



HAL
open science

The Space of Interaction

Beniamino Accattoli, Ugo Dal Lago, Gabriele Vanoni

► **To cite this version:**

Beniamino Accattoli, Ugo Dal Lago, Gabriele Vanoni. The Space of Interaction. LICS 2021 - 36th Annual ACM/IEEE Symposium on Logic in Computer Science, Jun 2021, Rome, France. pp.1-13, 10.1109/LICS52264.2021.9470726 . hal-03346767

HAL Id: hal-03346767

<https://inria.hal.science/hal-03346767>

Submitted on 19 Sep 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

The Space of Interaction

Beniamino Accattoli
Inria & École Polytechnique

Ugo Dal Lago
Università di Bologna & Inria

Gabriele Vanoni
Università di Bologna & Inria

Abstract—The space complexity of functional programs is not well understood. In particular, traditional implementation techniques are tailored to time efficiency, and space efficiency induces time inefficiencies, as it prefers re-computing to saving. Girard’s geometry of interaction underlies an alternative approach based on the interaction abstract machine (IAM), claimed as space efficient in the literature. It has also been conjectured to provide a reasonable notion of space for the λ -calculus, but such an important result seems to be elusive.

In this paper we introduce a new intersection type system precisely measuring the space consumption of the IAM on the typed term. Intersection types have been repeatedly used to measure time, which they achieve by dropping idempotency, turning intersections into multisets. Here we show that the space consumption of the IAM is connected to a further structural modification, turning multisets into trees. Tree intersection types lead to a finer understanding of some space complexity results from the literature. They also shed new light on the conjecture about reasonable space: we show that the usual way of encoding Turing machines into the λ -calculus cannot be used to prove that the space of the IAM is a reasonable cost model.

I. INTRODUCTION

Type systems are a form of compositional abstraction in which the behavior of programs, particularly higher-order programs, is described by *types*, that is, specifications of the kinds of objects programs expect in input and are supposed to produce as output. Typed programs usually *cannot go wrong*, as types guarantee the absence of run-time errors [13]. Some type systems ensure also other properties such as termination or (various forms of) security.

Intersection Types and Time Complexity: Among the many existing type systems one can define on top of the pure, untyped, λ -calculus [11], [29], *intersection types* have the peculiar feature of *characterizing* a property rather than merely *guaranteeing* it. This, in particular, is known to hold for various forms of termination properties since the pioneering work by Coppo and Dezani in 1978 [14]. More recently, a variant of intersection types has been proved to reflect *quantitative* properties of terms, such as the number of β -steps to normal form, or the number of steps of the Krivine abstract machine [30] (shortened to KAM), as discovered by de Carvalho [21]–[23]. The variant requires dropping 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 [7]–[10], [12], [16], [28]. In particular, research on the topic received

renewed attention after some recent progress in the study of reasonable cost models for the λ -calculus by Accattoli and Dal Lago [4] made evident that counting β -steps gives rise to a reasonable cost model for time.

Sharing, Time, and Space: The mainstream way of implementing the λ -calculus, at work also in the KAM, consists of mimicking β -steps while replacing meta-level substitutions with a finer approach based on environments. Environments are a way of realizing a form of *subterm sharing*, which is known to be mandatory in order to implement β -reduction in a time-efficient way. Traditional environment-based machines do not seem to be the right tool for space-efficiency. The reason is that these machines create a sharing annotation—counting as a unit of space—for *every* β -step—that is, for every abstract unit of time. They never garbage collect, because garbage collection is postponable, and ignoring it is safe for time, as its cost is negligible (it is polynomial, if not linear, in the number of β -steps). As a consequence, their use of space is linear in their time usage, which is the worst possible use of space. To study space efficiency, then, there are two possible approaches: either refining environment machines with an explicit treatment of garbage collection, or exploring alternative execution schemas. In this work, we follow the second approach.

Evaluating Without Sharing: Beyond the mainstream approach, there is an alternative execution schema for λ -terms, rooted in Girard’s Geometry of Interaction [27] and Abramsky, Jagadeesan, and Malacaria game semantics [1], which does *not* rely on sharing. The *interaction abstract machine* (shortened to IAM), first studied by Mackie [31] and Danos & Regnier [19], [20], evaluates a λ -term without tracing every β -step, thus disentangling time and space. As it is the case for space-efficient Turing machines, the IAM sometimes repeats computations to retrieve unsaved intermediate results—thus trading time for space. Environments are replaced by a *token*, a data structure where the machine saves minimal information, and the repetition of computations is realized via a sophisticated backtracking mechanism (unrelated to backtracking as in control operators or classical logic). The minimal information in the token amounts to tracing the points along the execution history where the IAM may need to backtrack. The entries of the token are *trees* of pointers called *logged positions* in a recent presentation of the IAM by Accattoli, Dal Lago, and Vanoni [5] (called instead *exponential signatures* by Danos & Regnier [20]). Everything else is ignored, in particular the token does not record encountered β -redexes.

Further evidence of the relevance for space of the interaction paradigm, comes from Ghica’s *Geometry of Synthesis* [25], [26],

in which the geometry of interaction is seen as a compilation scheme towards circuits, whose computation space is *finite*, and of paramount importance.

The Subtle Complexity of the IAM: These considerations suggest that the IAM is, roughly, bad for time and good for space. Interestingly, the situation is subtler. About time, there are indeed examples showing that the IAM is sometimes exponentially slower than environment machines. The slow-down however is not uniform, as in many cases the IAM exhibits good time performances.

The situation about space is not clear, either. The IAM has been used in the literature for obtaining sub-linear space bounds for functional programs (Dal Lago and Schöpp [18], [34], Mazza [32] and Ghica [25]), something hardly achievable with traditional environment machines. Having a close look at these results, however, one realizes that those bounds rely crucially on some tweaks (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 that the IAM space usage is a reasonable space cost model (that is, linearly related to the one of Turing machines) has been circulating among specialists for years, but has never found an answer.

Multi Types and the Time of the IAM: Very recently, there have been advances in the understanding of the subtle time behavior of the IAM. Accattoli, Dal Lago, and Vanoni have shown in [6] how to extract exact time bounds for the IAM from multi types derivations. Interestingly, they use the same types as de Carvalho’s, despite the IAM and the KAM computing in very different ways. While the time of the KAM is given by the multiplicity of the multi sets in multi types, they show that the time of the IAM requires to take into account also the size of the involved types. The result also provides a high-level understanding for the time (in)efficiency of the IAM: the inefficiency is proportional to the size of types. Since higher-order types are bigger than first-order ones, the more a program uses higher-order types the more its execution with the IAM is slower than with, say, the KAM.

A. Contributions of the Paper

The aim of this paper is to provide advances in the understanding of the subtle *space* behavior of the IAM. Inspired by the mentioned recent results by Accattoli, Dal Lago, and Vanoni, we introduce a new variant of multi types from which we extract *exact space bounds* for the IAM. To our knowledge, it is the first use of intersection/multi types to measure space.

A key point is that multi types as used in [6]—as well as in the many recent papers to extract quantitative bounds on λ -terms cited above—cannot measure the space consumption of the IAM, as they do not have enough structure. Our work shows that at the type level, indeed, one needs to add a *tree* structure to multi sets, similarly to how measuring time requires switching from idempotent intersections to multisets.

Once the main result is proved, we show how to use it to understand the subtle space behavior of the IAM. Here,

the main insight is a negative perspective about the elusive conjecture that the space of the IAM is a reasonable cost model. We also show that the new type system enlightens the key ingredients of the sub-linear space bounds in the literature.

Tree Types: Usually, multi types are structured in two mutually recursive and disjoint layers, linear types and multisets of linear types. Our types also have two layers, but they are no longer disjoint, as multisets can also contain multisets, not only linear types. This way multisets can naturally be seen as *trees*, whose internal nodes are multisets and whose leaves are linear types. Such tree types are very natural, and probably of independent interest. While flattening the tree structure recovers an ordinary multi type, the converse operation cannot be done in a canonical way. This fact shows that tree types add something new, they do not simply express a structure already present in multi types.

Actually, the tree structure reflects information about the token. The intuition is that while the number of leaves of a tree type counts the different uses of a variable/argument—as for multi types—internal nodes on the path to a leaf instead count how many backtracking pointers are stored in the token by the time the IAM gets to that copy.

The Complexity Analysis: Our space bounds are obtained by building on the technique developed by Accattoli, Dal Lago, and Vanoni in [6], which is inherently different from the one by de Carvalho and used in other recent works. The technique in [6] amounts to first defining an auxiliary machine evaluating multi type derivations and showing it bisimilar to the IAM. In this way, we obtain a representation of the states of the IAM run on the type derivation. Then, bounds are extracted from a global analysis based on *weighted* type derivations. We proceed similarly, replacing multi types with tree types, and introducing a new system of weights for space, based on the depth of tree types as trees.

Once the subtle bisimulation is established, our space complexity analysis is extremely simple, and also naturally provides exact bounds. This provides evidence that our types system is not ad-hoc. On the contrary, we believe it unveils a fundamental enrichment of multi types, deserving further studies.

Background on Reasonable Space: In order to discuss the mentioned conjecture about reasonable space, let us provide some background. First of all, studying interesting space complexity classes such as L, requires being able to measure *sub-linear* space. There is a recent result in the literature about reasonable space for the λ -calculus, by Forster, Kunze, and Roth [24], but their cost model—namely, the size of the term—can only measure linear and super-linear space, and thus it is not a solution for the general problem.

Now, showing that a cost model is reasonable requires studying the relationship with another known-to-be reasonable model, typically Turing machines. While for time the delicate direction is the simulation of the λ -calculus into Turing machines, for space the subtle one is the simulation of Turing machines in the λ -calculus, as it is hard to use as little space as a Turing machine.

The iteration of the transition function of Turing machines is a form of tail recursion, which in the λ -calculus is encoded via fixed-point operators. Such operators are also used to represent minimization in the encoding of partial recursive functions. Our insight about the reasonable space conjecture stems from an analysis of fixed-points operators in our type system.

The Elusive Reasonable Space Conjecture: Our contribution here is the fact that the IAM performs poorly when evaluating fixed-point operators, namely using an amount of space which is always at least *linear* in the number of performed recursive calls. This is done by deriving the type (schema) of a specific fixed-point operator in our system, then showing that its use of space is proportional to its use of time.

The specific operator we type is the one used in the encoding of Turing machines in the λ -calculus used by Accattoli and Dal Lago in their study of reasonable time [3], [15], as well as by Forster, Kunze, and Roth in [24]. Seeing it as the natural way of encoding tail recursion, it follows that the IAM space performance is poor with tail recursion, and, in turn, with the natural way of encoding Turing machines.

Summing up, our insight is that a positive answer to the conjecture cannot be done using the standard encoding of Turing machines, which explains the elusiveness of the conjecture.

Trees Density: A way of abstracting away from the issue with fixed-points is to look at how information is organized in the tree structure of tree types, itself reflecting the structure of (logged positions in) the token. When the tree is dense (its height is roughly the log of its number of nodes), then the IAM execution is space-efficient, while when the tree is sparse (height close to the number of nodes) it is inefficient—the type schema for fixed-points is indeed sparse.

Perspectives: The insight about the density of trees allows to re-understand some results in the literature, and opens new perspectives. As we detail in Section VIII, it sheds light on Dal Lago and Schöpp’s space bounds for a functional language [18], [34], as well as on the encoding of sub-linear space computations in the λ -calculus by Mazza [32].

Proofs: Proofs are in the Appendix.

II. CLOSED CALL-BY-NAME AND ABSTRACT MACHINES

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

$$\lambda\text{-TERMS } t, u, r ::= x \in \mathcal{V} \mid \lambda x.t \mid tu.$$

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.

LEVELED CONTEXTS

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

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 \rightarrow_{wh} , defined as follows:

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

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

Abstract Machines Glossary: In this paper, an *abstract machine* $M = (s, \rightarrow)$ is a transition system \rightarrow over a set of states, noted s . 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 $C\langle u \rangle = t$. States are composed by a position (u, C) plus some data structures. A state is *initial*, and noted s_t , if it is positioned on $(t, \langle \cdot \rangle)$, t is closed, and all the data structures are empty— t is always implicitly considered closed, without further mention. A state is *final* if no transitions apply.

A *run* $\rho : s \rightarrow^* s'$ is a possibly empty sequence of transitions, whose length is noted $|\rho|$. An *initial run* is a run from an initial state s_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.

III. THE INTERACTION ABSTRACT MACHINE, REVISITED

In this section we provide an overview of the Interaction Abstract Machine (IAM). We adopt the λ -calculus presentation of the IAM, rather called λ IAM and recently developed by Accattoli, Dal Lago, and Vanoni in [5]—we refer to their work for an in-depth study of the λ IAM. 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. Keep in mind that the λ IAM is an unusual machine, and that finding it hard to grasp is normal. Also, in [5] there is an alternative explanation of the λ IAM, that may be helpful, together with the relationship with proof nets, which is however not needed here.

