Interacting Seems Unreasonable, in Time and Space - Extended Abstract -

Beniamino Accattoli — Inria & École Polytechnique
Ugo Dal Lago — Università di Bologna & Inria
Gabriele Vanoni — Università di Bologna & Inria

Girard's Geometry of Interaction (GoI) can be made concrete by considering it as an implementation technique for functional programs, in particular the λ-calculus. Our work is about the complexity analysis of the interaction abstract machine (IAM), an abstract machine based on the GoI. In previous works, we have adapted de Carvalho's non-idempotent intersection types so as to precisely characterize the time and space complexity of the IAM, thus providing a logical account of the IAM resource usage. Here we show, by the way of the type systems we have introduced, that the IAM is unlikely to be reasonable in time nor in space according to Slot and van Emde Boas' invariance thesis.

1 Introduction

Intersection types [9, 7] (IT in the following), the geometry of interaction [16] (GoI), and cost models [1] are three topics in the theory of the λ-calculus [6] that are not generally studied together. In our talk, we are going to shed new light on the connections between these three research areas. This is only the last chapter of a larger project directed at comprehending the time and space complexity of λ-calculus implementation schemes.

Intersection Types. Intersection types are the core topic of ITRS workshop series, and for this reason we shall review them only quickly. We are going to use the non-idempotent version, as pioneered by Gardner [13] and de Carvalho [8]. In particular, we are inspired by de Carvalho's use of IT derivations to study the complexity of evaluating λ-terms using the Krivine Abstract Machine [18]. Here, we shall use IT to study the time and space complexities of the evaluating λ-terms using the *Interaction Abstract Machine* [20, 11, 4] (IAM), a machine inspired by the GoI. Technically, in order to capture the IAM space consumption, we need to refine the type system dropping also the associativity of the intersection operator, thus refining the multiset structure of non-idempotent intersections into a *tree* structure.

The Interaction Abstract Machine. The advantage, and at the same time the drawback, of the λ-calculus is its distance from low-level implementation details. It comes with just one rule, β-reduction, and with no indication about how to implement it on low-level machines. It is an advantage when *reasoning* about programs expressed as λ-terms. It is a drawback, instead, when one wants to *implement* the λ-calculus, or to do complexity analyses, because β-steps are far from being atomic operations. In particular, terms can grow exponentially with the number of β-steps, a degeneracy known as *size explosion*, which is why β-reduction cannot be reasonably implemented, at least if one sticks to an explicit representation of λ-terms. Abstract machines are automata-theoretic models that are *abstract* enough to be hardware independent, but *machines* in the sense that they are *reasonable* models, *i.e.* they can be used to perform complexity analyses. The theory of linear logic [15] provides the foundation to build an unusual abstract machine, rooted in Girard's Geometry of Interaction [17]. The Interaction

Submitted to:
SOS 2007

Abstract Machine (IAM) was pioneered by Mackie and Danos & Regnier in the nineties [20, 11]. Differently from traditional environment machines, such as Krivine's, the IAM does not use environments, and instead uses 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, GoI machines are good candidates for space-efficient implementation schemes, as first shown by Schöpp and coauthors [23, 10]. The price to pay is that the machine wastes a lot of time to retrieve β-redexes, so that time is sacrificed for space.

(Reasonable) Cost Models. What does exactly mean for a cost measure to be reasonable? First of all, one has to fix a computational theory T, whose role in our case is played by the λ-calculus. As proposed by Slot and van Emde Boas [24], a cost model for T is *reasonable* when there are simulations of T by Turing machines and of Turing machines by T having overhead bounded by

- *Time*: a polynomial on both the input and the time cost in the source theory.
- *Space*: a function linear in the space cost in the source theory.

The basic idea is to preserve the complexity classes P and L, that then become *robust*, that is, theory-independent.

Unitary Time Cost Models and the IAM. For the λ-calculus, time cost models of interest are usually given by the number of β-steps taken by a fixed strategy. Since they count 1 for every such β-steps, these time cost models are called *unitary*. There is an encoding of Turing machines into a fragment of the λ-calculus, due to Dal Lago and Accattoli, for which many strategies 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 Turing machines, or another reasonable model, typically random access machines. Often, this is done using an abstract machine and showing that the overhead of the abstract machine is polynomial in the number of β-steps and the size of the input. Note that in this way one shows, at the same time, that also the number of steps of the machine is a reasonable cost model for the λ-calculus—the abstract machine is then called *reasonable*.

Concerning the IAM, it is known since the first works by Mackie and Danos & Regnier that its overhead can be exponential in the number of β-steps of the implemented strategy. Therefore, it cannot be a tool to show that a 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.

Space Cost Models and the IAM. In the literature, there are no reasonable space cost model for the λ-calculus. There is a recent partial result by Forster, Kunze, and Roth [12] stating that the size of λ-terms is a reasonable space cost model for the weak call-by-value λ-calculus. The result is partial because such a cost model can only measure linear and super-linear space, while the main space class of interest is the class L of logarithmic space, which is sub-linear. Therefore, the general problem is still open.

The IAM has been used in the literature for obtaining sub-linear space bounds for functional programs (Dal Lago and Schöpp [23, 10], Mazza [21] and Ghica [14]). These results have then suggested the folklore conjecture that the IAM space usage is a reasonable space cost model.

