$	ϵ	$\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot C^{\mathsf{K}}$	$\to^2$
$\lambda x.\lambda k.\lambda C'.C't$	ϵ		
	$\overbrace{[k'\leftarrow k^{\mathsf{K}}]\cdot[x\leftarrow\mathsf{fix}^{\mathsf{K}}]}^{E}$		
$\lambda C'.C't$	$[C'\leftarrow C^{\mathsf{K}}]\cdot E$	C^{K}	
$C't$	$[C'\leftarrow C^{\mathsf{K}}]$	ϵ	
C'	$[C'\leftarrow C^{\mathsf{K}}]$	(t,E)	
$\lambda x.xfcm\lceil a_j\rceil d\lceil s_g\rceil$	$[f\leftarrow(\bar{i},\epsilon)],[c\leftarrow n^{\mathsf{K}}],[m\leftarrow s^{\mathsf{K}}],[d\leftarrow r^{\mathsf{K}}]$	(t,E)	$\to^7$
		$\overbrace{(\bar{i},\epsilon)}^{\bar{i}^{\mathsf{K}}}\cdot n^{\mathsf{K}}\cdot s^{\mathsf{K}}\cdot\overbrace{(\lceil a_j\rceil,\epsilon)}^{\lceil a\rceil^{\mathsf{K}}}\cdot r^{\mathsf{K}}\cdot\overbrace{(\lceil s_g\rceil,\epsilon)}^{\lceil s\rceil^{\mathsf{K}}}$	
x	$[x\leftarrow(t,E)]$		
$\lambda i'.\lambda n'.\lambda w_l'.\lambda a'.\lambda w_r'.\lambda s'.\mathtt{lookup}\,Ki'n'$	E	$\bar{i}^{\mathsf{K}}\cdot n^{\mathsf{K}}\cdot s^{\mathsf{K}}\cdot\lceil a\rceil^{\mathsf{K}}\cdot r^{\mathsf{K}}\cdot\lceil s\rceil^{\mathsf{K}}$	$\to^6$
	$\overbrace{[i'\leftarrow\bar{i}^{\mathsf{K}}]\cdot[n'\leftarrow n^{\mathsf{K}}]\cdot[w_l\leftarrow s^{\mathsf{K}}]\cdot[a'\leftarrow\lceil a\rceil^{\mathsf{K}}]\cdot[w_r\leftarrow r^{\mathsf{K}}]\cdot[s'\leftarrow\lceil q\rceil^{\mathsf{K}}]\cdot E}^{E'}$		
$\mathtt{lookup}\,Ki'n'$	ϵ	$(K,E')\cdot\bar{i}^{\mathsf{K}}\cdot n^{\mathsf{K}}$	$\to^3$
$\mathtt{lookup}$	ϵ	$(\lceil a_i\rceil,\epsilon)$	$\to^{\lvert S\rvert}$
$K := \lambda b'.b'A_0A_1A_{\mathsf{L}}A_{\mathsf{R}}a's'xk'i'n'w_l'w_r'$	E'		
$b'A_0A_1A_{\mathsf{L}}A_{\mathsf{R}}a's'xk'i'n'w_l'w_r'$	$[b'\leftarrow(\lceil a_i\rceil,\epsilon)],E'$	ϵ	$\to^{12}$
b'	$[b'\leftarrow(\lceil a_i\rceil,\epsilon)]$	$(A_b,\epsilon)_{b\in\mathbb{B}_{\mathsf{I}}}\cdot\lceil a\rceil^{\mathsf{K}}\cdot\lceil s\rceil^{\mathsf{K}}\cdot\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot n^{\mathsf{K}}\cdot s^{\mathsf{K}}\cdot r^{\mathsf{K}}$	
$\lceil a_i\rceil := \lambda x_0.\lambda x_1.\lambda x_{\mathsf{L}}.\lambda x_{\mathsf{R}}.x_i$	ϵ	$(A_b,\epsilon)_{b\in\mathbb{B}_{\mathsf{I}}}\cdot\lceil a\rceil^{\mathsf{K}}\cdot\lceil s\rceil^{\mathsf{K}}\cdot\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot n^{\mathsf{K}}\cdot s^{\mathsf{K}}\cdot r^{\mathsf{K}}$	$\to^5$
$A_i := \lambda a'.a'B_{i,0}B_{i,1}B_{i,\square}$	ϵ	$\lceil a\rceil^{\mathsf{K}}\cdot\lceil s\rceil^{\mathsf{K}}\cdot\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot n^{\mathsf{K}}\cdot s^{\mathsf{K}}\cdot r^{\mathsf{K}}$	
$a'B_{i,0}B_{i,1}B_{i,\square}$	$[a'\leftarrow\lceil a\rceil^{\mathsf{K}}]$	$\lceil s\rceil^{\mathsf{K}}\cdot\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot n^{\mathsf{K}}\cdot s^{\mathsf{K}}\cdot r^{\mathsf{K}}$	$\to^3$
a'	$[a'\leftarrow\lceil a\rceil^{\mathsf{K}}]$	$(B_{i,b},\epsilon)_{b\in\mathbb{B}_{\mathsf{W}}}\cdot\lceil s\rceil^{\mathsf{K}}\cdot\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot n^{\mathsf{K}}\cdot s^{\mathsf{K}}\cdot r^{\mathsf{K}}$	
$\lceil a_j\rceil := \lambda x_0.\lambda x_1.\lambda x_{\square}.x_j$	ϵ	$(B_{i,b},\epsilon)_{b\in\mathbb{B}_{\mathsf{W}}}\cdot\lceil s\rceil^{\mathsf{K}}\cdot\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot n^{\mathsf{K}}\cdot s^{\mathsf{K}}\cdot r^{\mathsf{K}}$	$\to^4$
$B_{i,j} := \lambda s'.s'C_{i,j,s_1}\dots C_{i,j,s_{\lvert S\rvert}}$	ϵ	$\lceil s\rceil^{\mathsf{K}}\cdot\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot n^{\mathsf{K}}\cdot s^{\mathsf{K}}\cdot r^{\mathsf{K}}$	
$s'C_{i,j,s_1}\dots C_{i,j,s_{\lvert S\rvert}}$	$[s'\leftarrow\lceil s\rceil^{\mathsf{K}}]$	$\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot n^{\mathsf{K}}\cdot s^{\mathsf{K}}\cdot r^{\mathsf{K}}$	$\to^{\lvert S\rvert}$
s'	$[s'\leftarrow\lceil s\rceil^{\mathsf{K}}]$	$(C_{i,j,s_g},\epsilon)_{1\le g\le\lvert\Sigma\rvert}\cdot\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot n^{\mathsf{K}}\cdot s^{\mathsf{K}}\cdot r^{\mathsf{K}}$	
$\lceil s_g\rceil := \lambda x_1\dots\lambda x_{\lvert S\rvert}.x_g$	ϵ	$(C_{i,j,s_g},\epsilon)_{1\le g\le\lvert\Sigma\rvert}\cdot\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot n^{\mathsf{K}}\cdot s^{\mathsf{K}}\cdot r^{\mathsf{K}}$	$\to^{1+\lvert S\rvert}$
C_{i,j,s_g}	ϵ	$\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot n^{\mathsf{K}}\cdot s^{\mathsf{K}}\cdot r^{\mathsf{K}}$	