Bird’s Eye view of the λ IAM: Intuitively, the behaviour of the λ IAM 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 λ IAM is that it does not store the encountered β -redexes in an environment. When it finds the head variable, the λ IAM 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

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

Sub-term	Context	Log	Tape		Sub-term	Context	Log	Tape
<u>tu</u>	C	L	T	$\rightarrow_{\bullet 1}$	<u>t</u>	$C\langle\langle \cdot \rangle u\rangle\rangle$	L	$\bullet \cdot T$
<u>$\lambda x.t$</u>	C	L	$\bullet \cdot T$	$\rightarrow_{\bullet 2}$	<u>t</u>	$C\langle\lambda x.\langle \cdot \rangle\rangle$	L	T
<u>x</u>	$C\langle\lambda x.D_n\rangle$	$L_n \cdot L$	T	\rightarrow_{var}	$\lambda x.D_n\langle x \rangle$	<u>C</u>	L	$(x, \lambda x.D_n, L_n) \cdot T$
<u>$\lambda x.D_n\langle x \rangle$</u>	C	L	$(x, \lambda x.D_n, L_n) \cdot T$	$\rightarrow_{\text{bt}2}$	<u>x</u>	<u>$C\langle\lambda x.D_n\rangle$</u>	$L_n \cdot L$	T
<u>t</u>	<u>$C\langle\langle \cdot \rangle u\rangle\rangle$</u>	L	$\bullet \cdot T$	$\rightarrow_{\bullet 3}$	<u>tu</u>	<u>C</u>	L	T
<u>t</u>	<u>$C\langle\lambda x.\langle \cdot \rangle\rangle$</u>	L	T	$\rightarrow_{\bullet 4}$	$\lambda x.t$	<u>C</u>	L	$\bullet \cdot T$
<u>t</u>	<u>$C\langle\langle \cdot \rangle u\rangle\rangle$</u>	L	$l \cdot T$	\rightarrow_{arg}	<u>u</u>	$C\langle t \langle \cdot \rangle \rangle$	$l \cdot L$	T
<u>t</u>	<u>$C\langle u \langle \cdot \rangle \rangle$</u>	$l \cdot L$	T	$\rightarrow_{\text{bt}1}$	<u>u</u>	$C\langle \langle \cdot \rangle t \rangle$	L	$l \cdot T$

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

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.

λ IAM States: The data structures of the λ IAM are in Fig. 1. The λ IAM 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 $s_t := (\underline{t}, \langle \cdot \rangle, \epsilon, \epsilon)$. 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 λ IAM are in Fig. 1. Their union is noted $\rightarrow_{\lambda\text{IAM}}$. 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 positions¹. Intuitively, the machine evaluates the term t until the head abstraction of the head normal form is found (more explanations below). The transitions realize three entangled mechanisms.

Mechanism 1: Search Up to β -Redexes: Note that $\rightarrow_{\bullet 1}$ 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 $\rightarrow_{\bullet 2}$ 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 λ IAM 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 $\rightarrow_{\bullet 3}$ and $\rightarrow_{\bullet 4}$ do the same during the \uparrow phase.

Let us illustrate this mechanism with an example: the first steps of evaluation on the term $l((\lambda x.xx)l)$, where l is the identity combinator. One can notice that the λ IAM 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
<u>$(\lambda z.z)((\lambda x.xx)l)$</u>	$\langle \cdot \rangle$	ϵ	ϵ	\downarrow
$\rightarrow_{\bullet 1}$ <u>$\lambda z.z$</u>	$\langle \cdot \rangle((\lambda x.xx)l)$	ϵ	\bullet	\downarrow
$\rightarrow_{\bullet 1}$ <u>z</u>	$(\lambda z.\langle \cdot \rangle)((\lambda x.xx)l)$	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 \rightarrow_{var} switches direction from \downarrow to \uparrow , and the machine starts looking for potential substitutions for x . The λ IAM 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

¹Transitions \rightarrow_{var} and $\rightarrow_{\text{bt}2}$ might not look 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.

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 \rightarrow_{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 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 $l_z := (z, (\lambda z.\langle \cdot \rangle)((\lambda x.xx)l), \epsilon)$, $l_{\langle \cdot \rangle x} := (x, \lambda x.\langle \cdot \rangle x, \epsilon)$ and $l_y := (y, \lambda y.\langle \cdot \rangle, \epsilon)$.

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

Mechanism 3: Backtracking: It is started by transition \rightarrow_{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 \rightarrow_{bt2} is fired.

The \downarrow -phase and the logged position on the tape mean that the λ IAM 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 λ IAM 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
	$\lambda y.y$	$l((\lambda x.xx)\langle \cdot \rangle)$	$l_{\langle \cdot \rangle x} \cdot l_z$	l_y	\uparrow
\rightarrow_{bt1}	$\lambda x.xx$	$l(\langle \cdot \rangle)l$	l_z	$l_{\langle \cdot \rangle x} \cdot l_y$	\downarrow
\rightarrow_{bt2}	x	$l((\lambda x.\langle \cdot \rangle)x)l$	l_z	l_y	\uparrow
\rightarrow_{arg}	x	$l((\lambda x.x)\langle \cdot \rangle)l$	$l_y \cdot l_z$	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 l , is found. We set $l_{x\langle \cdot \rangle} := (x, \lambda x.x\langle \cdot \rangle, l_y)$.

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

Basic invariants: 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 and the current position (t, C) form a logged position, *i.e.* the length of L is exactly the level of the code context C^2 . This fact guarantees that the λ IAM never gets stuck because the log is too short for transitions \rightarrow_{var} and \rightarrow_{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 *tape and direction* 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 III.1 (λ IAM basic invariants). *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: $d = \downarrow^{|T|_l}$.

Final States: If the λ IAM starts on the initial state s_t , the execution may either never stop or end in a state s of the shape $s = (\lambda x.u, C, L, \epsilon)$. The fact that no other shapes are possible for s is proved in [5].

Implementation: Usually, the λ IAM 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 λ IAM induces a semantics $\llbracket \cdot \rrbracket_{\lambda\text{IAM}}$ 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 [5]. For the sake of simplicity, here we restrict to Closed CbN. The λ IAM semantics then reduces to observing termination: $\llbracket t \rrbracket_{\lambda\text{IAM}}$ 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 λ IAM implements Closed CbN when its execution from the initial state s_t reaches a final state if and only if \rightarrow_{wh} terminates on t , for every closed term t .

²Then, the length of L is exactly the number of (linear logic) *boxes* in which the code term is contained.

$$\begin{array}{c}
\frac{}{x : [A] \vdash x : A} \text{T-VAR} \qquad \frac{\Gamma, x : T \vdash t : A}{\Gamma \vdash \lambda x.t : T \rightarrow A} \text{T-}\lambda \qquad \frac{\Gamma \vdash t : T \rightarrow A \quad \Delta \vdash u : T}{\Gamma \uplus \Delta \vdash tu : A} \text{T-}\@ \\
\frac{}{\Gamma \vdash \lambda x.t : \star} \text{T-}\lambda_{\star} \qquad \frac{\Gamma_i \vdash t : G_i \quad 1 \leq i \leq n}{[\uplus_{i=1}^n \Gamma_i] \vdash t : [G_1, \dots, G_n]} \text{T-MANY} \qquad \frac{}{\vdash t : [\cdot]} \text{T-NONE}
\end{array}$$

Figure 2: The tree type system.

Theorem III.2 ([5]). *The λ IAM implements Closed CbN.*

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

$$\begin{aligned}
|(t, C, L, T, d)|_{\text{sp}} &:= |L|_{\text{sp}} + |T|_{\text{sp}} \\
|(x, D, L')|_{\text{sp}} &:= X + |L'|_{\text{sp}} \quad |\epsilon|_{\text{sp}} := 0 \\
|l \cdot \Gamma|_{\text{sp}} &:= |l|_{\text{sp}} + |\Gamma|_{\text{sp}} \quad |\bullet \cdot T|_{\text{sp}} := 1 + |T|_{\text{sp}}
\end{aligned}$$

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

Definition III.3. *Let $\rho : s_0 \rightarrow_{\lambda\text{IAM}}^* s$ be a λ IAM run. Then,*

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

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 III.4. *Let ρ be an infinite λ IAM run.*

Then $|\rho|_{\text{sp}} = \infty$.

IV. TREE (INTERSECTION) TYPES

Here we introduce a type system that we shall use to measure the space used by λ IAM 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) *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 shall only need bijections between multisets, to describe the reformulation of the λ IAM on type derivations, and these bijections are just more easily managed if commutativity is removed. This *rigid* approach is also used by Tsukada, Asada, and Ong [35] and Mazza, Pellissier, and Vial [33]. 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*.

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

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 $T \rightarrow A$ 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 $T \uplus T'$. We use $|T|$ for the length of T as a sequence, that is, $|[G_1, \dots, G_n]| := n$.

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 $\Gamma = x_1 : T_1, \dots, x_n : T_n$ if $\text{dom}(\Gamma) = \{x_1, \dots, x_n\}$ —note that type environments *are* commutative, what is non-commutative is the sequence constructor $[\cdot]$, only. Given two type environments Γ, Δ , we use $\Gamma \uplus \Delta$ for the type environment assigning to every variable x the list $\Gamma(x) \uplus \Delta(x)$.

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

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.

$$\text{LEAF CTXS} \quad \mathbb{L} ::= [G_1, \dots, \langle \cdot \rangle, \dots, G_n] \mid [G_1, \dots, \mathbb{L}, \dots, G_n]$$

If $T^\ell = [A_1, \dots, A_n]$ 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, 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 IV.1 (Relevance and axiom sequences). *If $\pi \triangleright \Gamma \vdash t : A$ then $\text{dom}(\Gamma) \subseteq \text{fv}(t)$, 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 $\Gamma(x)^\ell = [A_1, \dots, A_k]$ 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 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 recipe³ for multi types, without surprises. See the Appendix for details.

Theorem IV.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

³Namely, 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- λ_\star).

fact is evidence that tree types are strictly richer than multi types, because the tree structure cannot be inferred from the multiset one.

V. THE TREE IAM

This section introduces a machine evaluating type derivations, the *Tree IAM*, or *TIAM*, that mimics the λ IAM directly on top of a type derivation π . It is the key tool that we shall use to measure the space cost of λ IAM runs. The TIAM is a very minor variation over the similar SIAM machine evaluating type derivations for sequence types by Accattoli, Dal Lago, and Vanoni in [6]. This and the next section mostly just recall and adapt notions and results from that paper.

Preamble about Duplications and (No) Logs: β -reducing a λ -term (potentially) duplicates arguments, whose different copies may be used differently, typically being applied to different further arguments. The λ IAM never duplicate arguments, but has nonetheless to distinguish different uses of the same piece of code. This is why it 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 key point of multi/tree type derivations is that duplication is explicitly accounted for, *in advance*, by multisets/trees: all arguments come with as many type derivations as the times they are duplicated during evaluation. With tree types, the number of copies is the number of leaves of the tree. Note that a multi/tree type derivation may be way bigger than the term itself, while this is not possible with, say, simple types.

The intuition behind the TIAM is that the walk over the code done by the λ IAM can be rephrased and simplified on multi/tree type derivations, because the mechanism of logs—needed to distinguish different copies of arguments—is no longer needed, since all copies are already there: simple positions in the type derivation (not in the term!) are informative enough.

The TIAM: On the one hand, the TIAM is simpler than the λ IAM because it has no logs, on the other hand, it is more technical to define because tree type derivations are less easily manipulated than λ -terms. The underlying idea however is simple. 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. 3. We shall now explain every involved concept.

The position of the machine is given by an occurrence of a type judgment⁴ J of π . As the λ IAM, the TIAM has two possible directions, noted \downarrow and \uparrow ⁵. 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 the tape of the λ IAM, 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):

⁴A 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.

⁵Type derivations are upside-down wrt to the term structure, then direction \downarrow of the λ IAM becomes here \uparrow , and \uparrow is \downarrow .