Having a close 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 λ-terms. A further evidence of the ambiguous space behavior of the IAM is that the folklore conjecture has been circulating among specialists for years, but has never found an answer.

Two Negative Results. The main contributions of this work are negative results about the possibility of proving the IAM reasonable with respect to its natural cost models. More specifically, through the use of the type systems we have crafted [5, 3], we are able to precisely capture, in a compositional way, the space and time consumption of the IAM. This allows us to devise type schemes for the fixed point combinator at work in Dal Lago and Accattoli's encoding of Turing machines into the λ-calculus, which is the fundamental ingredient of the encoding. The obtained type schemes show that a Turing machine taking n steps to halt is encoded into a λ-term taking time *exponential* in n and space *linear* in n, when evaluated with the IAM. These facts, together, suggest that the IAM is *unreasonable*. In particular, the space of the IAM is linear in the time (rather than the space) of the Turing machine, which forbids to prove any correlation with the space of the Turing machine. More precisely, the IAM is unreasonable with respect to Dal Lago and Accattoli's encoding of Turing machines into the pure λ-calculus [19]. While of course there might in principle be other encodings which could prove the IAM reasonable, this seems to be unlikely. On the one hand, the encoding of reference is very natural. On the other hand, all known encodings of a Turing-complete theory into the λ-calculus use fixpoint combinators to implement recursion, and thus are likely to suffer of the same problem.

2 The Interaction Abstract Machine

In this section we provide an overview of the Interaction Abstract Machine (IAM). We adopt the λ-calculus presentation of the IAM, rather called λIAM (we refer to [4] for an in-depth study of the λIAM). The literature usually studies the (λ)IAM with respect to head evaluation of potentially open terms, here we only deal with Closed CbN, which is closer to the practice of functional programming.

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

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

Free and *bound variables* are defined as usual: $\lambda x.t$ binds x in t. A term is *closed* when there are no free occurrences of variables in it. Terms are considered modulo α-equivalence, and capture-avoiding (meta-level) substitution of all the free occurrences of x for u in t is noted $t\{x\leftarrow u\}$. Contexts are just λ-terms containing exactly one occurrence of a special symbol, the *hole* $\langle\cdot\rangle$, intuitively standing for a removed subterm. Here we adopt *leveled* contexts, whose index, *i.e.* the level, stands for the number of arguments (that is, the number of !-boxes in linear logic terminology) the hole lies in.

$$\begin{array}{rcl} & \text{LEVELED CONTEXTS} & \\ C_0 & ::= & \langle\cdot\rangle \;\big|\; \lambda x.C_0 \;\big|\; C_0 t; \\ C_{n+1} & ::= & C_{n+1}t \;\big|\; \lambda x.C_{n+1} \;\big|\; tC_n. \end{array}$$

We simply write C for a context whenever the level is not relevant. 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*.

The operational semantics that we adopt here is weak head evaluation $\to_{wh}$, defined as follows:

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

where $t\{y\leftarrow u\}$ is our notation for meta-level substitution. We further restrict the setting by considering only closed terms, and refer to our framework as *Closed Call-by-Name* (shortened to Closed CbN). Basic

LOGGED POSITIONS	$l ::= (t, C_n, L_n)$	TAPES	$T ::= \varepsilon \mid \bullet\cdot T \mid l\cdot T$
LOGS	$L_0 ::= \varepsilon \quad L_{n+1} ::= l\cdot L_n$	DIRECTION	$d ::= \downarrow \mid \uparrow$
STATES	$s ::= (t, C, L, T, d)$		

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	$(x,\lambda x.D_n,L_n)\cdot T$
$\underline{\lambda x.D_n\langle x\rangle}$	C	L	$(x,\lambda x.D_n,L_n)\cdot T$	$\to_{\mathsf{bt2}}$	x	$\underline{C\langle\lambda x.D_n\rangle}$	$L_n\cdot L$	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	$\underline{C\langle u\langle\cdot\rangle\rangle}$	$l\cdot L$	T	$\to_{\mathsf{bt1}}$	$\underline{u}$	$C\langle\langle\cdot\rangle t\rangle$	L	$l\cdot T$

Figure 1: Data structures and transitions of the λ Interaction Abstract Machine (λIAM).

well known facts are that in Closed CbN the normal forms are precisely the abstractions and that $\to_{wh}$ is deterministic.

The machine considered in this paper moves over the code without ever changing it. A *position* in a term t is represented as a pair (u, C) of a sub-term u and a context C such that $C\langle u\rangle = t$. Keep in mind that the λIAM is an unusual machine which is *not* based on environments, and that finding it hard to grasp is normal. Also, in [4] there is an alternative explanation of the λIAM, that may be helpful, together with the relationship with proof nets, which is however not needed here.

A Bird's Eye view of the λIAM. Intuitively, the behaviour of the λIAM can be seen as that of a token that travels around the syntax tree of the program under evaluation. Similarly to environment machines such as Krivine's, it looks for the head variable of a term. The peculiarity of the λIAM is that it does not store the encountered β-redexes in an environment. When it finds the head variable, the λIAM looks for the argument which should replace it, because having no environment, it cannot simply look it up. These two search mechanisms are realized by two different phases and directions of exploration of the code, noted $\downarrow$ and $\uparrow$. The functioning is actually more involved because there is also a backtracking mechanism (which however has nothing to do with backtracking as modeled by classical logic and continuations), requiring to save and manipulate code positions in the token. Last, the machine never duplicates the code, but it distinguishes different uses of a same code (position) using *logs*. There are no easy intuitions about how logs handle different uses—this is both the magic and the mystery of the geometry of interaction.