Cases of the transition to apply:

- *No transition*, that is, C is a final configuration, which happens when $s_g \in S_{fin}$.
  We have $C_{i,j,s_g} := \lambda x.\lambda k'.\lambda i'.\lambda n'.\lambda w_l'.\lambda w_r'.k'\langle i', n' \mid w_l', \lceil a_j\rceil, w_r' \mid \lceil s_g\rceil\rangle$

Term	Env	Stack
C_{i,j,s_g}	ϵ	$\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot n^{\mathsf{K}}\cdot s^{\mathsf{K}}\cdot r^{\mathsf{K}}$
	$\overbrace{\qquad\qquad}^{E_2}$	
$k'\langle i', n' \mid w_l', \lceil a_j\rceil, w_r' \mid \lceil s_g\rceil\rangle$	$[k'\leftarrow k^{\mathsf{K}}]\cdot\overbrace{[w_l'\leftarrow s^{\mathsf{K}}]\cdot[w_r'\leftarrow r^{\mathsf{K}}]\cdot[i'\leftarrow\bar{i}^{\mathsf{K}}]\cdot[n'\leftarrow n^{\mathsf{K}}]}^{E_2}$	ϵ
k'	$[k'\leftarrow k^{\mathsf{K}} =: (k,E)]$	$(\langle i', n' \mid w_l', \lceil a_j\rceil, w_r' \mid \lceil s_g\rceil\rangle, E_2)$
k	E	$(\langle i', n' \mid w_l', \lceil a_j\rceil, w_r' \mid \lceil s_g\rceil\rangle, E_2) =: C^{\mathsf{K}}$

- *The heads do not move*, that is, $\delta(a_i, a_j, s_g) = (0 \mid a_h, \downarrow \mid s_l)$.

Term	Env	Stack	
$C_{i,j,s_g} := \lambda x.\lambda k'.\lambda i'.\lambda n'.\lambda w_l'.\lambda w_r'.Sn'$	ϵ	$\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot n^{\mathsf{K}}\cdot s^{\mathsf{K}}\cdot r^{\mathsf{K}}$	$\to^6$
Sn'	$[x\leftarrow\mathsf{fix}^{\mathsf{K}}]\cdot[k'\leftarrow k^{\mathsf{K}}]\cdot[w_l'\leftarrow s^{\mathsf{K}}]\cdot[w_r'\leftarrow r^{\mathsf{K}}]\cdot[i'\leftarrow\bar{i}^{\mathsf{K}}]\cdot[n'\leftarrow n^{\mathsf{K}}]$	ϵ	$\to$
$S := \lambda n''.xk'\langle i', n'' \mid w_l', \lceil a_h\rceil, w_r' \mid \lceil s_l\rceil\rangle$	$[x\leftarrow\mathsf{fix}^{\mathsf{K}}]\cdot[k'\leftarrow k^{\mathsf{K}}]\cdot[w_l'\leftarrow s^{\mathsf{K}}]\cdot[w_r'\leftarrow r^{\mathsf{K}}]\cdot[i'\leftarrow\bar{i}^{\mathsf{K}}]$	n^{K}	
	$\overbrace{\qquad\qquad}^{E_2}$		
$xk'\langle i', n'' \mid w_l', \lceil a_h\rceil, w_r' \mid \lceil s_l\rceil\rangle$	$[x\leftarrow\mathsf{fix}^{\mathsf{K}}]\cdot[k'\leftarrow k^{\mathsf{K}}]\cdot\overbrace{[w_l'\leftarrow s^{\mathsf{K}}]\cdot[w_r'\leftarrow r^{\mathsf{K}}]\cdot[i'\leftarrow\bar{i}^{\mathsf{K}}]\cdot[n''\leftarrow n^{\mathsf{K}}]}^{E_2}$		
		$\overbrace{\qquad\qquad}^{D^{\mathsf{K}}}$	
x	$[x\leftarrow\mathsf{fix}^{\mathsf{K}}]$	$k^{\mathsf{K}}\cdot\overbrace{(\langle i', n'' \mid w_l', \lceil a_h\rceil, w_r' \mid \lceil s_l\rceil\rangle, E_2)}^{D^{\mathsf{K}}}$	
xxy	$[y\leftarrow(\mathtt{tx},\epsilon)]\cdot[x\leftarrow(\theta,\epsilon)]$	$k^{\mathsf{K}}\cdot D^{\mathsf{K}}$	$\to^2$
x	$[x\leftarrow(\theta,\epsilon)]$	$(\theta,\epsilon)\cdot(\mathtt{tx},\epsilon)\cdot k^{\mathsf{K}}\cdot D^{\mathsf{K}}$	
θ	ϵ	$(\theta,\epsilon)\cdot(\mathtt{tx},\epsilon)\cdot k^{\mathsf{K}}\cdot D^{\mathsf{K}}$	

- *The heads move right*, that is, $\delta(a_i, a_j, s_g) = (1 \mid a_h, \to \mid s_l)$.

Term	Env	Stack	
$C_{i,j,s_g} := \lambda x.\lambda k'.\lambda i'.\lambda n'.\lambda w_l'.\lambda w_r'.\mathtt{succ}Rn'$	ϵ	$\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot \bar{i}^{\mathsf{K}}\cdot n^{\mathsf{K}}\cdot s^{\mathsf{K}}\cdot r^{\mathsf{K}}$	$\to^6$
$\mathtt{succ}Rn'$	$\overbrace{[x\leftarrow\mathsf{fix}^{\mathsf{K}}]\cdot[k'\leftarrow k^{\mathsf{K}}]\cdot[w_l'\leftarrow s^{\mathsf{K}}]\cdot[w_r'\leftarrow r^{\mathsf{K}}]\cdot[i'\leftarrow \bar{i}^{\mathsf{K}}]}^{E_2}\cdot[n'\leftarrow n^{\mathsf{K}}]$	ϵ	$\to^2$
$\mathtt{succ}$	ϵ	$(R, E_2)\cdot n^{\mathsf{K}}$	
$R := \lambda n''.w_r' R_0^{s_l,a_h} R_1^{s_l,a_h} R_\square^{s_l,a_h} R_\varepsilon^{s_l,a_h} xk'i'n''w_l'$	E_2	$m^{\mathsf{K}} := (n+1)^{\mathsf{K}}$	
$w_r' R_0^{s_l,a_h} R_1^{s_l,a_h} R_\square^{s_l,a_h} R_\varepsilon^{s_l,a_h} xk'i'n''w_l'$	$[x\leftarrow\mathsf{fix}^{\mathsf{K}}]\cdot[k'\leftarrow k^{\mathsf{K}}]\cdot[w_l'\leftarrow s^{\mathsf{K}}]\cdot[w_r'\leftarrow r^{\mathsf{K}}]\cdot[i'\leftarrow\bar{i}^{\mathsf{K}}]\cdot[n''\leftarrow m^{\mathsf{K}}]$	ϵ	
w_r'	$[w_r'\leftarrow r^{\mathsf{K}}]$	$(R_x^{s_l,a_h},\epsilon)_{x\in\{0,1,\square,\varepsilon\}}\cdot\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot m^{\mathsf{K}}\cdot s^{\mathsf{K}}$	

Two cases.