$\frac{\frac{\vdash t : T \rightarrow A \quad \vdash}{\vdash tu : \mathbb{A}\langle\star\uparrow\rangle(=A)}}{\rightarrow_{\bullet 1}} \quad \frac{\frac{\vdash t : T \rightarrow \mathbb{A}\langle\star\uparrow\rangle \quad \vdash}{\vdash tu : A}}{\rightarrow_{\bullet 2}} \quad \frac{\frac{\vdash t : A(=\mathbb{A}\langle\star\rangle)}{\vdash \lambda x.t : T \rightarrow \mathbb{A}\langle\star\uparrow\rangle}}{\rightarrow_{\bullet 2}} \quad \frac{\frac{\vdash t : \mathbb{A}\langle\star\uparrow\rangle}{\vdash \lambda x.t : T \rightarrow A}}{\rightarrow_{\bullet 2}}$
$\frac{\frac{\frac{\vdash t : T \rightarrow \mathbb{A}\langle\star\downarrow\rangle \quad \vdash}{\vdash tu : A(=\mathbb{A}\langle\star\rangle)}}{\rightarrow_{\bullet 3}} \quad \frac{\frac{\frac{\vdash t : T \rightarrow A \quad \vdash}{\vdash tu : \mathbb{A}\langle\star\downarrow\rangle}}{\rightarrow_{\bullet 3}}}{\rightarrow_{\bullet 3}} \quad \frac{\frac{\frac{\vdash t : \mathbb{A}\langle\star\downarrow\rangle(=A)}{\vdash \lambda x.t : T \rightarrow A}}{\rightarrow_{\bullet 4}} \quad \frac{\frac{\vdash t : A}{\vdash \lambda x.t : T \rightarrow \mathbb{A}\langle\star\downarrow\rangle}}{\rightarrow_{\bullet 4}}}{\rightarrow_{\bullet 4}}$
$\frac{\frac{\frac{\frac{\vdash x : \mathbb{A}\langle\star\uparrow\rangle_i(=A_i) \quad \vdash}{\vdots} \quad \vdash}{\vdash \lambda x.C(x) : \mathbb{L}\langle A_i \rangle \rightarrow A'}}{\rightarrow_{\text{var}}} \quad \frac{\frac{\frac{\frac{\vdash x : A_i \quad \vdash}{\vdots} \quad \vdash}{\vdash \lambda x.C(x) : \mathbb{L}\langle \mathbb{A}\langle\star\downarrow\rangle_i \rangle \rightarrow A'}}{\rightarrow_{\text{var}}}}{\rightarrow_{\text{var}}}} \quad \frac{\frac{\frac{\frac{\frac{\vdash x : A_i(=\mathbb{A}\langle\star\rangle_i) \quad \vdash}{\vdots} \quad \vdash}{\vdash \lambda x.C(x) : \mathbb{L}\langle \mathbb{A}\langle\star\uparrow\rangle_i \rangle \rightarrow A'}}{\rightarrow_{\text{bt2}}} \quad \frac{\frac{\frac{\frac{\vdash x : \mathbb{A}\langle\star\downarrow\rangle_i \quad \vdash}{\vdots} \quad \vdash}{\vdash \lambda x.C(x) : \mathbb{L}\langle A_i \rangle \rightarrow A'}}{\rightarrow_{\text{bt2}}}}{\rightarrow_{\text{bt2}}}}{\rightarrow_{\text{bt2}}}}$
$\frac{\frac{\frac{\frac{\vdash t : \mathbb{L}\langle \mathbb{A}\langle\star\downarrow\rangle_i \rangle \rightarrow A \quad \frac{\dots \vdash u : \mathbb{A}\langle\star\rangle_i \dots}{\vdash u : T(=\mathbb{L}\langle \mathbb{A}\langle\star\rangle_i \rangle)} \text{T-MANY}}{\vdash tu : A}}{\rightarrow_{\text{arg}}} \quad \frac{\frac{\frac{\frac{\vdash t : T \rightarrow A \quad \frac{\dots \vdash u : \mathbb{A}\langle\star\uparrow\rangle_i \dots}{\vdash u : \mathbb{L}\langle \mathbb{A}\langle\star\rangle_i \rangle} \text{T-MANY}}{\vdash tu : A}}{\rightarrow_{\text{arg}}}}{\rightarrow_{\text{arg}}}}$
$\frac{\frac{\frac{\frac{\vdash t : T \rightarrow A \quad \frac{\dots \vdash u : \mathbb{A}\langle\star\downarrow\rangle_i \dots}{\vdash u : \mathbb{L}\langle \mathbb{A}\langle\star\rangle_i \rangle(=T)} \text{T-MANY}}{\vdash tu : A}}{\rightarrow_{\text{bt1}}} \quad \frac{\frac{\frac{\frac{\vdash t : \mathbb{L}\langle \mathbb{A}\langle\star\uparrow\rangle_i \rangle \rightarrow A \quad \frac{\dots \vdash u : \mathbb{A}\langle\star\rangle_i \dots}{\vdash u : T} \text{T-MANY}}{\vdash tu : A}}{\rightarrow_{\text{bt1}}}}{\rightarrow_{\text{bt1}}}}{\rightarrow_{\text{bt1}}}}$

Figure 3: The transitions of the Tree IAM (TIAM).

LINEAR CTXS $\mathbb{A} ::= \langle \cdot \rangle \mid T \rightarrow \mathbb{A} \mid \mathbb{T} \rightarrow A$
 TREE CTXS $\mathbb{T} ::= [G_1, \dots, \mathbb{G}, \dots, G_n]$
 TYPE CTXS $\mathbb{G} ::= \mathbb{A} \mid \mathbb{T}$

Summing up, a state s is a quadruple (π, 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 λ IAM 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 λ IAM 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.

The transitions are in Fig. 3, their union is noted $\rightarrow_{\text{TIAM}}$. We now explain them one by one—the transitions have the labels of λ IAM transitions, because they correspond to each other, as we shall show.

Let's start with the simplest, $\rightarrow_{\bullet 2}$. 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 $T \rightarrow A$ to $\lambda x.t$, and the type context is $T \rightarrow \mathbb{A}$, 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: the extra premise \vdash simply denotes the right premise of the T-@ rule that is left unspecified since not relevant to the transition.

Transitions \rightarrow_{arg} : the focus is on the left premise of a T-@ rule, of type $T \rightarrow A'$ isolating \star inside the i -th leaf A_i of the tree type T . The transition then moves the state of the TIAM to the i -th leaf of the tree of T-MANY rules on the right premise, changing direction. Transition \rightarrow_{bt1} does the opposite move.

Transitions \rightarrow_{var} and \rightarrow_{bt2} are based on the axiom sequences property of Lemma IV.1. Consider a T- λ rule occurrence whose right-hand type of the conclusion is $T \rightarrow A'$. The premise has shape $\Gamma, x : T \vdash t : A'$, and by the lemma there is a bijection between the leaves in T and the axioms on x , respecting the order in T^ℓ . The left side of \rightarrow_{bt2} focuses on the i -th leaf A_i of T and the TIAM 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_{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- λ_\star . Accordingly, when the TIAM reaches a T- λ_\star rule, it is in a final state. Exactly as the λ IAM, the TIAM is bi-deterministic.

Proposition V.1. *The TIAM is bi-deterministic for each type derivation $\pi \triangleright \vdash t : \star$.*

$$\begin{array}{c}
\frac{}{z : [\star] \vdash^0 z : \star \uparrow 3} \text{T-VAR} \\
\frac{}{\vdash^X \lambda z.z : [\star \downarrow 4] \rightarrow \star \uparrow 2} \text{T-}\lambda \\
\frac{}{x : [[\star] \rightarrow \star] \vdash^X x : [\star \downarrow 14] \rightarrow \star \uparrow 8} \text{T-VAR} \\
\frac{}{x : [[\star] \rightarrow \star, [\star]] \vdash^X xx : \star \uparrow 7} \text{T-}\lambda \\
\frac{}{\vdash^{2X} \lambda x.xx : [[\star \uparrow 13] \rightarrow \star \downarrow 9, [\star \downarrow 16]] \rightarrow \star \uparrow 6} \text{T-}\lambda \\
\frac{}{x : [\star] \vdash^0 x : \star \uparrow 15} \text{T-VAR} \\
\frac{}{x : [[\star]] \vdash^X x : [\star]} \text{T-M} \\
\frac{}{y : [\star] \vdash^0 y : \star \uparrow 11} \text{T-VAR} \\
\frac{}{\vdash^X \lambda y.y : [\star \downarrow 12] \rightarrow \star \uparrow 10} \text{T-}\lambda \\
\frac{}{\vdash^{2X} \lambda y.y : [[\star] \rightarrow \star, [\star]]} \text{T-}\lambda \\
\frac{}{\vdash^{2X} (\lambda x.xx)(\lambda y.y) : \star \uparrow 5} \text{T-M} \\
\frac{}{\vdash^{3X} (\lambda x.xx)(\lambda y.y) : [\star]} \text{T-}\lambda \\
\frac{}{\vdash^{3X} (\lambda z.z)((\lambda x.xx)(\lambda y.y)) : \star \uparrow 1} \text{T-}\lambda
\end{array}$$

Figure 4: An example of TIAM execution.

An example: In Fig. 4 we present the very same example analyzed in Section III. 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. 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 III. Please note that judgments are decorated with weights (such as X), which shall be introduced only in Sect. VII, in order to later provide an example of decoration—they can be safely ignored for now.

VI. THE λ IAM AND TIAM BISIMULATION

The aim of this section is to explain the strong bisimulation between the TIAM and the λ IAM, that is essentially the same between the SIAM and the λ IAM studied in [6]. 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 λ IAM, the log $L = l_1 \dots l_n$ has a logged position for every argument u_1, \dots, 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, \dots, 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 λ IAM 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 λ IAM manipulates the log only via transitions \rightarrow_{arg} and \rightarrow_{bt1} , that on the TIAM correspond exactly to entering/exiting derivations for arguments. The tape, instead, contains logged positions for which the λ IAM either has not yet found the associated argument, or it is backtracking to. Note that the λ IAM puts logged positions on the tape via transitions \rightarrow_{var} and \rightarrow_{bt1} , and removes them using \rightarrow_{arg} and \rightarrow_{bt2} . By looking at Fig. 3, it is clear that there is a logged position on the λ IAM 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 λ IAM States: These ideas are used to extract from every TIAM state s a λ IAM state $\text{ext}(s)$ in a quite technical way. In particular, the extraction process retrieves a

log $L_{\text{ext}}(s)$ from the judgement J of s and a tape $T_{\text{ext}}(s)$ 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 $L_{\text{ext}}(s)$ and $T_{\text{ext}}(s)$.

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 $l_1 \dots l_n$ forming the extracted log $L_{\text{ext}}(s)$, following the correspondence described above.

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

$$\begin{array}{ll}
T_{\text{ext}}(\langle \cdot \rangle) & := \epsilon \\
T_{\text{ext}}(T \rightarrow \mathbb{A}) & := \bullet \cdot T_{\text{ext}}(\mathbb{A}) \\
T_{\text{ext}}(\mathbb{L} \langle \mathbb{A} \rangle \rightarrow A') & := l_{\text{ext}}(s_{\mathbb{L}}) \cdot T_{\text{ext}}(\mathbb{A})
\end{array}$$

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

For lack of space, and because this is essentially identical to what is done in [6] for the SIAM, the technical development is in Appendix VI. The extraction process induces a relation $s \simeq_{\text{ext}} \text{ext}(s)$ that is easily proved to be a strong bisimulation between the TIAM and the λ IAM.

Proposition VI.1 (TIAM and λ IAM bisimulation). *Let t a closed term and $\pi \triangleright \vdash t : \star$ a tree type derivation. Then \simeq_{ext} is a strong bisimulation between TIAM states on π and λ IAM states on t . Moreover, if $s_\pi \simeq_{\text{ext}} s_\lambda$ then s_π is TIAM reachable if and only if s_λ is λ IAM reachable.*

The *moreover* part of the above statement hints at a bijection between *all* the states in π and reachable λ IAM states. However, there still could be the possibility that some of the states in π are not reachable. This is actually *not* the case, and the technical development is in the Appendix.

Proposition VI.2. *Let t a closed term and $\pi \triangleright \vdash t : \star$ a tree type derivation. Then every state of π is reached exactly once.*

$$\begin{array}{c}
\frac{}{x : [A] \vdash x : A} \text{ T-VAR} \qquad \frac{\Gamma, x : T \vdash t : A}{\Gamma \vdash \lambda x.t : T \rightarrow A} \text{ T-}\lambda \qquad \frac{\Gamma \vdash t : T \rightarrow A \quad \Delta \vdash u : T}{\Gamma \uplus \Delta \vdash tu : A} \text{ T-@} \\
\\
\frac{}{\Gamma \vdash \lambda x.t : \star} \text{ T-}\lambda_{\star} \qquad \frac{\Gamma_i \vdash t : G_i \quad 1 \leq i \leq n}{[\uplus_{i=1}^n \Gamma_i] \vdash t : [G_1, \dots, G_n]} \text{ T-MANY} \qquad \frac{}{\Gamma \vdash t : [\cdot]} \text{ T-NONE}
\end{array}$$

Figure 5: The tree type system with weights.

VII. MEASURING THE SPACE OF INTERACTIONS

This section contains the main contribution of the article: it gives a way of measuring the space consumed by the complete λ IAM 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 λ IAM state $\text{ext}(s)$ 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 λ IAM 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 λ IAM, 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 λ IAM global code, that is, by $\log |t|$, where t 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 $\log |t|$.