λIAM States. The data structures of the λIAM are in Fig. 1. The λIAM travels on a λ-term t carrying data structures—representing the token—storing information about the computation and determining the next transition to apply. It travels according to a *direction* of navigation that is either $\downarrow$ or $\uparrow$, pronounced *down* and *up*. The *token* is given by two stacks, called *log* and *tape*, whose main components are *logged positions*. Roughly, a log is a trace of the relevant positions in the history of a computation, and a logged position is a position plus a log, meant to trace the history that led to that position. Logs and logged positions are defined by mutual induction. Note that in the definition of a logged position, the log is required to have length n, where n is the level of the context of the position. We use $\cdot$ also to concatenate

logs, writing, *e.g.*, $L_n \cdot L$, using L for a log of unspecified length. The *tape* T is a list of logged positions plus occurrences of the special symbol $\bullet$, needed to record the crossing of abstractions and applications.

A *state* of the machine is given by a position and a token (that is, a log L and a tape T), together with a *direction*. Initial states have the form $s_t := (\underline{t}, \langle\cdot\rangle, \varepsilon, \varepsilon)$. Directions are often omitted and represented via colors and underlining: $\downarrow$ is represented by a red and underlined code term, $\uparrow$ by a blue and underlined code context.

Transitions. The transitions of the λIAM are in Fig. 1. Their union is noted $\to_{\lambda\text{IAM}}$. The idea is that $\downarrow$-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. A key point is that navigation is done locally, moving only between adjacent positions[1]. Intuitively, the machine evaluates the term t until the head abstraction of the head normal form is found.

The transitions realize three entangled mechanisms.

Mechanism 1: Search Up to β-Redexes. Note that $\to_{\bullet 1}$ skips the argument and adds a $\bullet$ on the tape. The idea is that $\bullet$ keeps track that an argument has been encountered—its identity is however forgotten. Then $\to_{\bullet 2}$ does the dual job: it skips an abstraction when the tape carries a $\bullet$, that is, the trace of a previously encountered argument. Note that, when the λIAM moves through a β-redex with the two steps one after the other, the token is left unchanged. This mechanism thus realizes search *up to β-redexes*, that is, without ever recording them. Note that $\to_{\bullet 3}$ and $\to_{\bullet 4}$ do the same during the $\uparrow$ phase.

Let us illustrate this mechanism with an example: the first steps of evaluation on the term $\mathsf{I}((\lambda x.xx)\mathsf{I})$, where I is the identity combinator. One can notice that the λIAM traverses one β-redexes without altering the token, that is empty both at the beginning and at the end.

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

Mechanism 2: Finding Variables and Arguments. As a first approximation, navigating in direction $\downarrow$ corresponds to looking for the head variable of the term code, while navigating with direction $\uparrow$ corresponds to looking for the sub-term to replace the previously found head variable, what we call *the argument*. More precisely, when the head variable x of the active subterm is found, transition $\to_{\mathsf{var}}$ switches direction from $\downarrow$ to $\uparrow$, and the machine starts looking for potential substitutions for x. The λIAM then moves to the position of the binder λx of x, and starts exploring the context C, looking for the first argument up to β-redexes. The relative position of x w.r.t. its binder is recorded in a new logged position that is added to the tape. Since the machine moves out of a context of level n, namely D_n, the logged position contains the first n logged positions of the log. Roughly, this is an encoding of the run that led from the level of $\lambda x.D_n\langle x\rangle$ to the occurrence of x at hand, in case the machine would later need to backtrack.

When the argument t for the abstraction binding the variable x in l is found, transition $\to_{\mathsf{arg}}$ switches direction from $\uparrow$ to $\downarrow$, making the machine looking for the head variable of t. Note that moving to t, the level increases, and that the logged position l is moved from the tape to the log. The idea is that l is now

[1] Transitions $\to_{\mathsf{var}}$ and $\to_{\mathsf{bt2}}$ may note look as local, as they jump from a bound variable occurrence to its binder, and viceversa. If λ-terms are represented by implementing occurrences as pointers to their binders, as in the proof net representation of λ-terms—upon which some concrete implementation schemes are based, see [2]—then they are local.

a completed argument query, and it becomes part of the history of how the machine got to the current position, to be potentially used for backtracking. We continue the example of the previous point: the machine finds the head variable z and looks for its argument in $\uparrow$ mode. When it has been found, the direction turns to $\downarrow$ again and the process continues as before: first the head variable is found and then the machine looks for its argument. Let us set $l_z := (z, (\lambda z.\langle\cdot\rangle)((\lambda x.xx)\mathsf{I}), \varepsilon)$, $l_{\langle\cdot\rangle x} := (x, \lambda x.\langle\cdot\rangle x, \varepsilon)$ and $l_y := (y, \lambda y.\langle\cdot\rangle, \varepsilon)$.