– $r = \varepsilon$. Define $t := (\lambda d.\lambda w_l'.xk'\langle i', n' \mid w_l', \lceil\square\rceil, d \mid \lceil s_l\rceil\rangle)\bar{\varepsilon}$

Term	Env	Stack	
w_r'	$[w_r'\leftarrow r^{\mathsf{K}}]$	$(R_x^{s_l,a_h},\epsilon)_{x\in\{0,1,\square,\varepsilon\}}\cdot\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot m^{\mathsf{K}}\cdot s^{\mathsf{K}}$	$\to^5$
$R_\varepsilon^{s_l,a_h} := \lambda x.\lambda k'.\lambda i'.\lambda n'.\mathtt{append}^{a_h}t$	ϵ	$\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot m^{\mathsf{K}}\cdot s^{\mathsf{K}}$	$\to^4$
$\mathtt{append}^{a_h}t$	$\overbrace{[x\leftarrow\mathsf{fix}^{\mathsf{K}}]\cdot[k'\leftarrow k^{\mathsf{K}}]\cdot[i'\leftarrow\bar{i}^{\mathsf{K}}]}^{E_2}\cdot[n'\leftarrow m^{\mathsf{K}}]$	s^{K}	
$\mathtt{append}^{a_h}$	ϵ	$(t, E_2)\cdot s^{\mathsf{K}}$	
$t := (\lambda d.\lambda w_l'.xk'\langle i', n' \mid w_l', \lceil\square\rceil, d \mid \lceil s_l\rceil\rangle)\bar{\varepsilon}$	E_2	$s_h^{\mathsf{K}} := (a_h\cdot s)^{\mathsf{K}}$	
$\lambda d.\lambda w_l'.xk'\langle i', n' \mid w_l', \lceil\square\rceil, d \mid \lceil s_l\rceil\rangle$	E_2	$(\bar{\varepsilon},\epsilon)\cdot s_h^{\mathsf{K}}$	$\to^2$
$xk'\langle i', n' \mid w_l', \lceil\square\rceil, d \mid \lceil s_l\rceil\rangle$	$[x\leftarrow\mathsf{fix}^{\mathsf{K}}]\cdot[k'\leftarrow k^{\mathsf{K}}]\cdot\overbrace{[i'\leftarrow\bar{i}^{\mathsf{K}}]\cdot[n'\leftarrow m^{\mathsf{K}}]\cdot[d\leftarrow(\bar{\varepsilon},\epsilon)]\cdot[w_l'\leftarrow s_h^{\mathsf{K}}]}^{E_3}$	ϵ	$\to^2$
x	$[x\leftarrow\mathsf{fix}^{\mathsf{K}}]$	$k^{\mathsf{K}}\cdot\overbrace{(\langle i', n' \mid w_l', \lceil\square\rceil, d \mid \lceil s_l\rceil\rangle, E_3)}^{D^{\mathsf{K}}}$	
xxy	$[y\leftarrow(\mathtt{tx},\epsilon)]\cdot[x\leftarrow(\theta,\epsilon)]$	$k^{\mathsf{K}}\cdot D^{\mathsf{K}}$	$\to^2$
x	$[x\leftarrow(\theta,\epsilon)]$	$(\theta,\epsilon)\cdot(\mathtt{tx},\epsilon)\cdot k^{\mathsf{K}}\cdot D^{\mathsf{K}}$	
θ	ϵ	$(\theta,\epsilon)\cdot(\mathtt{tx},\epsilon)\cdot k^{\mathsf{K}}\cdot D^{\mathsf{K}}$	

– $r = a''\cdot r'$. Define $t := \lambda w_l'.xk'\langle i', n' \mid w_l', \lceil a''\rceil, w_r' \mid \lceil s_l\rceil\rangle$

Term	Env	Stack	
w_r'	$[w_r'\leftarrow r^{\mathsf{K}}]$	$(R_x^{s_l,a_h},\epsilon)_{x\in\{0,1,\square,\varepsilon\}}\cdot\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot m^{\mathsf{K}}\cdot s^{\mathsf{K}}$	
$\lambda x_0.\lambda x_1.\lambda x_\square.\lambda y.x_{i_{a''}}z$	$[z\leftarrow r'^{\mathsf{K}}]$	$(R_x^{s_l,a_h},\epsilon)_{x\in\{0,1,\square,\varepsilon\}}\cdot\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot m^{\mathsf{K}}\cdot s^{\mathsf{K}}$	$\to^6$
$R_{a''}^{s_l,a_h} := \lambda w_r'.\lambda x.\lambda k'.\lambda i'.\lambda n'.\mathtt{append}^{a_h}t$	ϵ	$r'^{\mathsf{K}}\cdot\mathsf{fix}^{\mathsf{K}}\cdot k^{\mathsf{K}}\cdot\bar{i}^{\mathsf{K}}\cdot m^{\mathsf{K}}\cdot s^{\mathsf{K}}$	$\to^5$
$\mathtt{append}^{a_h}t$	$\overbrace{[w_r'\leftarrow r'^{\mathsf{K}}]\cdot[x\leftarrow\mathsf{fix}^{\mathsf{K}}]\cdot[k'\leftarrow k^{\mathsf{K}}]\cdot[i'\leftarrow\bar{i}^{\mathsf{K}}]}^{E_2}\cdot[n'\leftarrow m^{\mathsf{K}}]$	s^{K}	
$\mathtt{append}^{a_h}$	ϵ	$(t, E_2)\cdot s^{\mathsf{K}}$	
$t := \lambda w_l'.xk'\langle i', n' \mid w_l', \lceil a''\rceil, w_r' \mid \lceil s_l\rceil\rangle$	E_2	$s_h^{\mathsf{K}} := (a_h\cdot s)^{\mathsf{K}}$	
$xk'\langle i', n' \mid w_l', \lceil a''\rceil, w_r' \mid \lceil s_l\rceil\rangle$	$[x\leftarrow\mathsf{fix}^{\mathsf{K}}]\cdot[k'\leftarrow k^{\mathsf{K}}]\cdot\overbrace{[i'\leftarrow\bar{i}^{\mathsf{K}}]\cdot[n'\leftarrow m^{\mathsf{K}}]\cdot[w_r'\leftarrow r'^{\mathsf{K}}]\cdot[w_l'\leftarrow s_h^{\mathsf{K}}]}^{E_3}$	ϵ	$\to^2$
x	$[x\leftarrow\mathsf{fix}^{\mathsf{K}}]$	$k^{\mathsf{K}}\cdot\overbrace{(\langle i', n' \mid w_l', \lceil a''\rceil, w_r' \mid \lceil s_l\rceil\rangle, E_3}^{D^{\mathsf{K}}}$	
xxy	$[y\leftarrow(\mathtt{tx},\epsilon)]\cdot[x\leftarrow(\theta,\epsilon)]$	$k^{\mathsf{K}}\cdot D^{\mathsf{K}}$	$\to^2$
x	$[x\leftarrow(\theta,\epsilon)]$	$(\theta,\epsilon)\cdot(\mathtt{tx},\epsilon)\cdot k^{\mathsf{K}}\cdot D^{\mathsf{K}}$	
θ	ϵ	$(\theta,\epsilon)\cdot(\mathtt{tx},\epsilon)\cdot k^{\mathsf{K}}\cdot D^{\mathsf{K}}$	