A. 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 $\text{ext}(s)$ 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 VII.1 (Branch size of type contexts/extracted tapes). *Let $s = (\pi, J, \mathbb{A}, d)$ be a TIAM state. The branch size $|\cdot|_{\mathbb{b}}$ for type contexts is defined as follows:*

$$\begin{aligned}
|\langle \cdot \rangle|_{\mathbb{b}} &:= 0 & |T \rightarrow \mathbb{A}|_{\mathbb{b}} &:= 1 + |\mathbb{A}|_{\mathbb{b}} \\
|T \rightarrow A|_{\mathbb{b}} &:= |T|_{\mathbb{b}} & |[G_1, \dots, G_n]|_{\mathbb{b}} &:= X + |G|_{\mathbb{b}}
\end{aligned}$$

Let us interpret the branch size with respect to the tape extraction schema of the previous section. The $+1$ in the clause for $T \rightarrow \mathbb{A}$ is there to count \bullet . The clause for sequences instead gives $|\mathbb{L}| = n \cdot X$ 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 VII.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 $|J|_{\mathbb{b}} := n \cdot X$.*
- Branch size of TIAM states: $|s|_{\mathbb{b}} := |\mathbb{A}|_{\mathbb{b}} + |J|_{\mathbb{b}}$.

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 λ IAM state.

Proposition VII.3 (Space of Single Extracted States). *Let $s = (\pi, J, \mathbb{A}, d)$ be a reachable TIAM state. Then $|\mathbb{A}|_{\mathbb{b}} = |T_{\text{ext}}(s)|_{\text{sp}}$ and $|J|_{\mathbb{b}} = |L_{\text{ext}}(s)|_{\text{sp}}$, and thus $|s|_{\mathbb{b}} = |\text{ext}(s)|_{\text{sp}}$. Moreover,*

- 1) *if $L_{\text{ext}}(s) = l_1..l_n$, 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 $|l_i|_{\text{sp}} = h_i$;*
- 2) *for each extracted tape position l , i.e. for each \mathbb{G} such that $\mathbb{A} = \mathbb{G}(\mathbb{L}(\mathbb{A}'(\star)) \rightarrow A)$, then $|l|_{\text{sp}} = |\mathbb{L}| \cdot X$.*

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, λ IAM states are extracted from *sequence* type derivations in [6]. 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 $\text{ext}(s)$, 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. VII.3, but do not describe the internal structure of logged positions. Given a TIAM state s , we can easily know the length of $L_{\text{ext}}(s)$ and $T_{\text{ext}}(s)$, 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.

B. The Space of the Whole Execution

Type Weights: To obtain the space cost of the whole execution we endow tree types derivations with *weights*⁶. 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 $\mathbb{G}(\star)$, as confirmed by the associated lemma.

$$\begin{aligned} \|\star\| &:= 0 & \|T \rightarrow A\| &:= \max\{\|T\|, \|A\| + 1\} \\ \|[G_1, \dots, G_n]\| &:= X + \max_i\{\|G_i\|\} \end{aligned}$$

Lemma VII.4. *Let G be a type. Then $\|G\| = \max_{\mathbb{G}|G=\mathbb{G}(\star)} \|\mathbb{G}\|_b$.*

Note that, via the space of single extracted states (Prop. VII.3), the previous lemma states that the size of G is the maximum space of all the tapes extracted from TIAM states over a same judgment $\Gamma \vdash t : G$.

Judgements and Derivations Weights: Weights are extended to judgments in Fig. 5, and the weight of a derivation is the weight of its final judgment. The idea is that the weight w of a weighted judgment $J = \Gamma \vdash t : G$ 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 VII.5 (Judgment weights bound extracted tapes). *Let $\pi : \Gamma \vdash t : G$ be a weighted derivation and \mathcal{J} be the set of all the judgments occurring in π . Then $w \geq \max_{\Delta \vdash u : G' \in \mathcal{J}} \|\mathbb{G}'\|$.*

Judgement weights actually take also logs into account.

Lemma VII.6 (Weights bound also extracted logs). *Let $\pi \triangleright \Gamma \vdash t : G$ be a weighted derivation. Then $w \geq v + |J|_b$ for every weighted judgment $J \vdash^v$ in π .*

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 λ IAM execution of t .

Theorem VII.7 (λ IAM space bounds). *Let $\pi \triangleright \Gamma \vdash t : \star$ be a weighted tree types derivation. Then $|\text{ext}(s)|_{\text{sp}} \leq w$ for every $s \in \text{states}(\pi)$.*

⁶We 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.

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

Proposition VII.8 (Weight witness). *Let $\pi \triangleright \Gamma \vdash t : G$ be a weighted derivation and $G \neq [\cdot]$. Then there exists a TIAM state s over π such that $w = |s|_b$.*

We can then conclude our complexity analysis.

Corollary VII.9 (λ IAM exact bound via tree types derivations). *Let $\pi \triangleright \Gamma \vdash t : \star$ be a tree types derivation and ρ the complete λ IAM run on t . Then $|\rho|_s = w$.*

Now, the reader can fully understand and appreciate the weights in the derivation of Fig. 4. Please note that we have considered $\max\{1, X\} = X$ when assigning the weights.

VIII. ON THE λ IAM SPACE (IN)EFFICIENCY

Tree Density: As we have proved in the last section, the space consumed by a λ IAM run ultimately depends on the level of nesting of tree types. Let us be slightly more precise. When we are dealing with multi types, the argument u of an application tu has to be typed with a multiset, say $M := [\star, \dots, \star]$ where $|M| = n$ and n is the number of times that u shall be copied. In our type system, M can be represented as a tree T , capturing the space needed for evaluating the copies of u , in several ways. In the tree type derivation for tu only one of these representations is used, and depends on the type of t . The important point is that different representations lead to very different space consumptions. The ideal case is the flat representation $T_f := [\star, \dots, \star]$, for which $\|T_f\| = X$. Another good case is given by a full binary tree T_b , or, more generally, by a tree T_d whose height is logarithmic in n , so that $\|T_d\| = \log n \cdot X$. Let us deem this case *dense*. A bad case is given by linearly shaped trees such as $T_l := T_n$, where $T_0 := [\star]$ and $T_{n+1} := [\star, T_n]$, for which $\|T_s\| = (n+1) \cdot X \geq \|T_d\|$. We call *sparse* a tree with n leaves and height n like T_l . Not that there can even be worse, as in general a tree can have an arbitrary number of internal nodes, for instance a multi set $[\star]$ can be represented as a tree also as $[[[[\star]]]]$ —again, it depends on the type of t . Intuitively, if a term can be typed with flat or dense trees, the λ IAM evaluates it space *efficiently*, while in the case of sparse trees (or worse) the evaluation is space *inefficient*.

Fixed-Points Are Sparse: As is well known, the λ -calculus is a universal (or Turing complete) model of computation. The fundamental ingredient that allows one to achieve universality is the presence of fixed-point combinators. These combinators allow for an encoding of general recursion, as needed when simulating, e.g., partial recursive functions or Turing machines. In particular, the fixed point combinator can capture unbounded iteration (i.e. `while` loops) or tail-recursion, as needed e.g. in the encoding of minimization from Kleene algebra.

As an example, consider how the encoding of a Turing machine M may look like, in the λ -calculus. Let t_M be the encoding of M 's transition function (which is typically very simple if states and tapes are encoded using, e.g., Scott's numerals [36]). Then, the (recursively defined) function that

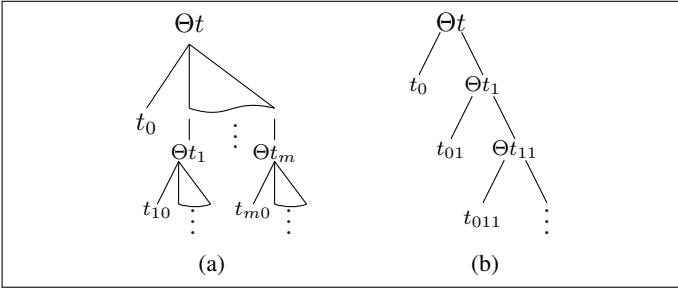


Figure 6: Different ways Θ can copy its argument.

iterates t_M , thus capturing the overall behavior of M , can be written as follows:

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

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 Θ , such as Turing's:

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

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

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 6a. This is a faithful description of how recursion unfolds, independently on how t uses its argument.

If $t = \text{iteraux}$, however, the situation is much simpler: t uses its argument at most once, and the complicated tree in Figure 6a becomes the one in Figure 6b. 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 6b is nothing more than the number of reduction steps the Turing machine M performs.

In order to understand if the λ IAM could be reasonable in space, *i.e.* if it can simulate Turing machines with a constant overhead in space, we apply our technique by giving Θ a suitable type (schema) $\mathbb{F}_n^{\bar{A}}$, depending on a list of types $\bar{A} := A_n, \dots, A_0$ and on n , which is the number of times the fixed point is unfolded. In particular, $\mathbb{F}_n^{\bar{A}}$ turns out to be *sparse*. The details of the technical development can be found in the Appendix.

Proposition VIII.1. *For each $n \geq 0$, and for each list of types \bar{A} such that $|\bar{A}| \geq n + 1$,*

$$\vdash \Theta : \mathbb{F}_n^{\bar{A}}.$$

In particular, $\Theta : \mathbb{F}_n^{\bar{A}}$ when Θ is unfolded n times and thus, inside the encoding of a Turing machine M which takes n steps to halt. This way the λ IAM, independently of the space consumption of M , requires space at least linear in the number of reduction steps it performs, and there is thus no hope to stay within sub-linear space constraints.

(In)Efficiency in Related Works: How could we reconcile all this with the claims from the literature about the existence of sub-linear space bounds (Mazza [32] and Dal Lago and Schöpp [17], [18]) within the realm of geometry of interaction machines? The answer is relatively simple: the kind of machines considered in the cited works are *fundamentally different* than ours, being based on the idea that the information stored in logged positions can be taken to be a natural number *smaller or equal to* the cardinality of the underlying multi type. In Mazza [32], this is possible due to the peculiarities of the underlying type system. There, the use of non-linearity is much more restricted than in the λ -calculus, being based on parsimonious types. This allows for tail-recursive schemas *only*: in our terminology, all trees are linearly shaped, and then logged positions can be represented differently and with less space, namely by simply taking the height of the tree. The works by Dal Lago and Schöpp [17], [18] rely on the fact that the underlying type system is resource-aware, this way allowing for a different representation of logged positions as natural numbers rather than trees. Moreover, space-efficient simulations are done via an ad-hoc combinator for skewed iterations, thus circumventing the problem with fixed-point operators. This, unfortunately, is not available in the realm of the λ -calculus.

A question, however, remains. Is it possible *at all* that the λ IAM consumes an amount of computation *space* significantly smaller than computation *time*? The answer is positive: there is a family of terms $\{t_i\}_{i \in \mathbb{N}}$ such that t_i reduces in Closed CbN to normal form in time exponential in i (namely, taking an exponential number of β -steps) but requires space linear in i , when reduced by the λ IAM. Details are in the Appendix.

IX. CONCLUSIONS

Space efficiency of higher-order languages accommodating sub-linear complexity is a topic about which almost nothing is known. The literature has suggested that the right tool to achieve it is the alternative paradigm of the IAM, a machine rooted in the geometry of interaction, without however clarifying whether its use of space can be used as a reasonable space cost model.

In this paper we develop a sharp tool—a type system—for the understanding of the space consumption of the (λ)IAM. Our new tree intersection types provide—for the first time—exact space bounds, via a simple system of weights measuring the depth of the tree structure in the types.

The tree structure seems to naturally complement the multiset one needed for measuring time, and it is of independent interest, given the relevance of intersection types in semantics. For instance, can such a structure be seen in game semantics? What does it measure with respect to cut-elimination or environment machines?

Beyond the theoretical result, the type system has a direct application, as we show by studying the space usage of the IAM on the traditional encoding of Turing machines. Such a usage turns out to be very inefficient, providing negative insights on the elusive conjecture about the reasonable space usage of the IAM.

Acknowledgements. The second author is funded by the ERC CoG “DIAPASoN” (GA 818616). This work has been partially funded by the ANR JCJC grant “COCA HOLA” (ANR-16-CE40-004-01).