Sub-term	Context	Log	Tape	Dir
z	$(\lambda z.\langle\cdot\rangle)((\lambda x.xx)\mathsf{I})$	ε	ε	$\downarrow$
$\to_{\mathsf{var}}\ \lambda z.z$	$\underline{\langle\cdot\rangle((\lambda x.xx)\mathsf{I})}$	ε	l_z	$\uparrow$
$\to_{\mathsf{arg}}\ \underline{(\lambda x.xx)\mathsf{I}}$	$\mathsf{I}\langle\cdot\rangle$	l_z	ε	$\downarrow$
$\to_{\bullet 1}\ \underline{\lambda x.xx}$	$\mathsf{I}(\langle\cdot\rangle\mathsf{I})$	l_z	$\bullet$	$\downarrow$
$\to_{\bullet 2}\ \underline{xx}$	$\mathsf{I}((\lambda x.\langle\cdot\rangle)\mathsf{I}$	l_z	ε	$\downarrow$
$\to_{\bullet 1}\ \underline{x}$	$\mathsf{I}((\lambda x.\langle\cdot\rangle x)\mathsf{I})$	l_z	$\bullet$	$\downarrow$
$\to_{\mathsf{var}}\ \lambda x.xx$	$\underline{\mathsf{I}(\langle\cdot\rangle(\lambda y.y))}$	l_z	$l_{\langle\cdot\rangle x}\cdot\bullet$	$\uparrow$
$\to_{\mathsf{arg}}\ \underline{\lambda y.y}$	$\mathsf{I}((\lambda x.xx)\langle\cdot\rangle)$	$l_{\langle\cdot\rangle x}\cdot l_z$	$\bullet$	$\downarrow$
$\to_{\bullet 2}\ \underline{y}$	$\mathsf{I}((\lambda x.xx)(\lambda y.\langle\cdot\rangle))$	$l_{\langle\cdot\rangle x}\cdot l_z$	ε	$\downarrow$
$\to_{\mathsf{var}}\ \lambda y.y$	$\underline{\mathsf{I}((\lambda x.xx)\langle\cdot\rangle)}$	$l_{\langle\cdot\rangle x}\cdot l_z$	l_y	$\uparrow$

Mechanism 3: Backtracking. It is started by transition $\to_{\mathsf{bt1}}$. The idea is that the search for an argument of the $\uparrow$-phase has to temporarily stop, because there are no arguments left at the current level. The search of the argument then has to be done among the arguments of the variable occurrence that triggered the search, encoded in l. Then the machine enters into backtracking mode, which is denoted by a $\downarrow$-phase with a logged position on the tape, to reach the position in l. Backtracking is over when $\to_{\mathsf{bt2}}$ is fired.

The $\downarrow$-phase and the logged position on the tape mean that the λIAM is backtracking. During backtracking, the machine is not looking for the head variable of the current code $\lambda x.t$, it is rather going back to the variable position in the tape, to find its argument. This is realized by moving to the position in the tape and changing direction. Moreover, the log L_n encapsulated in the logged position is put back on the global log. An invariant guarantees that the logged position on the tape always contains a position relative to the active abstraction.

In our example, a backtracking phase starts when the λIAM looks for the argument of y. Since $\lambda y.y$ has been virtually substituted for x, its argument is actually the second occurrence of x. Backtracking retrieves the variable which a term was virtually substituted for.

Sub-term	Context	Log	Tape	Dir
$\lambda y.y$	$\underline{\mathsf{I}((\lambda x.xx)\langle\cdot\rangle)}$	$l_{\langle\cdot\rangle x}\cdot l_z$	l_y	$\uparrow$
$\to_{\mathsf{bt1}}\ \underline{\lambda x.xx}$	$\mathsf{I}(\langle\cdot\rangle\mathsf{I})$	l_z	$l_{\langle\cdot\rangle x}\cdot l_y$	$\downarrow$
$\to_{\mathsf{bt2}}\ x$	$\underline{\mathsf{I}((\lambda x.\langle\cdot\rangle x)\mathsf{I})}$	l_z	l_y	$\uparrow$
$\to_{\mathsf{arg}}\ \underline{x}$	$\mathsf{I}((\lambda x.x\langle\cdot\rangle)\mathsf{I})$	$l_y\cdot l_z$	ε	$\downarrow$

For the sake of completeness, we conclude the example, which runs until the head abstraction of the weak head normal form of the term under evaluation, namely the second occurrence of I, is found. We set $l_{x\langle\cdot\rangle} := (x, \lambda x.x\langle\cdot\rangle, l_y)$.

Sub-term	Context	Log	Tape	Dir
$\underline{x}$	$\mathsf{I}((\lambda x.x\langle\cdot\rangle)\mathsf{I})$	$l_y\cdot l_z$	ε	$\downarrow$
$\to_{\mathsf{var}}\ \lambda x.xx$	$\underline{\mathsf{I}(\langle\cdot\rangle(\lambda y.y))}$	l_z	$l_{x\langle\cdot\rangle}$	$\uparrow$
$\to_{\mathsf{arg}}\ \underline{\lambda y.y}$	$\mathsf{I}((\lambda x.xx)\langle\cdot\rangle)$	$l_{x\langle\cdot\rangle}\cdot l_z$	ε	$\downarrow$

Final States. If the λIAM starts on the initial state s_t, the execution may either never stop or end in a state s of the shape $s = (\underline{\lambda x.u}, C, L, \varepsilon)$. The fact that no other shapes are possible for s is proved in [4].