- All the other cases are almost identical *mutatis mutandis*.

About the space bound we observe that in the simulations *all* the pointers except for those related to the input part of the state are pointers to the machine, and not to the input. Moreover, the space overhead of the simulation of one step of the TM is constant, *i.e.* non input dependent. □

Lemma E.0.9. *If $\rho : C \to^n D$ and D is final, then $(\mathtt{trans}\,k, \epsilon, C_{\mathtt{in}}(i)^{\mathsf{K}}) \to_{SpKAM} (k, \epsilon, C^{\mathsf{K}})$ in space $\mathcal{O}(S_{TM}(\rho) + \log(|i|))$.*

Proof. By a simple induction on n, using the two lemmata above, and knowing that $S_{\mathrm{TM}}(\rho) = \max_{C\in\rho}|C|$ (we have also to consider that $|C| = |C^{\mathsf{K}}|$, by Lemma E.0.2). □

Bibliography

Abramsky, Samson, Esfandiar Haghverdi, and Philip Scott (2002). "Geometry of Interaction and linear combinatory algebras". en. In: *Mathematical Structures in Computer Science* 12.5, pp. 625–665. ISSN: 1469-8072, 0960-1295. (Visited on 04/30/2019).

Abramsky, Samson, Radha Jagadeesan, and Pasquale Malacaria (2000). "Full Abstraction for PCF". In: *Inf. Comput.* 163.2, pp. 409–470. DOI: 10.1006/inco.2000.2930.

Accattoli, Beniamino (2012). "An Abstract Factorization Theorem for Explicit Substitutions". In: *Proceedings of RTA'12*. Vol. 15. LIPIcs, pp. 6–21.

— (2016). "The Complexity of Abstract Machines". In: *Proceedings Third International Workshop on Rewriting Techniques for Program Transformations and Evaluation, WPTE@FSCD 2016, Porto, Portugal, 23rd June 2016*. Ed. by Horatiu Cirstea and Santiago Escobar. Vol. 235. EPTCS, pp. 1–15. DOI: 10.4204/EPTCS.235.1.

— (2017). "(In)Efficiency and Reasonable Cost Models". In: *12th Workshop on Logical and Semantic Frameworks, with Applications, LSFA 2017, Brasília, Brazil, September 23-24, 2017*. Vol. 338. Electronic Notes in Theoretical Computer Science. Elsevier, pp. 23–43. DOI: 10.1016/j.entcs.2018.10.003.

— (2018). "Proof Nets and the Linear Substitution Calculus". In: *Proceedings of the 15th ICTAC*, pp. 37–61.

Accattoli, Beniamino, Pablo Barenbaum, and Damiano Mazza (2014a). "Distilling abstract machines". In: *Proceedings of the 19th ACM SIGPLAN international conference on Functional programitarmming, Gothenburg, Sweden, September 1-3, 2014*. Ed. by Johan Jeuring and Manuel M. T. Chakravarty. ACM, pp. 363–376. DOI: 10.1145/2628136.2628154.

Accattoli, Beniamino and Bruno Barras (2017). "Environments and the complexity of abstract machines". In: *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming, Namur, Belgium, October 09 - 11, 2017*. Ed. by Wim Vanhoof and Brigitte Pientka. ACM, pp. 4–16. DOI: 10.1145/3131851.3131855.

Accattoli, Beniamino and Claudio Sacerdoti Coen (2017). "On the value of variables". In: *Inf. Comput.* 255, pp. 224–242.

Accattoli, Beniamino, Andrea Condoluci, and Claudio Sacerdoti Coen (2021a). "Strong Call-by-Value is Reasonable, Implosively". In: *LICS*. IEEE, pp. 1–14.

Accattoli, Beniamino and Ugo Dal Lago (2012). "On the Invariance of the Unitary Cost Model for Head Reduction". In: *23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012, May 28 - June 2, 2012, Nagoya, Japan*. Ed. by Ashish Tiwari. Vol. 15. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 22–37. ISBN: 978-3-939897-38-5. DOI: 10.4230/LIPIcs.RTA.2012.22.

— (2016). "(Leftmost-Outermost) Beta Reduction is Invariant, Indeed". en. In: *Logical Methods in Computer Science* 12.1. DOI: 10.2168/LMCS-12(1:4)2016.

Accattoli, Beniamino, Ugo Dal Lago, and Gabriele Vanoni (2020a). "The Machinery of Interaction". In: *Proceedings of the 22nd International Symposium on Principles and Practice of Declarative Programming*. PPDP '20. Bologna, Italy: Association for Computing Machinery. ISBN: 9781450388214. DOI: 10.1145/3414080.3414108.

— (2021b). "Interacting Seems Unreasonable, in Time and Space". In: *Tenth Workshop on Intersection Types and Related Systems - ITRS 2021*. http://www.di.unito.it/~deligu/ITRS2021/ITRS_2021_paper_3.pdf.

— (Jan. 2021c). "The (In)Efficiency of Interaction". In: *Proc. ACM Program. Lang.* 5.POPL. DOI: 10.1145/3434332. URL: https://doi.org/10.1145/3434332.

— (2021d). "The Space of Interaction". In: *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pp. 1–13. DOI: 10.1109/LICS52264.2021.9470726.

Accattoli, Beniamino, Ugo Dal Lago, and Gabriele Vanoni (2022a). "Multi Types and Reasonable Space". Accepted at ICFP 2022.

— (2022b). "Reasonable Space for the λ-calculus, Logarithmically". Accepted at LICS 2022.

Accattoli, Beniamino, Stéphane Graham-Lengrand, and Delia Kesner (2020b). "Tight typings and split bounds, fully developed". In: *J. Funct. Program.* 30, e14. DOI: 10.1017/S095679682000012X.