REFERENCES

- [1] S. Abramsky, R. Jagadeesan, and P. Malacaria, “Full abstraction for PCF,” *Inf. Comput.*, vol. 163, no. 2, pp. 409–470, 2000.
- [2] B. Accattoli and B. Barras, “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*, W. Vanhoof and B. Pientka, Eds. ACM, 2017, pp. 4–16.
- [3] B. Accattoli and U. Dal Lago, “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, ser. LIPIcs, A. Tiwari, Ed., vol. 15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 22–37.
- [4] —, “(Leftmost-Outermost) Beta Reduction is Invariant, Indeed,” *Logical Methods in Computer Science*, vol. 12, no. 1, 2016.
- [5] B. Accattoli, U. Dal Lago, and G. Vanoni, “The machinery of interaction,” in *PPDP ’20: 22nd International Symposium on Principles and Practice of Declarative Programming, Bologna, Italy, 9-10 September, 2020*. ACM, 2020, pp. 4:1–4:15.
- [6] —, “The (in)efficiency of interaction,” *Proc. ACM Program. Lang.*, vol. 5, no. POPL, pp. 1–33, 2021.
- [7] B. Accattoli, S. Graham-Lengrand, and D. Kesner, “Tight typings and split bounds, fully developed,” *J. Funct. Program.*, vol. 30, p. e14, 2020.
- [8] B. Accattoli and G. Guerrieri, “Types of fireballs,” in *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings*, ser. Lecture Notes in Computer Science, S. Ryu, Ed., vol. 11275. Springer, 2018, pp. 45–66.
- [9] B. Accattoli, G. Guerrieri, and M. Leberle, “Types by need,” in *28th European Symposium on Programming, ESOP 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, ser. Lecture Notes in Computer Science, vol. 11423. Springer, 2019, pp. 410–439.
- [10] S. Alves, D. Kesner, and D. Ventura, “A quantitative understanding of pattern matching,” in *25th International Conference on Types for Proofs and Programs, TYPES 2019, June 11-14, 2019, Oslo, Norway*, 2019, pp. 3:1–3:36.
- [11] H. P. Barendregt, *The lambda calculus: its syntax and semantics*. North-Holland, 1984.
- [12] A. Bucciarelli, D. Kesner, A. Ríos, and A. Viso, “The bang calculus revisited,” in *Functional and Logic Programming - 15th International Symposium, FLOPS 2020, Akita, Japan, September 14-16, 2020, Proceedings*, ser. Lecture Notes in Computer Science, K. Nakano and K. Sagonas, Eds., vol. 12073. Springer, 2020, pp. 13–32.
- [13] L. Cardelli, “Type systems,” in *The Computer Science and Engineering Handbook*, A. B. Tucker, Ed. CRC Press, 1997, pp. 2208–2236.
- [14] M. Coppo and M. Dezani-Ciancaglini, “A new type assignment for λ -terms,” *Arch. Math. Log.*, vol. 19, no. 1, pp. 139–156, 1978.
- [15] U. Dal Lago and B. Accattoli, “Encoding turing machines into the deterministic lambda-calculus,” *CoRR*, vol. abs/1711.10078, 2017.
- [16] U. Dal Lago, C. Faggian, and S. Ronchi Della Rocca, “Intersection types and (positive) almost-sure termination,” *Proc. ACM Program. Lang.*, vol. 5, no. POPL, pp. 1–32, 2021.
- [17] U. Dal Lago and U. Schöpp, “Functional programming in sublinear space,” in *19th European Symposium on Programming, ESOP 2010, Paphos, Cyprus, March 20-28, 2010, Proceedings.*, ser. Lecture Notes in Computer Science, A. D. Gordon, Ed., vol. 6012. Springer, 2010, pp. 205–225.
- [18] U. Dal Lago and U. Schöpp, “Computation by interaction for space-bounded functional programming,” *Information and Computation*, vol. 248, pp. 150–194, 2016.
- [19] V. Danos, H. Herbelin, and L. Regnier, “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, 1996, pp. 394–405.
- [20] V. Danos and L. Regnier, “Proof-nets and the hilbert space,” in *Proceedings of the Workshop on Advances in Linear Logic*. USA: Cambridge University Press, 1995, p. 307–328.
- [21] D. de Carvalho, “Sémantiques de la logique linéaire et temps de calcul,” Thèse de Doctorat, Université Aix-Marseille II, 2007.
- [22] —, “Execution time of λ -terms via denotational semantics and intersection types,” *Math. Str. in Comput. Sci.*, vol. 28, no. 7, pp. 1169–1203, 2018.
- [23] D. de Carvalho, M. Pagani, and L. Tortora de Falco, “A semantic measure of the execution time in linear logic,” *Theoretical Computer Science*, vol. 412, no. 20, pp. 1884–1902, 2011.
- [24] Y. Forster, F. Kunze, and M. Roth, “The weak call-by-value λ -calculus is reasonable for both time and space,” *Proc. ACM Program. Lang.*, vol. 4, no. POPL, pp. 27:1–27:23, 2020.
- [25] D. R. Ghica, “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*, M. Hofmann and M. Felleisen, Eds. ACM, 2007, pp. 363–375.
- [26] D. R. Ghica and A. I. Smith, “Geometry of synthesis II: from games to delay-insensitive circuits,” in *Proceedings of the 26th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2010, Ottawa, Ontario, Canada, May 6-10, 2010*, ser. Electronic Notes in Theoretical Computer Science, M. W. Mislove and P. Selinger, Eds., vol. 265. Elsevier, 2010, pp. 301–324.
- [27] J.-Y. Girard, “Geometry of interaction I: Interpretation of system f ,” in *Logic Colloquium ’88*, ser. Studies in Logic and the Foundations of Mathematics, R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo, Eds. Elsevier, 1989, vol. 127, pp. 221 – 260.
- [28] D. Kesner and P. Vial, “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*, H. Hermanns, L. Zhang, N. Kobayashi, and D. Miller, Eds. ACM, 2020, pp. 619–632.
- [29] J. Krivine, *Lambda-calculus, types and models*, ser. Ellis Horwood series in computers and their applications. Masson, 1993.
- [30] J.-L. Krivine, “A Call-by-name Lambda-calculus Machine,” *Higher Order Symbol. Comput.*, vol. 20, no. 3, pp. 199–207, 2007.
- [31] I. Mackie, “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*, R. K. Cytron and P. Lee, Eds. ACM Press, 1995, pp. 198–208.
- [32] D. Mazza, “Simple parsimonious types and logarithmic space,” in *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, ser. LIPIcs, S. Kreutzer, Ed., vol. 41. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 24–40.
- [33] D. Mazza, L. Pellissier, and P. Vial, “Polyadic approximations, fibrations and intersection types,” *Proc. ACM Program. Lang.*, vol. 2, no. POPL, pp. 6:1–6:28, 2018.
- [34] U. Schöpp, “Stratified bounded affine logic for logarithmic space,” in *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wrocław, Poland, Proceedings*. IEEE Computer Society, 2007, pp. 411–420.
- [35] T. Tsukada, K. Asada, and C.-H. L. Ong, “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, 2017, pp. 1–12.
- [36] C. P. Wadsworth, “Some unusual λ -calculus numeral systems,” in *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, J. P. Seldin and J. R. Hindley, Eds., 1980, pp. 215–230.

APPENDIX A
PROOF OF PROPOSITION III.4

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

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

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

Proposition A.2. *Let s be a reachable λ IAM state. Then s is reached exactly once.*

Proof. We proceed by induction on the length of the run $\rho : s_0 \rightarrow_{\lambda\text{IAM}}^* s$. If $|\rho| = 0$, the result is trivial. Otherwise, if $|\rho| > 0$, we have $\rho : s_0 \rightarrow_{\lambda\text{IAM}}^* s' \rightarrow_{\lambda\text{IAM}} s$. Let us call $\sigma : s_0 \rightarrow_{\lambda\text{IAM}}^* s'$. 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 . \square

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

Proposition A.3. *Let ρ be an infinite λ IAM run. Then $|\rho|_{\text{sp}} = \infty$.*

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 λ IAM needs an unbounded space to perform the computation. \square

APPENDIX B
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 T_{MANY} . It is the quantity that is used to prove the termination argument for typed terms.

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 B.1 (Tree splitting and merging). *Let $T = T_1 \uplus \dots \uplus T_k$. 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 $i \in \{1, \dots, k\}$. Moreover, $[\Gamma] = [\Gamma_1 \uplus \dots \uplus \Gamma_k]$ and $|\pi| = \sum_{i=1}^k |\pi_i|$.*

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.

$$\frac{\Gamma_i \vdash t : G_i \quad 1 \leq i \leq n}{[\uplus_{i=1}^n \Gamma_i] \vdash t : [G_1, \dots, G_n] = T_1 \uplus \dots \uplus T_k} \text{T-MANY}$$

We can prove the statement considering k derivations, each of them deriving the judgment $t : T_j$.

$$\frac{\Gamma_i \vdash t : G_i \quad 1 \leq i \leq |T_j|}{[\uplus_{i=1}^{|T_j|} \Gamma_i] \vdash t : [G_1, \dots, G_{|T_j|}] = T_j} \text{T-MANY}$$

\square

Lemma B.2 (Quantitative substitution). *Let $\pi_t \triangleright \Gamma, x : T \vdash t : G$ and $\pi_u \triangleright \vdash u : T$. Then there exists $\pi_{t\{x \leftarrow u\}} \triangleright \Gamma \vdash t\{x \leftarrow u\} : A$ 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- λ_* .* Then $t = \lambda y.r$, $G = *$, 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- λ_* 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:

$$\frac{\begin{array}{c} \vdots \\ \pi_r \triangleright \Gamma, x : T, y : T' \vdash r : A' \end{array}}{\Gamma, x : T \vdash \lambda y.r : T' \rightarrow A'} \text{ T-}\lambda$$

with $t = \lambda y.r$ and $G = T' \rightarrow A'$. By *i.h.*, there exists a derivation $\pi_{r\{x \leftarrow u\}} \triangleright \Gamma, y : T' \vdash r\{x \leftarrow u\} : A'$ such that $|\pi_{r\{x \leftarrow u\}}| = |\pi_r| + |\pi_u| - |T^\ell|$. Applying back rule T- λ we obtain $\pi_{(\lambda y.r)\{x \leftarrow u\}}$

$$\frac{\begin{array}{c} \vdots \\ \pi_{r\{x \leftarrow u\}} \triangleright \Gamma, y : T' \vdash r\{x \leftarrow u\} : A' \end{array}}{\Gamma \vdash \lambda y.r\{x \leftarrow u\} : T' \rightarrow A'} \text{ T-}\lambda$$

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:

$$\frac{\begin{array}{c} \vdots \\ \pi_r \triangleright x : T_1, \Gamma \vdash r : T' \rightarrow A \end{array} \quad \begin{array}{c} \vdots \\ \pi_w \triangleright x : T_2, \Delta \vdash w : T' \end{array}}{x : T_1 \uplus T_2, \Gamma \uplus \Delta \vdash rw : A} \text{ T-@}$$

with $t = rw$, $G = A$, and $T = T_1 \uplus T_2$. By Lemma B.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) $\pi_{r\{x \leftarrow u\}} \triangleright \Gamma \vdash r\{x \leftarrow u\} : T' \rightarrow A$ 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:

$$\frac{\pi_{r\{x \leftarrow u\}} \triangleright \Gamma \vdash r\{x \leftarrow u\} : T' \rightarrow A \quad \pi_{w\{x \leftarrow u\}} \triangleright \Delta \vdash w\{x \leftarrow u\} : T'}{\Gamma \uplus \Delta \vdash r\{x \leftarrow u\}w\{x \leftarrow u\} : A} \text{ T-@}$$

for which

$$\begin{aligned} |\pi_{(rw)\{x \leftarrow u\}}| &= |\pi_{r\{x \leftarrow u\}}| + |\pi_{w\{x \leftarrow u\}}| + 1 \\ &=_{i.h.} |\pi_r| + |\pi_u^1| - |T_1^\ell| + |\pi_w| + |\pi_u^2| - |T_2^\ell| + 1 \\ &= (|\pi_r| + |\pi_w| + 1) + (|\pi_u^1| + |\pi_u^2|) - (|T_1^\ell| + |T_2^\ell|) \\ &= |\pi_{rw}| + |\pi_u| - |T^\ell| \end{aligned}$$

- **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:

$$\frac{\begin{array}{c} \vdots \\ \Gamma_i, x : T_i \vdash t : G_i \quad 1 \leq i \leq n \end{array}}{[\uplus_{i=1}^n \Gamma_i], x : [\uplus_{i=1}^n T_i] \vdash t : [G_1, \dots, G_n]} \text{ T-MANY}$$

By Lemma B.1, the hypothesis $\pi_u \triangleright \vdash u : T$ splits into n derivations $\pi_u^i \triangleright \vdash u : T_i$ with $i \in \{1, \dots, n\}$ such that $|\pi_u| = \sum_{i=1}^n |\pi_u^i|$. By *i.h.*, there exist n derivations $\pi_{t\{x \leftarrow u\}}^i \triangleright \Gamma_i \vdash t\{x \leftarrow u\} : G_i$ 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:

$$\frac{\begin{array}{c} \vdots \\ \pi_{t\{x \leftarrow u\}}^i \triangleright \Gamma_i \vdash t\{x \leftarrow u\} : G_i \quad 1 \leq i \leq n \end{array}}{[\uplus_{i=1}^n \Gamma_i] \vdash t\{x \leftarrow u\} : [G_1, \dots, G_n]} \text{ T-MANY}$$

for which

$$\begin{aligned} |\pi_{t\{x \leftarrow u\}}| &= \sum_{i=1}^n |\pi_{t\{x \leftarrow u\}}^i| \\ &=_{i.h.} \sum_{i=1}^n (|\pi_t|^i + |\pi_u^i| - |T_i^\ell|) \\ &= \sum_{i=1}^n |\pi_t|^i + \sum_{i=1}^n |\pi_u^i| - \sum_{i=1}^n |T_i^\ell| \\ &= |\pi_t| + |\pi_u| - |T^\ell| \end{aligned}$$

□

Proposition B.3 (Quantitative Subject Reduction). *If t is closed, $\pi \triangleright \vdash t : A$, and $t \rightarrow_{wh} u$ then there exists $\pi' \triangleright \vdash u : A$ such that $|\pi| > |\pi'|$.*

Proof. By induction on $t \rightarrow_{wh} u$.