A *run* $\rho : s \to^* s'$ is a possibly empty sequence of transitions, whose length is noted $|\rho|$. A run is *complete* if it ends in a final state.

Implementation. Usually, the λIAM is shown to implement (a micro-step variant of) head reduction. The details are quite different from those in the usual notion of implementation for environment machines, such as the KAM. Essentially, it is shown that the λIAM induces a semantics $[\![\cdot]\!]_{\lambda\text{IAM}}$ of terms that is a sound and adequate with respect to head reduction, rather than showing a bisimulation between the machine and head reduction—this is explained at length in [4]. For the sake of simplicity, here we restrict to Closed CbN. The λIAM semantics then reduces to observing termination: $[\![t]\!]_{\lambda\text{IAM}}$ is defined if and only if weak head reduction terminates on t. Therefore, we avoid discussing semantics and only study termination. We say that the *λIAM implements Closed CbN* when its execution from the initial state s_t reaches a final state if and only if $\to_{wh}$ terminates on t, for every closed term t.

Theorem 2.1 ([4])**.** *The λIAM implements Closed CbN.*

λIAM Space Consumption. This work wants to analyze not only the time but also the space consumption of the λIAM. First, we have to define the space consumed by a state (the meta-variable Γ to denote either a tape T or a log L):

$$|(t,C,L,T,d)|_{\mathsf{sp}} := |L|_{\mathsf{sp}} + |T|_{\mathsf{sp}} \qquad ^{2}$$

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

The value of the unknown X is simply the size of a pointer to a subterm of the term under evaluation, *i.e.* $\mathsf{X} = \log|t|$, if the λIAM is evaluating the term t. 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 2.2. *Let $\rho : s_0 \to^*_{\lambda\text{IAM}} s$ be a λIAM run. Then, $|\rho|_{\mathsf{sp}} := \max_{s'\in\rho} |s'|_{\mathsf{sp}}$.*

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.

Proposition 2.3. *Let ρ be an infinite λIAM run. Then $|\rho|_{\mathsf{sp}} = \infty$.*

3 Tree (Intersection) Types

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

From Intersections Types to Tree Types The framework that we adopt is the one of intersection types, with three tweaks:

1. *No idempotency*: we use the non-idempotent variant, where the intersection type $A \wedge A$ is not equivalent to A, and with strong 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. Non-idempotent intersections are multisets, which is why these types are sometimes called *multi types* and an intersection $A \wedge B \wedge A$ is rather noted $[A,B,A]$.

[2] Please note that considering space this way, we are not counting the pointer to the code term/context (t,C), which consumes space X, and the direction bit d, which consumes 1. We can do this simplification w.l.o.g. because their sizes are constant, namely $\mathsf{X}+1$.

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

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

Figure 2: The tree type system.

2. *Nesting/tree shape*: multi 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.
3. *No commutativity*: we also consider *non-commutative* tree types. Removing commutativity turns multisets into lists, or sequences, and trees into ordered trees. Removing commutativity is an inessential tweak. Our study does not depend on the ordered structure, we only need bijections between multisets, to describe the TIAM, and these bijections are just more easily managed if commutativity is removed. This *rigid* approach is also used by Tsukada, Asada, and Ong [25] and Mazza, Pellissier, and Vial [22]. The inessential aspect is stressed by referring to our types as to *tree types*, rather than as to *ordered tree types*, despite adopting the ordered variant.

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

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

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 $T \to A$ can have a tree type only on the left. About trees, since commutativity is ruled out, we have, for instance, that $[A, A'] \neq [A', A]$. 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 $T \uplus T'$.

Type judgments have the form $\Gamma \vdash t : G$, where Γ is a type environment, defined below. Type derivations are noted π and we write $\pi \triangleright \Gamma \vdash t : G$ for a type derivation π of ending judgment $\Gamma \vdash t : G$. Type environments, ranged over by Γ, Δ are total maps from variables to tree types such that only finitely many variables are mapped to non-empty tree types, and we write $\Gamma = x_1 : T_1, \ldots, x_n : T_n$ if $dom(\Gamma) = \{x_1, \ldots, x_n\}$—note that type environments *are* commutative, what is non-commutative is the sequence constructor $[\cdot]$, only.

The typing rules are in Fig. 2. With respect to the literature, the difference is in the 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 $\Gamma = x_1 : T_1, \ldots, x_n : T_n$.

Given two type environments Γ, Δ, we use $\Gamma \uplus \Delta$ for the type environment assigning to every variable x the list $\Gamma(x) \uplus \Delta(x)$. We keep using $|T|$ for the length of T as a sequence, that is, $|[G_1, .., G_n]| = n$. The

size $|\pi|$ of a tree types derivation π is the number of its rules that are not $\mathrm{T}_{\mathrm{MANY}}$.

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[3] for multi types, without surprises.

Theorem 3.1 (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 types derivation $\pi \triangleright \vdash t : \star$.*

Relationship with Multi Types. It is easy to define 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.