Accattoli, Beniamino and Giulio Guerrieri (2018). "Types of Fireballs". In: *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings*. Ed. by Sukyoung Ryu. Vol. 11275. Lecture Notes in Computer Science. Springer, pp. 45–66. DOI: 10.1007/978-3-030-02768-1_3.

— (2019). "Abstract machines for Open Call-by-Value". In: *Sci. Comput. Program.* 184. DOI: 10.1016/j.scico.2019.03.002. URL: https://doi.org/10.1016/j.scico.2019.03.002.

Accattoli, Beniamino, Giulio Guerrieri, and Maico Leberle (2019a). "Types by Need". In: *28th European Symposium on Programming, ESOP 2019,Prague, Czech Republic, April 6-11, 2019, Proceedings*. Vol. 11423. Lecture Notes in Computer Science. Springer, pp. 410–439. DOI: 10.1007/978-3-030-17184-1_15.

Accattoli, Beniamino and Delia Kesner (2010). "The Structural λ-Calculus". In: *Proceedings of CSL'10*, pp. 381–395.

Accattoli, Beniamino et al. (2014b). "A nonstandard standardization theorem". In: *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*. Ed. by Suresh Jagannathan and Peter Sewell. ACM, pp. 659–670. DOI: 10.1145/2535838.2535886.

Accattoli, Beniamino et al. (2019b). "Crumbling Abstract Machines". In: *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*. Ed. by Ekaterina Komendantskaya. ACM, 4:1–4:15. DOI: 10.1145/3354166.3354169.

Alves, Sandra, Delia Kesner, and Daniel Ventura (2019). "A Quantitative Understanding of Pattern Matching". In: *25th International Conference on Types for Proofs and Programs, TYPES 2019, June 11-14, 2019, Oslo, Norway*, 3:1–3:36. DOI: 10.4230/LIPIcs.TYPES.2019.3.

Asperti, Andrea et al. (1994). "Paths in the lambda-calculus". In: *Proceedings of the Ninth Annual Symposium on Logic in Computer Science (LICS '94), Paris, France, July 4-7, 1994*. IEEE Computer Society, pp. 426–436. DOI: 10.1109/LICS.1994.316048.

Aubert, Clément et al. (2014). "Logic Programming and Logarithmic Space". In: *Programming Languages and Systems - 12th Asian Symposium, APLAS 2014, Singapore, November 17-19, 2014, Proceedings*. Ed. by Jacques Garrigue. Vol. 8858. Lecture Notes in Computer Science. Springer, pp. 39–57. DOI: 10.1007/978-3-319-12736-1_3.

Baillot, Patrick (1999). "Approches dynamiques en sémantique de la logique lineaire: jeux et géométrie de l'interaction". PhD Thesis. Universite Aix-Marseille 2.

Baillot, Patrick, Paolo Coppola, and Ugo Dal Lago (2011). "Light logics and optimal reduction: Completeness and complexity". In: *Inf. Comput.* 209.2, pp. 118–142. DOI: 10.1016/j.ic.2010.10.002.

Barendregt, Hendrik Pieter (1984). *The lambda calculus: its syntax and semantics*. en. North-Holland. ISBN: 978-0-444-86748-3.

Biernacka, Malgorzata, Witold Charatonik, and Tomasz Drab (2021). "A Derived Reasonable Abstract Machine for Strong Call by Value". In: *PPDP 2021: 23rd International Symposium on Principles and Practice of Declarative Programming, Tallinn, Estonia, September 6-8, 2021*. Ed. by Niccolò Veltri, Nick Benton, and Silvia Ghilezan. ACM, 6:1–6:14. DOI: 10.1145/3479394.3479401. URL: https://doi.org/10.1145/3479394.3479401.

Blelloch, Guy E. and John Greiner (1995). "Parallelism in Sequential Functional Languages". In: *Proceedings of the seventh international conference on Functional programming languages and computer architecture, FPCA 1995, La Jolla, California, USA, June 25-28, 1995*. ACM, pp. 226–237. DOI: 10.1145/224164.224210.

— (1996). "A Provable Time and Space Efficient Implementation of NESL". In: *Proceedings of the 1996 ACM SIGPLAN International Conference on Functional Programming, ICFP 1996, Philadelphia, Pennsylvania, USA, May 24-26, 1996*. Ed. by Robert Harper and Richard L. Wexelblat. ACM, pp. 213–225. DOI: 10.1145/232627.232650.

Bono, Viviana and Mariangiola Dezani-Ciancaglini (2020). "A tale of intersection types". In: *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*. Ed. by Holger Hermanns et al. ACM, pp. 7–20. DOI: 10.1145/3373718.3394733. URL: https://doi.org/10.1145/3373718.3394733.

Bucciarelli, Antonio and Thomas Ehrhard (2001). "On phase semantics and denotational semantics: the exponentials". In: *Ann. Pure Appl. Log.* 109.3, pp. 205–241. DOI: 10.1016/S0168-0072(00)00056-7.

Bucciarelli, Antonio, Delia Kesner, and Simona Ronchi Della Rocca (2018). "Inhabitation for Non-idempotent Intersection Types". In: *Log. Methods Comput. Sci.* 14.3. DOI: 10.23638/LMCS-14(3:7)2018. URL: https://doi.org/10.23638/LMCS-14(3:7)2018.

— (2021). "Solvability = Typability + Inhabitation". In: *Log. Methods Comput. Sci.* 17.1. URL: https://lmcs.episciences.org/7141.

Bucciarelli, Antonio, Delia Kesner, and Daniel Ventura (2017). "Non-idempotent intersection types for the Lambda-Calculus". In: *Logic Journal of the IGPL* 25.4, pp. 431–464. DOI: 10.1093/jigpal/jzx018.

Bucciarelli, Antonio et al. (2020). "The Bang Calculus Revisited". In: *Functional and Logic Programming - 15th International Symposium, FLOPS 2020, Akita, Japan, September 14-16, 2020, Proceedings*. Ed. by Keisuke Nakano and Konstantinos Sagonas. Vol. 12073. Lecture Notes in Computer Science. Springer, pp. 13–32. DOI: 10.1007/978-3-030-59025-3_2.

Cardone, Felice and J. Roger Hindley (2009). "Lambda-Calculus and Combinators in the 20th Century". In: *Handbook of the History of Logic*. Ed. by Dov M. Gabbay and John Woods. Vol. 5. North-Holland, pp. 723–817.

Carraro, Alberto and Giulio Guerrieri (2014). "A Semantical and Operational Account of Call-by-Value Solvability". In: *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*. Ed. by Anca Muscholl. Vol. 8412. Lecture Notes in Computer Science. Springer, pp. 103–118. DOI: 10.1007/978-3-642-54830-7_7. URL: https://doi.org/10.1007/978-3-642-54830-7_7.

Church, Alonzo (1932). "A Set of Postulates for the Foundation of Logic". In: *Annals of Mathematics* 33.2, pp. 346–366.

— (1936a). "A Note on the Entscheidungsproblem". In: *Journal of Symbolic Logic* 1.1, pp. 40–41.