- *Base case, step at top level:* $t = (\lambda x.r)w \rightarrow_{wh} r\{x \leftarrow w\} = u$. Note that w is closed because t is closed by hypothesis. Then π has the following shape.

$$\frac{\frac{\frac{\vdots}{\pi_r \triangleright x : T \vdash r : A} \text{T-}\lambda}{\vdash \lambda x.r : T \rightarrow A} \text{T-}\lambda \quad \frac{\vdots}{\pi_w \triangleright \vdash w : T} \text{T-}@}{\vdash (\lambda x.r)w : A} \text{T-}@$$

We can apply the quantitative substitution lemma (Lemma B.2) to the sub-derivations π_r and π_w obtaining a derivation $\pi_{r\{x \leftarrow w\}} \triangleright \vdash r\{x \leftarrow w\} : A$ 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:* $t = rw \rightarrow_{wh} r'w = u$ with $r \rightarrow_{wh} r'$. Then π has the following shape.

$$\frac{\frac{\vdots}{\pi_r \triangleright \vdash r : T \rightarrow A} \quad \frac{\vdots}{\pi_w \triangleright \vdash w : T} \text{T-}@}{\vdash rw : A} \text{T-}@$$

Applying the *i.h.* to the left sub-derivation π_r , we obtain $\pi'_{r'} \triangleright \vdash r' : T \rightarrow A$ such that $|\pi_r| > |\pi'_{r'}|$. Then π' is defined as follows.

$$\pi' := \frac{\frac{\vdots}{\pi'_{r'} \triangleright \vdash r' : T \rightarrow A} \quad \frac{\vdots}{\pi_w \triangleright \vdash w : T} \text{T-}@}{\vdash r'w : A} \text{T-}@$$

for which $|\pi| = |\pi_r| + 1 + |\pi_w| >_{i.h.} |\pi'_{r'}| + 1 + |\pi_w| = |\pi'|$, as required.

□

Theorem B.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 $t \rightarrow_{wh}$ -reduces. Cases:

- 1) t does not reduce. Then it is \rightarrow_{wh} -normal.
- 2) $t \rightarrow_{wh} u$ for some u . By quantitative subject reduction (Prop. B.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 .

□

a) *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 B.5 (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 lexicographic 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:

$$\frac{\frac{\vdots}{\pi^i \triangleright \Gamma_i \vdash t\{x \leftarrow u\} : G_i} \quad 1 \leq i \leq n}{[\uplus_{i=1}^n \Gamma_i] \vdash t\{x \leftarrow u\} : [G_1, \dots, G_n]} \text{T-MANY}$$

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 B.1, the derivations π_u^i merge into a derivation $\pi_u \triangleright \vdash u : T$ where $T := T_1 \uplus \dots \uplus T_n$. The derivation π_t is instead obtained as follows.

$$\frac{\begin{array}{c} \vdots \\ \Gamma_i, x : T_i \vdash t : G_i \quad 1 \leq i \leq n \end{array}}{[\uplus_{i=1}^n \Gamma_i], x : [\uplus_{i=1}^n T_i] \vdash t : [G_1, \dots, G_n]} \text{T-MANY}$$

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- λ_\star .* Then $t = \lambda y.r$ and $G = \star$. The statements holds by taking
 - $T := [\cdot]$
 - π_t as a T- λ_\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:

$$\frac{\begin{array}{c} \vdots \\ \pi_{r\{x \leftarrow u\}} \triangleright \Gamma, y : T' \vdash r\{x \leftarrow u\} : A' \end{array}}{\Gamma \vdash \lambda y.r\{x \leftarrow u\} : T' \rightarrow A'} \text{T-}\lambda$$

with $t = \lambda y.r$ and $A = T' \rightarrow A'$. By *i.h.* (1st component), there exists T and derivations $\pi_r \triangleright \Gamma, x : T, y : T' \vdash r : A'$ and $\pi_u \triangleright \vdash u : T$. Applying back rule T- λ to π_r , we obtain $\pi_{\lambda y.r}$ as follows.

$$\frac{\begin{array}{c} \vdots \\ \pi_r \triangleright \Gamma, x : T, y : T' \vdash r : A' \end{array}}{\Gamma, x : T \vdash \lambda y.r : T' \rightarrow A'} \text{T-}\lambda$$

- *Rule T-@.* Then π has the following shape:

$$\frac{\pi_{r\{x \leftarrow u\}} \triangleright \Gamma \vdash r\{x \leftarrow u\} : T' \rightarrow A \quad \pi_{w\{x \leftarrow u\}} \triangleright \Delta \vdash w\{x \leftarrow u\} : T'}{\Gamma \uplus \Delta \vdash r\{x \leftarrow u\}w\{x \leftarrow u\} : A} \text{T-@}$$

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 $\pi_r \triangleright \Gamma, x : T_1 \vdash r : T' \rightarrow A$ 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 B.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.

$$\frac{\begin{array}{c} \vdots \\ \pi_r \triangleright x : T_1, \Gamma \vdash r : T' \rightarrow A \end{array} \quad \begin{array}{c} \vdots \\ \pi_w \triangleright x : T_2, \Delta \vdash w : T' \end{array}}{x : \underbrace{T_1 \uplus T_2}_T, \Gamma \uplus \Delta \vdash rw : A} \text{T-@}$$

□

Proposition B.6 (Subject expansion). *If t is closed, $\pi \triangleright \vdash u : A$, and $t \rightarrow_{wh} u$ then there exists $\pi' \triangleright \vdash t : A$.*

Proof. By induction on $t \rightarrow_{wh} u$.

- *Base case, step at top level:* $t = (\lambda x.r)w \rightarrow_{wh} r\{x \leftarrow w\} = u$. 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 B.5) 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:

$$\frac{\frac{\frac{\vdots}{\pi_r \triangleright x : T \vdash r : A}}{\vdash \lambda x.r : T \rightarrow A} \text{T-}\lambda \quad \frac{\frac{\vdots}{\pi_w \triangleright \vdash w : T}}{\vdash (\lambda x.r)w : A} \text{T-}@}{\vdash (\lambda x.r)w : A} \text{T-}@$$

- *Inductive case, step on the left of the root application:* $t = rw \rightarrow_{wh} r'w = u$ with $r \rightarrow_{wh} r'$. Then π has the following shape.

$$\frac{\frac{\frac{\vdots}{\pi_{r'} \triangleright \vdash r' : T \rightarrow A} \quad \frac{\frac{\vdots}{\pi_w \triangleright \vdash w : T}}{\vdash r'w : A} \text{T-}@}}{\vdash r'w : A} \text{T-}@$$

Applying the *i.h.* to the left sub-derivation $\pi_{r'}$, we obtain $\pi'_r \triangleright \vdash r : T \rightarrow A$. Then π' is defined as follows.

$$\pi' := \frac{\frac{\frac{\vdots}{\pi'_r \triangleright \vdash r : T \rightarrow A} \quad \frac{\frac{\vdots}{\pi_w \triangleright \vdash w : T}}{\vdash r'w : A} \text{T-}@}}{\vdash rw : A} \text{T-}@$$

□

Theorem B.7 (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 $t \rightarrow_{wh}^n u$ 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- λ_* .
- 2) If $n > 0$ then $t \rightarrow_{wh} r \rightarrow_{wh}^{n-1} u$. By *i.h.*, there exists $\pi'' \triangleright \vdash r : A$. By subject expansion Prop. B.6, there exists $\pi' \triangleright \vdash t : A$.

□

APPENDIX C PROOFS FROM SECTION VI

In the first part of this section we prove the T-exhaustible state invariant for the TIAM, then use it to extract λ IAM states from TIAM ones, and finally prove the strong bisimulation between the two machines.

In the second part we deal with showing that the TIAM never loops on type derivations. The key tool shall be a loop-preserving bisimulation between TIAM states of the type derivation of t and u if $t \rightarrow_{wh} u$.

1) *T-Exhaustible Invariant:* We present an example of type derivation for the term $t = (\lambda y.\lambda x.xy)(\lambda z.z)$, the same example used in Section III. We use it to explain the next technical definitions. We have annotated the occurrences of \star with natural numbers, so that they represent the run on the type derivation.

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. V, a state is a quadruple (π, J, \mathbb{A}, d) where J is an occurrence of a judgement $\Gamma \vdash u : A$ 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 $\mathbb{A}\langle A' \rangle = A$ for some A' , that is, not necessarily isolating \star . A pair (A', \mathbb{A}) such that $\mathbb{A}\langle A' \rangle = A$ 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 A' , not necessarily \star —it can be found in Fig. 7, and it amounts to simply replace \star with A' .

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.

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

The intuition is that a test focuses on (the occurrence of) a leaf A' of a tree T related to s , and that these leaf elements play the role of logged positions in the λ IAM. 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 λ IAM. Note that the positions on the log are those for which the λ IAM has previously found the corresponding arguments. In the TIAM these arguments are exactly those in which the focused judgment is contained.

$\frac{\frac{\vdash t : T \rightarrow A \quad \vdash}{\vdash tu : \mathbb{A}\langle A'_{\uparrow} \rangle (= A)} \quad \rightarrow_{\bullet 1}}{\vdash t : T \rightarrow \mathbb{A}\langle A'_{\uparrow} \rangle \quad \vdash} \quad \vdash tu : A$	$\frac{\vdash t : T \rightarrow \mathbb{A}\langle A'_{\uparrow} \rangle \quad \vdash}{\vdash tu : A}$	$\frac{\vdash t : A (= \mathbb{A}\langle A' \rangle)}{\vdash \lambda x.t : T \rightarrow \mathbb{A}\langle A'_{\uparrow} \rangle} \quad \rightarrow_{\bullet 2} \quad \frac{\vdash t : \mathbb{A}\langle A'_{\uparrow} \rangle}{\vdash \lambda x.t : T \rightarrow A}$
$\frac{\vdash t : T \rightarrow \mathbb{A}\langle A'_{\downarrow} \rangle \quad \vdash}{\vdash tu : A (= \mathbb{A}\langle A' \rangle)} \quad \rightarrow_{\bullet 3} \quad \frac{\vdash t : T \rightarrow A \quad \vdash}{\vdash tu : \mathbb{A}\langle A'_{\downarrow} \rangle}$	$\frac{\vdash t : \mathbb{A}\langle A'_{\downarrow} \rangle (= A)}{\vdash \lambda x.t : T \rightarrow A} \quad \rightarrow_{\bullet 4} \quad \frac{\vdash t : A}{\vdash \lambda x.t : T \rightarrow \mathbb{A}\langle A'_{\downarrow} \rangle}$	
$\frac{\frac{\vdash x : \mathbb{A}\langle A'_{\uparrow} \rangle_i (= A_i) \quad \vdash \dots}{\vdash \lambda x.C(x) : \mathbb{L}\langle A_i \rangle \rightarrow A''} \quad \rightarrow_{\text{var}} \quad \frac{\vdash x : A_i \quad \vdash \dots}{\vdash \lambda x.C(x) : \mathbb{L}\langle \mathbb{A}\langle A'_{\downarrow} \rangle_i \rangle \rightarrow A''}}{\vdash \lambda x.C(x) : \mathbb{L}\langle A_i \rangle \rightarrow A''} \quad \rightarrow_{\text{bt2}} \quad \frac{\vdash x : \mathbb{A}\langle A'_{\downarrow} \rangle_i \quad \vdash \dots}{\vdash \lambda x.C(x) : \mathbb{L}\langle A_i \rangle \rightarrow A''}}$		
$\frac{\vdash t : \mathbb{L}\langle \mathbb{A}\langle A'_{\downarrow} \rangle_i \rangle \rightarrow A \quad \frac{\dots \vdash u : \mathbb{A}\langle A' \rangle_i \quad \dots}{\vdash u : T (= \mathbb{L}\langle \mathbb{A}\langle A' \rangle_i \rangle)} \text{T-MANY}}{\vdash tu : A} \quad \rightarrow_{\text{arg}} \quad \frac{\vdash t : T \rightarrow A \quad \frac{\dots \vdash u : \mathbb{A}\langle A'_{\uparrow} \rangle_i \quad \dots}{\vdash u : \mathbb{L}\langle \mathbb{A}\langle A' \rangle_i \rangle} \text{T-MANY}}{\vdash tu : A}$		
$\frac{\vdash t : T \rightarrow A \quad \frac{\dots \vdash u : \mathbb{A}\langle A'_{\downarrow} \rangle_i \quad \dots}{\vdash u : \mathbb{L}\langle \mathbb{A}\langle A' \rangle_i \rangle (= T)} \text{T-MANY}}{\vdash tu : A} \quad \rightarrow_{\text{bt1}} \quad \frac{\vdash t : \mathbb{L}\langle \mathbb{A}\langle A'_{\uparrow} \rangle_i \rangle \rightarrow A \quad \frac{\dots \vdash u : \mathbb{A}\langle A' \rangle_i \quad \dots}{\vdash u : T} \text{T-MANY}}{\vdash tu : A}$		

Figure 7: The transitions of the (Generalized) Tree IAM (TIAM).