4 Characterizing λIAM Time and Space Consumption

This section explains how to exploit tree types to measure the time and the space consumed by the complete λIAM run on the term t via a quantitative analysis of the tree type derivation for t. In particular tree type derivations have to be *weighted*, in such a way that they precisely reflect the resource consumption of the typed term. Basically, the intuition is that every occurrence of $\star$ in a type derivation stands for a state of the λIAM. Actually, one could define a machine strongly bisimilar to the λIAM directly on top of type derivations. This is explained in detail in [5, 3].

Time Consumption. First, we need to define a measure on types. Morally, it counts the number of $\star$ occurrences in a type.

$$\|\star\| := 1 \qquad \|T \to A\| := \|T\| + \|A\| \qquad \big\|[G_1, \dots, G_n]\big\| := \textstyle\sum_{1 \le i \le n} \|G_i\|$$

Then, we are able to define the weighted type system, whose weights count the total number of $\star$ occurrences in the type derivation[4]. It is presented in Figure 3 (type environments are omitted, since they do not influence weights).

Theorem 4.1 (λIAM time via tree types)**.** *Let t be a closed term that is $\to_{wh}$-normalizable, σ the complete λIAM run from s_t, and $\pi \triangleright \overset{n}{\vdash}_{\mathsf{ti}} t : \star$ a type derivation for t. Then $|\sigma| = n$.*

[3] 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-$\lambda_\star$).

[4] Actually, it counts all the $\star$ occurrences *minus one*. Indeed, the weight associated to rule T-$\lambda_\star$, which represents the final state, is *zero*. This way, the final weight of a derivation reflects the run time. In fact, if in a derivation n states are reached, its length is $n-1$.

$$\frac{}{\vdash_{\mathsf{ti}}^{\|A\|} x : A}\ \text{T-VAR} \qquad \frac{x : S \vdash_{\mathsf{ti}}^{w} t : A}{\vdash_{\mathsf{ti}}^{w+\|T\to A\|} \lambda x.t : T \to A}\ \text{T-}\lambda \qquad \frac{\vdash_{\mathsf{ti}}^{v} t : T \to A \quad \vdash_{\mathsf{ti}}^{w} u : T}{\vdash_{\mathsf{ti}}^{v+w+\|A\|} tu : A}\ \text{T-@}$$

$$\frac{}{\vdash_{\mathsf{ti}}^{0} \lambda x.t : \star}\ \text{T-}\lambda_\star \qquad \frac{\vdash_{\mathsf{ti}}^{w_i} t : G_i \quad 1 \le i \le n}{\vdash_{\mathsf{ti}}^{\sum w_i} t : [G_1,..,G_n]}\ \text{T-MANY} \qquad \frac{}{\vdash_{\mathsf{ti}}^{0} t : [\cdot]}\ \text{T-NONE}$$

$$\frac{}{x : [A] \vdash_{\mathsf{sp}}^{(\!|A|\!)} x : A}\ \text{T-VAR} \qquad \frac{x : T \vdash_{\mathsf{sp}}^{w} t : A}{\vdash_{\mathsf{sp}}^{\max\{w,(\!|T\to A|\!)\}} \lambda x.t : T \to A}\ \text{T-}\lambda \qquad \frac{\vdash_{\mathsf{sp}}^{w} t : T \to A \quad \vdash_{\mathsf{sp}}^{v} u : T}{\vdash_{\mathsf{sp}}^{\max\{w,v\}} tu : A}\ \text{T-@}$$

$$\frac{}{\vdash_{\mathsf{sp}}^{0} \lambda x.t : \star}\ \text{T-}\lambda_\star \qquad \frac{\vdash_{\mathsf{sp}}^{w_i} t : G_i \quad 1 \le i \le n}{\vdash_{\mathsf{sp}}^{\mathsf{X}+\max_i\{w_i\}} t : [G_1,..,G_n]}\ \text{T-MANY} \qquad \frac{}{\vdash_{\mathsf{sp}}^{0} t : [\cdot]}\ \text{T-NONE}$$

Figure 3: The weighted tree type system capturing time $\vdash_{\mathsf{ti}}$ (up) and space $\vdash_{\mathsf{sp}}$ (down).

Space Consumption. In order to take space consumption into account, we need to define another measure on types.

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

The unknown X again stands for the size of a pointer to the term under evaluation, or in the case of type derivations, of the term occuring in the final judgment. Then, in the same way we did for time, we can define a weighted type system that takes space consumption into account, in Figure 3.

Theorem 4.2 (λIAM space via tree types)**.** *Let t be a closed term that is $\to_{wh}$-normalizable, σ the complete λIAM run from s_t, and $\pi \triangleright \vdash_{\mathsf{sp}}^{n} t : \star$ a type derivation for t. Then $|\sigma|_{\mathsf{sp}} = n$.*

An example. In Fig. 4 we present the very same example analyzed in Section 2. We have reported its type derivation in the system $\vdash_{\mathsf{sp}}$, with the occurrences of $\star$ on the right of $\vdash_{\mathsf{sp}}$ annotated with increasing integers and a direction. The occurrence of $\star$ marked with 1 represents the first state of the λIAM executing the same term, and so on. 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 2. Please note that we have considered $\max\{1,\mathsf{X}\} = \mathsf{X}$ when assigning the weights, since here $\mathsf{X} = \log|(\lambda z.z)((\lambda x.xx)(\lambda y.y))|$.