— (1936b). "An Unsolvable Problem of Elementary Number Theory". In: *American Journal of Mathematics* 58.2, pp. 345–363.

Condoluci, Andrea, Beniamino Accattoli, and Claudio Sacerdoti Coen (2019). "Sharing Equality is Linear". In: *PPDP*. ACM, 9:1–9:14.

Coppo, Mario and Mariangiola Dezani-Ciancaglini (1978). "A new type assignment for λ-terms". In: *Arch. Math. Log.* 19.1, pp. 139–156. DOI: 10.1007/BF02011875.

Curien, Pierre-Louis and Hugo Herbelin (1998). "Computing with Abstract Böhm Trees". In: *Third Fuji International Symposium on Functional and Logic Programming, FLOPS 1998, Kyoto, Japan, Apil 2-4, 1998*. Ed. by Masahiko Sato and Yoshihito Toyama. World Scientific, Singapore, pp. 20–39.

— (2007). "Abstract machines for dialogue games". In: URL: http://arxiv.org/abs/0706.2544.

Curry, Haskell Brooks and Robert Feys (1958). *Combinatory Logic*. North-Holland.

Dal Lago, Ugo and Beniamino Accattoli (2017). "Encoding Turing Machines into the Deterministic Lambda-Calculus". In: *CoRR* abs/1711.10078.

Dal Lago, Ugo, Claudia Faggian, and Simona Ronchi Della Rocca (2021). "Intersection types and (positive) almost-sure termination". In: *Proc. ACM Program. Lang.* 5.POPL, pp. 1–32. DOI: 10.1145/3434313.

Dal Lago, Ugo and Marco Gaboardi (2011). "Linear Dependent Types and Relative Completeness". In: *Log. Methods Comput. Sci.* 8.4. DOI: 10.2168/LMCS-8(4:11)2012.

Dal Lago, Ugo and Ulrich Schöpp (2016). "Computation by interaction for space-bounded functional programming". In: *Information and Computation* 248, pp. 150–194. DOI: 10.1016/j.ic.2015.04.006.

Dal Lago, Ugo, Ryo Tanaka, and Akira Yoshimizu (2017a). "The geometry of concurrent interaction: Handling multiple ports by way of multiple tokens". In: *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, pp. 1–12. DOI: 10.1109/LICS.2017.8005112.

Dal Lago, Ugo et al. (2014). "The geometry of synchronization". In: *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*. Ed. by Thomas A. Henzinger and Dale Miller. ACM, 35:1–35:10. DOI: 10.1145/2603088.2603154.

Dal Lago, Ugo et al. (2015). "Parallelism and Synchronization in an Infinitary Context". In: *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*. IEEE Computer Society, pp. 559–572. DOI: 10.1109/LICS.2015.58.

— (2017b). "The geometry of parallelism: classical, probabilistic, and quantum effects". In: *Proceedings of the 44th POPL*, pp. 833–845.

Danos, Vincent, Hugo Herbelin, and Laurent Regnier (1996). "Game Semantics & Abstract Machines". In: *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*. IEEE Computer Society, pp. 394–405. DOI: 10.1109/LICS.1996.561456.

Danos, Vincent and Laurent Regnier (1993). "Local and asynchronous beta-reduction (an analysis of Girard's execution formula)". In: *Proceedings of the Eighth Annual Symposium on Logic in Computer Science (LICS '93), Montreal, Canada, June 19-23, 1993*. IEEE Computer Society, pp. 296–306. DOI: 10.1109/LICS.1993.287578.

— (1999). "Reversible, irreversible and optimal lambda-machines". In: *Theoretical Computer Science* 227.1, pp. 79–97. ISSN: 0304-3975. DOI: 10.1016/S0304-3975(99)00049-3. (Visited on 05/03/2018).

— (2004). *Head Linear Reduction*. Tech. rep.

de Carvalho, Daniel (2007). "Sémantiques de la logique linéaire et temps de calcul". Thèse de Doctorat. Université Aix-Marseille II.

— (2018). "Execution time of λ-terms via denotational semantics and intersection types". In: *Math. Str. in Comput. Sci.* 28.7, pp. 1169–1203. DOI: 10.1017/S0960129516000396.

Douence, Rémi and Pascal Fradet (2007). "The next 700 Krivine machines". In: *High. Order Symb. Comput.* 20.3, pp. 237–255.

Ehrhard, Thomas (2012). "Collapsing non-idempotent intersection types". In: *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*. Ed. by Patrick Cégielski and Arnaud Durand. Vol. 16. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 259–273. DOI: 10.4230/LIPIcs.CSL.2012.259. URL: https://doi.org/10.4230/LIPIcs.CSL.2012.259.

Emde Boas, Peter van (2012). "Turing Machines for Dummies - Why Representations Do Matter". In: *SOFSEM*. Vol. 7147. Lecture Notes in Computer Science. Springer, pp. 14–30.

Fernández, Maribel and Ian Mackie (2002). "Call-by-Value lambda-Graph Rewriting Without Rewriting". In: *Graph Transformation, First International Conference, ICGT 2002, Barcelona, Spain, October 7-12, 2002, Proceedings*. Ed. by Andrea Corradini et al. Vol. 2505. Lecture Notes in Computer Science. Springer, pp. 75–89. DOI: 10.1007/3-540-45832-8_8.

Forster, Yannick, Fabian Kunze, and Marc Roth (2020). "The weak call-by-value λ-calculus is reasonable for both time and space". In: *Proc. ACM Program. Lang.* 4.POPL, 27:1–27:23. DOI: 10.1145/3371095.

Friedman, Daniel P. et al. (2007). "Improving the lazy Krivine machine". In: *High. Order Symb. Comput.* 20.3, pp. 271–293.

Gaboardi, Marco, Jean-Yves Marion, and Simona Ronchi Della Rocca (2012). "An Implicit Characterization of PSPACE". In: *ACM Trans. Comput. Log.* 13.2, 18:1–18:36.

Gardner, Philippa (1994). "Discovering Needed Reductions Using Type Theory". In: *Theoretical Aspects of Computer Software (TACS '94)*. Vol. 789. Lecture Notes in Computer Science, pp. 555–574. DOI: 10.1007/3-540-57887-0_115.

Ghica, Dan R. (2007). "Geometry of Synthesis: A Structured Approach to VLSI Design". In: *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*. Ed. by Martin Hofmann and Matthias Felleisen. ACM, pp. 363–375. DOI: 10.1145/1190216.1190269.

Girard, Jean-Yves (1987). "Linear logic". In: *Theoretical Computer Science* 50.1, pp. 1–101. DOI: 10.1016/0304-3975(87)90045-4.

Girard, Jean-Yves (1988). "Geometry of interaction 2: deadlock-free algorithms". In: *COLOG-88, International Conference on Computer Logic, Tallinn, USSR, December 1988, Proceedings*. Ed. by Per Martin-Löf and Grigori Mints. Vol. 417. Lecture Notes in Computer Science. Springer, pp. 76–93. DOI: 10.1007/3-540-52335-9_49.