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 λ IAM. They correspond to λ IAM queries for which the argument has not yet been found, or positions to which the λ IAM is backtracking to.

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

Definition C.1 (Focus). A focus f in a derivation π is a pair $f = (J, (A, \mathbb{A}))$ of a judgement 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 C.2 (Judgement 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 $\Gamma \vdash t : A'$. Then $s_f^i = (\pi, J_i, (A', \langle \cdot \rangle), \downarrow)$ is the i -th judgement test of s , having as focus $f := (J_i, (A', \langle \cdot \rangle))$.

We often omit the judgement from the focus, writing simply $s_{(A', \langle \cdot \rangle)}$, and even concisely note s_f as $\vdash t : A', \langle \cdot \rangle_{\downarrow}$.

Note that judgement tests always have type context $\langle \cdot \rangle$. According to the intended correspondance judgement/ log and type context/tape between the TIAM and the λ IAM, having type context $\langle \cdot \rangle$ corresponds to the fact that the log tests of the λ IAM have an empty tape.

Type (Context) Tests: While judgement tests depend only on the judgement 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 $(J, (A', \mathbb{A}'))$ where $\mathbb{A}'\langle A' \rangle = \mathbb{A}\langle A \rangle$ 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:

$$\begin{aligned} \mathbb{A}_0 &:= \langle \cdot \rangle \mid T \rightarrow \mathbb{A}_0 \\ \mathbb{A}_{n+1} &:= \mathbb{L}\langle \mathbb{A}_n \rangle \rightarrow A \mid T \rightarrow \mathbb{A}_{n+1} \end{aligned}$$

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 $\mathbb{A}'\langle \mathbb{L} \rightarrow A \rangle$ for a type context \mathbb{A}' of level $i - 1$.

Definition C.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 $\text{DiPref}(\mathbb{A})$ of \mathbb{A} is the sequence of pairs (\mathbb{A}', d') , where \mathbb{A}' is a prefix of \mathbb{A} , defined as follows:

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

The i -th directed prefix (from left to right) (\mathbb{A}', d') in $\text{DiPref}(\mathbb{A})$ induces the type test $s_f^i := (\pi, J, (\mathbb{A}''\langle A \rangle, \mathbb{A}'), d')$ of s and focus $f := (J, (\mathbb{A}''\langle A \rangle, \mathbb{A}'))$, 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 $\text{DiPref}(\mathbb{A})$ has \uparrow direction, and that the direction alternates along the sequence. This is 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 C.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 C.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 judgement test of s_f of focus f there exists a run $\rho : s_f \rightarrow_{\text{TIAM}}^* s'$ where s' respects f and for the shortest such run $s' \in \mathcal{E}_T$.

Lemma C.6 (T-exhaustible invariant). Let t be a closed term, $\pi \triangleright \Gamma \vdash t : A$ a tree type derivation for it, and $\sigma : \vdash t : \langle A \rangle_{\uparrow} \rightarrow_{\text{TIAM}}^k s$ 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 judgement nor type tests. Then suppose $\sigma' : s_0 \rightarrow_{\text{TIAM}}^{k-1} s'$ and that the run continues with $s' \rightarrow_{\text{TIAM}} s$. 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 $s' \rightarrow_{\text{TIAM}} s$:

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

$$s' = \frac{\vdash t : T \rightarrow A \quad \vdash}{s' = \vdash tu : \mathbb{A}\langle \star_{\uparrow} \rangle (= A)} \rightarrow_{\bullet 1} \frac{\vdash t : T \rightarrow \mathbb{A}\langle \star_{\uparrow} \rangle \quad \vdash}{\vdash tu : A} = s$$

- *Judgement tests.* Note that s has the same judgement 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 $s'_f \rightarrow_{\text{TIAM}} s_f$. 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 $s_f \rightarrow s'_f$. Then s_f is positive.

- Case $\rightarrow_{\bullet 2}$. Identical to the previous one.

- Case \rightarrow_{var} .

$$s' = \frac{\overline{\vdash x : \mathbb{A}\langle \star_{\uparrow} \rangle_i (= A_i)}^i}{\vdash \lambda x. C\langle x \rangle : \mathbb{L}\langle A_i \rangle \rightarrow A'} \rightarrow_{\text{var}} \frac{\overline{\vdash x : A_i}^i}{\vdash \lambda x. C\langle x \rangle : \mathbb{L}\langle \mathbb{A}\langle \star_{\downarrow} \rangle_i \rangle \rightarrow A'} = s$$

- *Judgement tests.* Judgement tests of s are a subset of judgement 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 $\text{DiPref}(\mathbb{L}\langle A_i \rangle \rightarrow A')$. Three cases, depending on the index j of s^j :

- 1) $j = 1$: then s^1 is $\vdash \lambda x. C\langle x \rangle : \mathbb{A}\langle \star \rangle_{i\uparrow}, \mathbb{L} \rightarrow A'$. Note that $s^1 \rightarrow_{\text{bt2}} \vdash x : \mathbb{A}\langle \star \rangle_{i\downarrow}, \langle \cdot \rangle$, which has no type tests and has the same judgement 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 $s'^{j-1} \rightarrow_{\text{var}} s^j$. 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 $s^j \rightarrow_{\text{bt2}} s'^{j-1}$. Thus one can conclude by *i.h.*

- Case \rightarrow_{bt2} .

$$s' = \frac{\overline{\vdash x : A_i (= \mathbb{A}\langle \star \rangle_i)}^i}{\vdash \lambda x. C\langle x \rangle : \mathbb{L}\langle \mathbb{A}\langle \star_{\uparrow} \rangle_i \rangle \rightarrow A'} \rightarrow_{\text{bt2}} \frac{\overline{\vdash x : \mathbb{A}\langle \star_{\downarrow} \rangle_i}^i}{\vdash \lambda x. C\langle x \rangle : \mathbb{L}\langle A_i \rangle \rightarrow A'} = s$$

- *Judgement tests.* The first type test of s' is $s'^1 := \vdash \lambda x. C(x) : \mathbb{A}(\star)_{i\uparrow}, \mathbb{L} \rightarrow A'$. Note that $s'^1 \rightarrow_{\text{bt}2} \vdash x : \mathbb{A}(\star)_{i\downarrow}, \langle \cdot \rangle := s''$ 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 judgement 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 $s^i \rightarrow_{\text{var}} s'^{i+1}$. 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 $s'^{i+1} \rightarrow_{\text{bt}2} s^i$. Then s^i is positive by determinism of the TIAM.

- Cases $\rightarrow_{\bullet 3}$ and $\rightarrow_{\bullet 4}$. They are identical to case $\rightarrow_{\bullet 1}$.
- Case \rightarrow_{arg} .

$$s' = \frac{\vdash t : \mathbb{L}(\mathbb{A}(\star)_{i\downarrow}) \rightarrow A \quad \frac{\dots \vdash u : \mathbb{A}(\star)_{i\uparrow} \dots}{\vdash u : T(=\mathbb{L}(\mathbb{A}(\star)_{i\downarrow}))} \text{T-MANY}}{\vdash tu : A} \rightarrow_{\text{arg}} \frac{\vdash t : T \rightarrow A \quad \frac{\dots \vdash u : \mathbb{A}(\star)_{i\uparrow} \dots}{\vdash u : \mathbb{L}(\mathbb{A}(\star)_{i\downarrow})} \text{T-MANY}}{\vdash tu : A} = s$$

- *Judgement tests.* Judgement tests of s are those of s' , which are positive by *i.h.*, plus $s^u := \vdash u : \mathbb{A}(\star)_{i\downarrow}, \langle \cdot \rangle$. Please note that $s^u \rightarrow_{\text{bt}1} \vdash t : \mathbb{A}(\star)_{i\uparrow}, \mathbb{L} \rightarrow A =: s'^t$. 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 $s'^{i+1} \rightarrow_{\text{arg}} s^i$. 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 $s^i \rightarrow_{\text{bt}1} s'^{i+1}$. Then s^i is positive.

- Case $\rightarrow_{\text{bt}1}$.

$$s' = \frac{\vdash t : T \rightarrow A' \quad \frac{\dots \vdash u : \mathbb{A}(\star)_{i\downarrow} \dots}{\vdash u : \mathbb{L}(\mathbb{A}(\star)_{i\uparrow})(=T)} \text{T-MANY}}{\vdash tu : A'} \rightarrow_{\text{bt}1} \frac{\vdash t : \mathbb{L}(\mathbb{A}(\star)_{i\uparrow}) \rightarrow A' \quad \frac{\dots \vdash u : \mathbb{A}(\star)_{i\downarrow} \dots}{\vdash u : T} \text{T-MANY}}{\vdash tu : A'} = s$$

- *Judgement tests.* All judgement tests of s are judgement test of s' , which are this way positive by *i.h.*
- *Type tests.* The first type test of s is $s^1 := \vdash t : \mathbb{A}(\star)_{i\uparrow}, [\dots \langle \cdot \rangle \dots] \rightarrow A$. Please note that $s'^u := \vdash u : \mathbb{A}(\star)_{i\downarrow}, \langle \cdot \rangle$ is a judgement test of s' such that $s'^u \rightarrow_{\text{bt}1} s^1$. 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 $s'^{i-1} \rightarrow_{\text{bt}1} s^i$. 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 $s^i \rightarrow_{\text{arg}} s'^{i-1}$. Then s^i is positive. □

2) *Extracting λ IAM States from TIAM T-Exhaustible States, and the λ IAM/TIAM Strong Bisimulation:* From T-exhaustible states one is able to *extract* λ IAM 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-exhaustibility allows recursive definition on T-exhaustible states to be well-behaved.

Definition C.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 judgement or type test of s . Since s is T-exhaustible, there is an exhausting run $s_f \rightarrow_{\text{TIAM}}^+ s' \in \mathcal{E}_T$. Let x be the variable of s' . Then the logged position extracted from s_f is $l_{\text{ext}}(s_f) := (x, \lambda x. D_n, l_{\text{ext}}(s'^1) \cdot \dots \cdot l_{\text{ext}}(s'^n))$, 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 judgement test of s' .*

Definition C.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 $\Gamma \vdash u : \mathbb{A}(A)$. The λ IAM state extracted from s is $s_{\text{ext}}(s) := (u, C_s, L_{\text{ext}}(s), T_{\text{ext}}(s), d)^7$ where*

- Context: C_s is the only term context such that $t = C_s(u)$;
- Log: $L_{\text{ext}}(s) := l_1 \cdot \dots \cdot l_i \cdot \dots \cdot l_n$ where $l_i = l_{\text{ext}}(s_f^i)$ where s_f^i is the i -th judgement test of s .
- Tape: $T_{\text{ext}}(s) = T_{\text{ext}}^s(\mathbb{A}, 0)$ where $T_{\text{ext}}^s(\mathbb{A}, i)$ is the auxiliary function defined by induction on \mathbb{A} as follows.

$$\begin{aligned} T_{\text{ext}}^s(\langle \cdot \rangle, i) &:= \epsilon \\ T_{\text{ext}}^s(T \rightarrow \mathbb{A}, i) &:= \bullet \cdot T_{\text{ext}}^s(\mathbb{A}, i) \\ T_{\text{ext}}^s(\mathbb{L}(\mathbb{A}) \rightarrow A', i) &:= l_{\text{ext}}(s_f^i) \cdot T_{\text{ext}}^s(\mathbb{A}, i+1) \end{aligned}$$

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

We use \simeq_{ext} for the extraction relation between T-exhaustible TIAM states and λ IAM states defined as $(s, s_{\text{ext}}(s)) \in \simeq_{\text{ext}}$.

⁷We leave the color of d unchanged, in the sense that $s_{\text{ext}}(s)$ is red/blue if s is red/blue, *i.e.* \downarrow becomes \uparrow and \uparrow becomes \downarrow .