5 Typing Turing Machines

In order to prove the λIAM reasonable for both time and space we need to show a simulation of a reasonable model by the λIAM with a polynomially bounded overhead in time and a constant overhead in space. Turing machines are the most natural reasonable model to simulate. In this section, we show

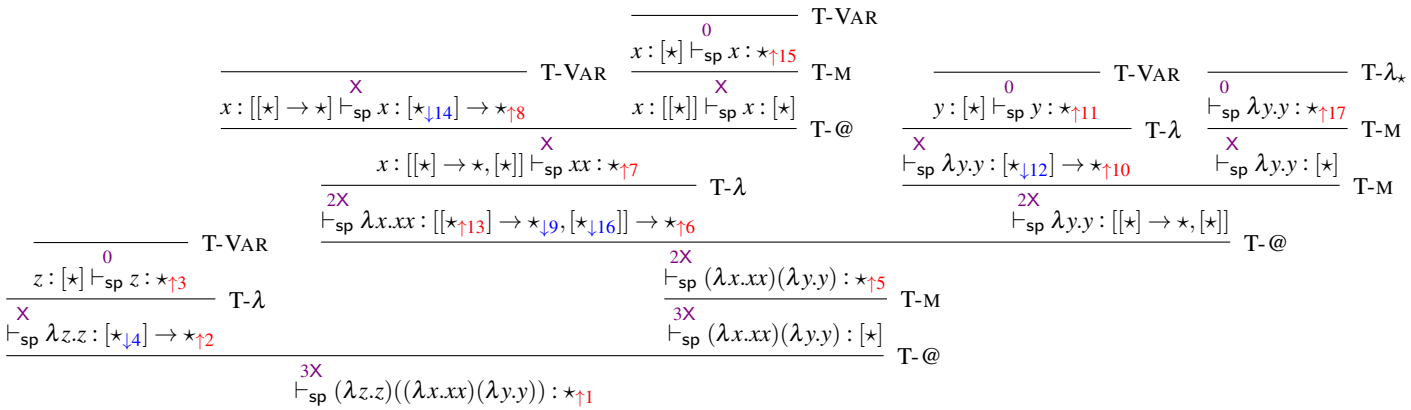

Figure 4: An example of a type derivation in the system $\vdash_{\mathsf{sp}}$.

that the main ingredient of the encoding of Turing machines into the λ-calculus, *i.e.* the fixed point combinator, makes the simulation unreasonable.

We give a type schema for the Turing's fixed point combinator Θ, defined as follows:

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

In doing so, we can safely assume, when typing Θ, that the argument we plan to pass to it, will use its argument linearly, and let us attribute Θ a type with this simplifying assumption in mind—this is actually the case needed for the encoding of Turing machines.

Consider a list of types $\overline{A} := A_k,..,A_0$, type A_i being the type one would like to attribute to Θt after i unfolding steps inside the recursion. We can first of all type Θ in the following way, when the recursion is never unfolded:

$$\Theta : \mathbb{F}_0^{\overline{A}} := \mathbb{T}_0^{\overline{A}} \to A_0 \quad \text{where} \quad \mathbb{T}_0^{\overline{A}} := [\mathbb{Y}_0^{\overline{A}}] \quad \text{and} \quad \mathbb{Y}_0^{\overline{A}} := [\cdot] \to A_0$$

But this is not the end of the story. What if recursion is unfolded more than once? These type schemes can be inductively defined to accommodate the general case:

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

Then, using the technology we have developed in the last two sections, we are able to derive some lower bounds to the time and space consumption of the λIAM when used to evaluate Θ.

Theorem 5.1. *For each $n \geq 0$, and for each list of types $\overline{A}$ such that $|\overline{A}| \geq n+1$,*

- $\vdash_{\mathsf{ti}}^{\geq 2^n} \Theta : \mathbb{F}_n^{\overline{A}}$;
- $\vdash_{\mathsf{sp}}^{\geq n\mathsf{X}} \Theta : \mathbb{F}_n^{\overline{A}}$.

Since Θ is used to copy the transition function of the encoded Turing machine M, the subscript n in $\mathbb{F}$ can be thought of as the number of transitions M needs to halt. This way, we have that the

λIAM consumes time exponential in n and space linear in n, whereas the invariance thesis prescribes an overhead which is polynomial in n for time and a linear function in the *space* (and not time) consumption of M[5].

References

[1] Beniamino Accattoli (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*, *Electronic Notes in Theoretical Computer Science* 338, Elsevier, pp. 23–43, doi:10.1016/j.entcs.2018.10.003.

[2] Beniamino Accattoli & Bruno Barras (2017): *Environments and the complexity of abstract machines*. In Wim Vanhoof & Brigitte Pientka, editors: *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming, Namur, Belgium, October 09 - 11, 2017*, ACM, pp. 4–16, doi:10.1145/3131851.3131855.

[3] Beniamino Accattoli, Ugo Dal Lago & Gabriele Vanoni: *The Space of Interaction*. To appear in the Proceedings of LICS 2021.