Girard, Jean-Yves (1989a). "Geometry of Interaction 1: Interpretation of System F". In: *Logic Colloquium '88*. Ed. by R. Ferro et al. Vol. 127. Studies in Logic and the Foundations of Mathematics. Elsevier, pp. 221 –260. DOI: https://doi.org/10.1016/S0049-237X(08)70271-4.

— (1989b). "Geometry of Interaction 1: Interpretation of System F". In: *Studies in Logic and the Foundations of Mathematics*. Ed. by R. Ferro et al. Vol. 127. Elsevier, pp. 221–260.

Gonthier, Georges, Martín Abadi, and Jean-Jacques Lévy (1992). "The Geometry of Optimal Lambda Reduction". In: *Proceedings of the 19th POPL*, pp. 15–26.

Hartmanis, J. and R. E. Stearns (1965). "On the Computational Complexity of Algorithms". In: *Transactions of the American Mathematical Society* 117, pp. 285–306.

Hartmanis, Juris and Janos Simon (1974). "On the Power of Multiplication in Random Access Machines". In: *15th Annual Symposium on Switching and Automata Theory, New Orleans, Louisiana, USA, October 14-16, 1974*. IEEE Computer Society, pp. 13–23. DOI: 10.1109/SWAT.1974.20.

Hoshino, Naohiko, Koko Muroya, and Ichiro Hasuo (2014). "Memoryful geometry of interaction: from coalgebraic components to algebraic effects". In: *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*. Ed. by Thomas A. Henzinger and Dale Miller. ACM, 52:1–52:10. DOI: 10.1145/2603088.2603124.

Howard, William Alvin (1980). "The Formulae-as-Types Notion of Construction". In: *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. Ed. by Haskell Curry et al. Academic Press.

Hyland, J. M. E. and C.-H. Luke Ong (2000). "On Full Abstraction for PCF: I, II, and III". In: *Inf. Comput.* 163.2, pp. 285–408. DOI: 10.1006/inco.2000.2917. URL: https://doi.org/10.1006/inco.2000.2917.

Jaber, Guilhem (2015). "Operational Nominal Game Semantics". In: *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*. Ed. by Andrew M. Pitts. Vol. 9034. Lecture Notes in Computer Science. Springer, pp. 264–278. DOI: 10.1007/978-3-662-46678-0_17.

Jones, Neil D. (1999). "LOGSPACE and PTIME Characterized by Programming Languages". In: *Theor. Comput. Sci.* 228.1-2, pp. 151–174.

Jones, Richard E., Antony L. Hosking, and J. Eliot B. Moss (2011). *The Garbage Collection Handbook: The art of automatic memory management*. Chapman and Hall / CRC Applied Algorithms and Data Structures Series. CRC Press. ISBN: 978-1-4200-8279-1. URL: http://gchandbook.org/.

Jones, Simon L. Peyton (1992). "Implementing Lazy Functional Languages on Stock Hardware: The Spineless Tagless G-Machine". In: *J. Funct. Program.* 2.2, pp. 127–202. DOI: 10.1017/S0956796800000319. URL: https://doi.org/10.1017/S0956796800000319.

Joyal, André, Ross Street, and Dominic Verity (1996). "Traced monoidal categories". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 119.3, 447–468.

Kesner, Delia and Shane Ó Conchúir (2008). *Milner's Lambda Calculus with Partial Substitutions*. Tech. rep. http://www.pps.univ-paris-diderot.fr/~kesner/papers/shortpartial.pdf. Paris 7 University.

Kesner, Delia and Pierre Vial (2017). "Types as Resources for Classical Natural Deduction". In: *2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, September 3-9, 2017, Oxford, UK*. Ed. by Dale Miller. Vol. 84. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 24:1–24:17. DOI: 10.4230/LIPIcs.FSCD.2017.24. URL: https://doi.org/10.4230/LIPIcs.FSCD.2017.24.

— (2020). "Consuming and Persistent Types for Classical Logic". In: *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*. Ed. by Holger Hermanns et al. ACM, pp. 619–632. DOI: 10.1145/3373718.3394774.

Kfoury, Assaf J. (2000). "A linearization of the Lambda-calculus and consequences". In: *Journal of Logic and Computation* 10.3, pp. 411–436. DOI: 10.1093/logcom/10.3.411.

Kleene, S. C. (1936). "λ-definability and recursiveness". In: *Duke Mathematical Journal* 2.2, pp. 340–353.

Kleene, S. C. and J. B. Rosser (1935). "The Inconsistency of Certain Formal Logics". In: *Annals of Mathematics* 36.3, pp. 630–636.

Krishnaswami, Neelakantan R., Nick Benton, and Jan Hoffmann (2012). "Higher-order functional reactive programming in bounded space". In: *Proc. of POPL 2012*. Ed. by John Field and Michael Hicks. ACM, pp. 45–58. ISBN: 978-1-4503-1083-3. DOI: 10.1145/2103656.2103665.

Krivine, Jean-Louis (2007). "A Call-by-name Lambda-calculus Machine". In: *Higher Order Symbol. Comput.* 20.3, pp. 199–207. ISSN: 1388-3690. DOI: 10.1007/s10990-007-9018-9. (Visited on 04/28/2019).

Landin, P. J. (1964). "The Mechanical Evaluation of Expressions". In: *The Computer Journal* 6.4, pp. 308–320.

— (1965a). "A Correspondence Between ALGOL 60 and Church's Lambda-notation: Part I". In: *Commun. ACM* 8.2, pp. 89–101.

— (1965b). "A Correspondence Between ALGOL 60 and Church's Lambda-notations: Part II". In: *Commun. ACM* 8.3, pp. 158–167.

Laurent, Olivier (2001). "A Token Machine for Full Geometry of Interaction". In: *Typed Lambda Calculi and Applications, 5th International Conference, TLCA 2001, Krakow, Poland, May 2-5, 2001, Proceedings*. Ed. by Samson Abramsky. Vol. 2044. Lecture Notes in Computer Science. Springer, pp. 283–297. DOI: 10.1007/3-540-45413-6_23.

Leroy, Xavier (1990). *The ZINC experiment: an economical implementation of the ML language*. Technical report 117. INRIA.

Mackie, Ian (1995). "The Geometry of Interaction Machine". In: *Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23-25, 1995*. Ed. by Ron K. Cytron and Peter Lee. ACM Press, pp. 198–208. DOI: 10.1145/199448.199483.

— (2017). "A Geometry of Interaction Machine for Gödel's System T". In: *Logic, Language, Information, and Computation - 24th International Workshop, WoLLIC 2017, London, UK, July 18-21, 2017, Proceedings*. Ed. by Juliette Kennedy and Ruy J. G. B. de Queiroz. Vol. 10388. Lecture Notes in Computer Science. Springer, pp. 229–241. DOI: 10.1007/978-3-662-55386-2_16.

Mascari, Gianfranco and Marco Pedicini (1994). "Head Linear Reduction and Pure Proof Net Extraction". In: *Theor. Comput. Sci.* 135.1, pp. 111–137. DOI: 10.1016/0304-3975(94)90263-1. URL: https://doi.org/10.1016/0304-3975(94)90263-1.

Mazza, Damiano (2015). "Simple Parsimonious Types and Logarithmic Space". In: *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*. Ed. by Stephan Kreutzer. Vol. 41. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 24–40. DOI: 10.4230/LIPIcs.CSL.2015.24.

Mazza, Damiano, Luc Pellissier, and Pierre Vial (2018). "Polyadic approximations, fibrations and intersection types". In: *Proc. ACM Program. Lang.* 2.POPL, 6:1–6:28. DOI: 10.1145/3158094.

Mazza, Damiano and Kazushige Terui (2015). "Parsimonious Types and Non-uniform Computation". In: *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*. Ed. by Magnús M. Halldórsson et al. Vol. 9135. Lecture Notes in Computer Science. Springer, pp. 350–361. DOI: 10.1007/978-3-662-47666-6_28.

Milner, Robin (1977). "Fully Abstract Models of Typed *lambda*-Calculi". In: *Theor. Comput. Sci.* 4.1, pp. 1–22. DOI: 10.1016/0304-3975(77)90053-6. URL: https://doi.org/10.1016/0304-3975(77)90053-6.

— (2007). "Local Bigraphs and Confluence: Two Conjectures". In: *Electronic Notes in Theoretical Computer Science* 175.3, pp. 65–73.

Muroya, Koko and Dan R. Ghica (2017). "The Dynamic Geometry of Interaction Machine: A Call-by-Need Graph Rewriter". In: *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*. Ed. by Valentin Goranko and Mads Dam. Vol. 82. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 32:1–32:15. ISBN: 978-3-95977-045-3. DOI: 10.4230/LIPIcs.CSL.2017.32.

— (2019). "The Dynamic Geometry of Interaction Machine: A Token-Guided Graph Rewriter". In: *Log. Methods Comput. Sci.* 15.4. URL: https://lmcs.episciences.org/5882.

Neergaard, Peter Møller and Harry G. Mairson (2004). "Types, potency, and idempotency: why nonlinearity and amnesia make a type system work". In: *Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming, ICFP 2004, Snow Bird, UT, USA, September 19-21, 2004*. Ed. by Chris Okasaki and Kathleen Fisher. ACM, pp. 138–149. DOI: 10.1145/1016850.1016871.

Olimpieri, Federico (2021). "Intersection Type Distributors". In: *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, pp. 1–15. DOI: 10.1109/LICS52264.2021.9470617. URL: https://doi.org/10.1109/LICS52264.2021.9470617.

Paraskevopoulou, Zoe and Andrew W. Appel (2019). "Closure conversion is safe for space". In: *Proc. ACM Program. Lang.* 3.ICFP, 83:1–83:29.

Plotkin, Gordon D. (1975). "Call-by-Name, Call-by-Value and the lambda-Calculus". In: *Theor. Comput. Sci.* 1.2, pp. 125–159. DOI: 10.1016/0304-3975(75)90017-1.

Regnier, Laurent (1994). "Une équivalence sur les lambda- termes". In: *Theoretical Computer Science* 126.2, pp. 281 –292.

Sands, David (1996). "Total Correctness by Local Improvement in the Transformation of Functional Programs". In: *ACM Trans. Program. Lang. Syst.* 18.2, pp. 175–234.

Sands, David, Jörgen Gustavsson, and Andrew Moran (2002). "Lambda Calculi and Linear Speedups". In: *The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones [on occasion of his 60th birthday]*. Ed. by Torben Æ. Mogensen, David A. Schmidt, and Ivan Hal Sudborough. Vol. 2566. Lecture Notes in Computer Science. Springer, pp. 60–84. DOI: 10.1007/3-540-36377-7_4.

Sansom, Patrick M. and Simon L. Peyton Jones (1995). "Time and Space Profiling for Non-Strict Higher-Order Functional Languages". In: *POPL*. ACM Press, pp. 355–366.

Schöpp, Ulrich (2006). "Space-Efficient Computation by Interaction". In: *CSL*. Vol. 4207. Lecture Notes in Computer Science. Springer, pp. 606–621.

— (2007). "Stratified Bounded Affine Logic for Logarithmic Space". In: *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*. IEEE Computer Society, pp. 411–420. DOI: 10.1109/LICS.2007.45.

— (2014). "On the Relation of Interaction Semantics to Continuations and Defunctionalization". In: *Logical Methods in Computer Science* 10.4. DOI: 10.2168/LMCS-10(4:10)2014.

— (2015). "From Call-by-Value to Interaction by Typed Closure Conversion". In: *Programming Languages and Systems - 13th Asian Symposium, APLAS 2015, Pohang, South Korea, November 30 - December 2, 2015, Proceedings*. Ed. by Xinyu Feng and Sungwoo Park. Vol. 9458. Lecture Notes in Computer Science. Springer, pp. 251–270. DOI: 10.1007/978-3-319-26529-2_14.

Schönfinkel, Moses (1924). "Über die Bausteine der mathematischen Logik". In: *Mathematische Annalen* 92, pp. 305–316.

Sestoft, Peter (1997). "Deriving a Lazy Abstract Machine". In: *J. Funct. Program.* 7.3, pp. 231–264.

Slot, Cees F. and Peter van Emde Boas (1988). "The Problem of Space Invariance for Sequential Machines". In: *Inf. Comput.* 77.2, pp. 93–122. DOI: 10.1016/0890-5401(88)90052-1. URL: https://doi.org/10.1016/0890-5401(88)90052-1.

Spoonhower, Daniel et al. (2010). "Space profiling for parallel functional programs". In: *J. Funct. Program.* 20.5-6, pp. 417–461. DOI: 10.1017/S0956796810000146.

Tsukada, Takeshi, Kazuyuki Asada, and C.-H. Luke Ong (2017). "Generalised species of rigid resource terms". In: *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, pp. 1–12. DOI: 10.1109/LICS.2017.8005093.

Tsukada, Takeshi and C.-H. Luke Ong (2016). "Plays as Resource Terms via Non-idempotent Intersection Types". In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*. Ed. by Martin Grohe, Eric Koskinen, and Natarajan Shankar. ACM, pp. 237–246. DOI: 10.1145/2933575.2934553. URL: https://doi.org/10.1145/2933575.2934553.

Turing, A. M. (1937). "Computability and λ-Definability". In: *Journal of Symbolic Logic* 2.4, pp. 153–163.

Wadsworth, C. P. (1980). "Some unusual λ-calculus numeral systems". In: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Ed. by J. P. Seldin and J. R. Hindley, pp. 215–230.

Wand, Mitchell (2007). "On the correctness of the Krivine machine". In: *High. Order Symb. Comput.* 20.3, pp. 231–235.