[4] Beniamino Accattoli, Ugo Dal Lago & Gabriele Vanoni (2020): *The Machinery of Interaction*. In: *PPDP '20: 22nd International Symposium on Principles and Practice of Declarative Programming, Bologna, Italy, 9-10 September, 2020*, ACM, pp. 4:1–4:15, doi:10.1145/3414080.3414108.

[5] Beniamino Accattoli, Ugo Dal Lago & Gabriele Vanoni (2021): *The (In)Efficiency of interaction*. *Proc. ACM Program. Lang.* 5(POPL), pp. 1–33, doi:10.1145/3434332.

[6] Hendrik Pieter Barendregt (1984): *The lambda calculus: its syntax and semantics*. North-Holland.

[7] Antonio Bucciarelli, Delia Kesner & Daniel Ventura (2017): *Non-idempotent intersection types for the Lambda-Calculus*. *Logic Journal of the IGPL* 25(4), pp. 431–464, doi:10.1093/jigpal/jzx018.

[8] Daniel de Carvalho (2018): *Execution time of λ-terms via denotational semantics and intersection types*. *Math. Str. in Comput. Sci.* 28(7), pp. 1169–1203, doi:10.1017/S0960129516000396.

[9] Mario Coppo & Mariangiola Dezani-Ciancaglini (1978): *A new type assignment for λ-terms*. *Arch. Math. Log.* 19(1), pp. 139–156, doi:10.1007/BF02011875.

[10] Ugo Dal Lago & Ulrich Schöpp (2016): *Computation by interaction for space-bounded functional programming*. *Information and Computation* 248, pp. 150–194, doi:10.1016/j.ic.2015.04.006.

[11] Vincent Danos & Laurent Regnier (1999): *Reversible, irreversible and optimal lambda-machines*. *Theoretical Computer Science* 227(1), pp. 79–97, doi:10.1016/S0304-3975(99)00049-3.

[12] Yannick Forster, Fabian Kunze & Marc Roth (2020): *The weak call-by-value λ-calculus is reasonable for both time and space*. *Proc. ACM Program. Lang.* 4(POPL), pp. 27:1–27:23, doi:10.1145/3371095.

[13] Philippa Gardner (1994): *Discovering Needed Reductions Using Type Theory*. In: *Theoretical Aspects of Computer Software (TACS '94)*, *Lecture Notes in Computer Science* 789, pp. 555–574, doi:10.1007/3-540-57887-0_115.

[14] Dan R. Ghica (2007): *Geometry of Synthesis: A Structured Approach to VLSI Design*. In Martin Hofmann & Matthias Felleisen, editors: *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, ACM, pp. 363–375, doi:10.1145/1190216.1190269.

[15] Jean-Yves Girard (1987): *Linear logic*. *Theoretical Computer Science* 50(1), pp. 1–101, doi:10.1016/0304-3975(87)90045-4.

[5] We underline that this negative result is not *absolute*, in that it rests on a specific, although very natural, encoding of Turing machines into the λ-calculus. The possibility that other encodings/fixed point combinators could allow a space efficient simulation of Turing machines is left open.

[16] Jean-Yves Girard (1989): *Geometry of Interaction 1: Interpretation of System F*. In R. Ferro, C. Bonotto, S. Valentini & A. Zanardo, editors: *Logic Colloquium '88*, *Studies in Logic and the Foundations of Mathematics* 127, Elsevier, pp. 221 – 260, doi:https://doi.org/10.1016/S0049-237X(08)70271-4.

[17] Jean-Yves Girard (1989): *Geometry of Interaction 1: Interpretation of System F*. In R. Ferro, C. Bonotto, S. Valentini & A. Zanardo, editors: *Studies in Logic and the Foundations of Mathematics*, 127, Elsevier, pp. 221–260.

[18] Jean-Louis Krivine (2007): *A Call-by-name Lambda-calculus Machine*. *Higher Order Symbol. Comput.* 20(3), pp. 199–207, doi:10.1007/s10990-007-9018-9.

[19] Ugo Dal Lago & Beniamino Accattoli (2017): *Encoding Turing Machines into the Deterministic Lambda-Calculus*. Available at `http://arxiv.org/abs/1711.10078`.

[20] Ian Mackie (1995): *The Geometry of Interaction Machine*. In Ron K. Cytron & Peter Lee, editors: *Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23-25, 1995*, ACM Press, pp. 198–208, doi:10.1145/199448.199483.

[21] Damiano Mazza (2015): *Simple Parsimonious Types and Logarithmic Space*. In Stephan Kreutzer, editor: *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, *LIPIcs* 41, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 24–40, doi:10.4230/LIPIcs.CSL.2015.24.

[22] Damiano Mazza, Luc Pellissier & Pierre Vial (2018): *Polyadic approximations, fibrations and intersection types*. *Proc. ACM Program. Lang.* 2(POPL), pp. 6:1–6:28, doi:10.1145/3158094.

[23] Ulrich Schopp (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.

[24] Cees F. Slot & Peter van Emde Boas (1988): *The Problem of Space Invariance for Sequential Machines*. *Inf. Comput.* 77(2), pp. 93–122, doi:10.1016/0890-5401(88)90052-1. Available at `https://doi.org/10.1016/0890-5401(88)90052-1`.

[25] Takeshi Tsukada, Kazuyuki Asada & 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.