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

Lemma C.9. *Let s be an T-exhaustible TIAM state and $s_{\text{ext}}(s) = (t, C_s, L_{\text{ext}}(s), T_{\text{ext}}(s), d)$ the λ IAM state extracted from it. Then the level of C_s is exactly the length of $L_{\text{ext}}(s)$, that is, $(t, C_s, L_{\text{ext}}(s))$ is a logged position.*

Proof. The length of $L_{\text{ext}}(s)$ is the number of judgement tests of s , which is the number of T-MANY rule trees, and thus of T-@ rules, traversed descending from the focused judgement J of s to the final judgement 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 judgement of π . \square

Proposition C.10 (TIAM- λ IAM bisimulation). *Let t a closed and \rightarrow_{wh} -normalizable term, and $\pi \triangleright \vdash t : \star$ a type derivation. Then \simeq_{ext} is a strong bisimulation between T-exhaustible TIAM states on π and λ IAM states on t . Moreover, if $s_\pi \simeq_{\text{ext}} s_\lambda$ then s_π is TIAM reachable if and only if s_λ is λ IAM 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_{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 λ IAM, the other half is essentially identical.

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

$$\begin{array}{c} \frac{\vdash t : T \rightarrow A \quad \vdash}{s' = \vdash \mathbf{t}u : \mathbb{A}\langle \star \uparrow \rangle (= A)} \quad \rightarrow_{\bullet 1} \quad \frac{\vdash t : T \rightarrow \mathbb{A}\langle \star \uparrow \rangle \quad \vdash}{\vdash \mathbf{t}u : A} = s \\ \simeq_{\text{ext}} \\ s_{\text{ext}}(s') = (\mathbf{t}u, C_{s'}, L_{\text{ext}}(s'), T_{\text{ext}}(s')) \quad \rightarrow_{\bullet 1} \quad (\mathbf{t}, C\langle \langle \cdot \rangle r \rangle, L_{\text{ext}}(s'), \bullet \cdot T_{\text{ext}}(s')) = s_\lambda \end{array}$$

Note that $C_s = C_{s'}\langle \langle \cdot \rangle r \rangle$, $L_{\text{ext}}(s) = L_{\text{ext}}(s')$, and $T_{\text{ext}}(s) = \bullet \cdot T_{\text{ext}}(s')$. Then, $s_\lambda = s_{\text{ext}}(s)$, that is, $s \simeq_{\text{ext}} s_\lambda$.

- Case $\rightarrow_{\bullet 2}$. Identical to the previous one.
- Case \rightarrow_{var} .

$$\begin{array}{c} \frac{\frac{\vdash x : \mathbb{A}\langle \star \uparrow \rangle_i (= A_i)}{\vdots}^i}{s' = \vdash \lambda x. C\langle x \rangle : \mathbb{L}\langle A_i \rangle \rightarrow A'} \quad \rightarrow_{\text{var}} \quad \frac{\frac{\vdash x : A_i}{\vdots}^i}{\vdash \lambda x. C\langle x \rangle : \mathbb{L}\langle \mathbb{A}\langle \star \downarrow \rangle_i \rangle \rightarrow A'} = s \\ \simeq_{\text{ext}} \\ s_{\text{ext}}(s') = (\underbrace{\mathbf{x}, C\langle \lambda x. D_n \rangle}_{= C_{s'}}, \underbrace{L_n \cdot L}_{= L_{\text{ext}}(s')}, T_{\text{ext}}(s')) \quad \rightarrow_{\text{var}} \quad (\lambda x. D_n\langle x \rangle, \mathbf{C}, L, (x, \lambda x. D_n, L_n) \cdot T_{\text{ext}}(s')) = s_\lambda \end{array}$$

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 C.9 there is a correspondence between the level of term contexts and the length of the extracted log, so that $L_{\text{ext}}(s')$ is at least of length n , that is, $L_{\text{ext}}(s') = L_n \cdot L$, and $L_{\text{ext}}(s) = L$.

About the tape, note that $T_{\text{ext}}(s) = l_{\text{ext}}(s_f^1) \cdot T_{\text{ext}}^s(\mathbb{A}, 1)$ where s_f^1 is the first type test of s . To show that $s_{\text{ext}}(s) = (x, \lambda x. D_n, L_n) \cdot T_{\text{ext}}(s')$ we have to show two things:

- 1) $l_{\text{ext}}(s_f^1) = (x, \lambda x. D_n, L_n)$. Note that s_f^1 is $\vdash \lambda x. C\langle x \rangle : \mathbb{A}\langle \star \uparrow \rangle_i, \mathbb{L} \rightarrow A'$. Note that $s_f^1 \rightarrow_{\text{bt2}} \vdash x : \mathbb{A}\langle \star \downarrow \rangle_i, \langle \cdot \rangle = s''$, where s'' focusses on the same judgement of s' , and that s'' is the state that T-exhausts s_f^1 . By definition of extraction, $l_{\text{ext}}(s_f^1) = (x, \lambda x. D_n, L_n)$.
- 2) $T_{\text{ext}}^s(\mathbb{A}, 1) = T_{\text{ext}}(s')$, that is, $T_{\text{ext}}^s(\mathbb{A}, 1) = T_{\text{ext}}^{s'}(\mathbb{A}, 0)$. Note that $T_{\text{ext}}^s(\mathbb{A}, 1)$ and $T_{\text{ext}}^{s'}(\mathbb{A}, 0)$ 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 \rightarrow_{bt2} case of the proof of the T-exhaustible invariant (Lemma C.6), 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 $T_{\text{ext}}^s(\mathbb{A}, 1) = T_{\text{ext}}(s')$.

Then $s_{\text{ext}}(s) = (x, \lambda x. D_n, L_n) \cdot T_{\text{ext}}(s')$, and so $s_\lambda = s_{\text{ext}}(s)$, that is, $s \simeq_{\text{ext}} s_\lambda$.

- Case $\rightarrow_{\text{bt}2}$.

$$\begin{array}{ccc}
\frac{\overline{\vdash x : A_i (= \mathbb{A}\langle \star \rangle_i)}^i}{\vdots} & & \frac{\overline{\vdash x : \mathbb{A}\langle \star \downarrow \rangle_i}^i}{\vdots} \\
s' = \frac{\overline{\lambda x. C\langle x \rangle : \mathbb{L}\langle \mathbb{A}\langle \star \uparrow \rangle_i} \rightarrow A'}{\simeq_{\text{ext}}} & \rightarrow_{\text{bt}2} & \frac{\overline{\vdash \lambda x. C\langle x \rangle : \mathbb{L}\langle A_i \rangle} \rightarrow A' = s}{\simeq_{\text{ext}}} \\
s_{\text{ext}}(s') = (\lambda x. D_n\langle x \rangle, C_{s'}, L_{\text{ext}}(s'), \underbrace{(x, \lambda x. D_n, L_n) \cdot T_{\text{ext}}^{s'}(\mathbb{A}, 1)}_{=T_{\text{ext}}(s')}) & \rightarrow_{\text{bt}2} & (x, C_{s'}\langle \lambda x. D_n \rangle, L_n \cdot L_{\text{ext}}(s'), T_{\text{ext}}^{s'}(\mathbb{A}, 1)) = s'_{\lambda}
\end{array}$$

About the tape of $s_{\text{ext}}(s')$, note that $T_{\text{ext}}(s') = l_{\text{ext}}(s_f^1) \cdot T_{\text{ext}}^{s'}(\mathbb{A}, 1)$ where s_f^1 is the first type test of s' . We have to show that s_f^1 exhausts on x , so that $l_{\text{ext}}(s_f^1) = (x, \lambda x. D_n, L_n)$ for some L_n . Note that s_f^1 is $\vdash \lambda x. C\langle x \rangle : \mathbb{A}\langle \star \rangle_{i\uparrow}, \mathbb{L} \rightarrow A'$. Note that $s_f^1 \rightarrow_{\text{bt}2} \vdash x : \mathbb{A}\langle \star \rangle_{i\downarrow}, \langle \cdot \rangle = s''$, where s'' focusses on the same judgement of s , and that s'' is the state that S-exhausts s_f^1 . By definition of extraction, $l_{\text{ext}}(s_f^1) = (x, \lambda x. D_n, L_n)$ where L_n is the extraction of the first n judgement tests of s . Then $C_s = C_{s'}\langle \lambda x. D_n \rangle$ and $L_{\text{ext}}(s) = L_n \cdot L_{\text{ext}}(s')$.

About the tape, for s we have to prove that $T_{\text{ext}}^s(\mathbb{A}, 1) = T_{\text{ext}}(s) = T_{\text{ext}}^s(\mathbb{A}, 0)$. This is done as for \rightarrow_{var} , mimicking the reasoning in the proof of the T-exhaustible invariant (Lemma C.6).

Then, $s_{\lambda} = s_{\text{ext}}(s)$, that is, $s \simeq_{\text{ext}} s_{\lambda}$.

- Cases $\rightarrow_{\bullet 3}$ and $\rightarrow_{\bullet 4}$. They are identical to case $\rightarrow_{\bullet 1}$.
- Case \rightarrow_{arg} .

$$\begin{array}{ccc}
\frac{\vdash t : \mathbb{L}\langle \mathbb{A}\langle \star \downarrow \rangle_i \rangle \rightarrow A \quad \frac{\dots \vdash u : \mathbb{A}\langle \star \rangle_i \dots}{\vdash u : T (= \mathbb{L}\langle \mathbb{A}\langle \star \rangle_i \rangle)} \text{T-MANY}}{\vdash tu : A} & \rightarrow_{\text{arg}} & \frac{\vdash t : T \rightarrow A \quad \frac{\dots \vdash u : \mathbb{A}\langle \star \uparrow \rangle_i \dots}{\vdash u : \mathbb{L}\langle \mathbb{A}\langle \star \rangle_i \rangle} \text{T-MANY}}{\vdash tu : A} = s \\
\frac{\vdash tu : A}{\simeq_{\text{ext}}} & & \\
s_{\text{ext}}(s') = (t, \underbrace{D\langle \langle \cdot \rangle \rangle u}_{=C_{s'}}, L_{\text{ext}}(s'), \underbrace{l_{\text{ext}}(s_f^1) \cdot T_{\text{ext}}^{s'}(\mathbb{A}, 1)}_{=T_{\text{ext}}(s')}) & \rightarrow_{\text{arg}} & (\underline{u}, D\langle \langle \cdot \rangle \rangle, l_{\text{ext}}(s_f^1) \cdot L_{\text{ext}}(s'), T_{\text{ext}}^{s'}(\mathbb{A}, 1)) = s'_{\lambda}
\end{array}$$

where s_f^1 is the first type test of s' . Obviously, $C_s = D\langle \langle \cdot \rangle \rangle$. For the log we have to show that $L_{\text{ext}}(s)$ is equal to $l_{\text{ext}}(s_f^1) \cdot L_{\text{ext}}(s')$, which amounts to show that the first judgement test s^1 of s exhausts on the same state as the first tape test s_f^1 of s' . This is exactly the reasoning done in the proof of the T-exhaustible invariant. Similarly, one obtains that $T_{\text{ext}}^{s'}(\mathbb{A}, 1) = T_{\text{ext}}(s) = T_{\text{ext}}^s(\mathbb{A}, 0)$.

- Case $\rightarrow_{\text{bt}1}$.

$$\begin{array}{ccc}
\frac{\vdash t : T \rightarrow A \quad \frac{\dots \vdash u : \mathbb{A}\langle \star \downarrow \rangle_i \dots}{\vdash u : \mathbb{L}\langle \mathbb{A}\langle \star \rangle_i \rangle (= T)} \text{T-MANY}}{\vdash tu : A} & \rightarrow_{\text{bt}1} & \frac{\vdash t : \mathbb{L}\langle \mathbb{A}\langle \star \uparrow \rangle_i \rangle \rightarrow A \quad \frac{\dots \vdash u : \mathbb{A}\langle \star \rangle_i \dots}{\vdash u : T} \text{T-MANY}}{\vdash tu : A} = s \\
\frac{\vdash tu : A}{\simeq_{\text{ext}}} & & \\
s_{\text{ext}}(s') = (u, \underbrace{D\langle \langle \cdot \rangle \rangle}_{=C_{s'}} u, \underbrace{l_{\text{ext}}(s_f^1) \cdot L}_{=L_{\text{ext}}(s')}, T_{\text{ext}}(s')) & \rightarrow_{\text{bt}1} & (\underline{t}, D\langle \langle \cdot \rangle \rangle u, L, l_{\text{ext}}(s_f^1) \cdot T_{\text{ext}}(s')) = s'_{\lambda}
\end{array}$$

where s_f^1 is the first judgement test of s' . Obviously, $C_s = D\langle \langle \cdot \rangle \rangle u$. For the log, there is nothing to prove. For the tape, we have to show that $T_{\text{ext}}(s)$ is equal to $l_{\text{ext}}(s_f^1) \cdot T_{\text{ext}}(s')$, which amounts to show two things. First, that the first tape test s^1 of s exhausts on the same state as the first judgement test s_f^1 of s' . Second, that $T_{\text{ext}}^s(\mathbb{A}, 1) = T_{\text{ext}}(s) = T_{\text{ext}}^{s'}(\mathbb{A}, 0)$. Both points follow exactly the reasoning done in the proof of the T-exhaustible invariant. \square

A. The TIAM is acyclic

First of all, we prove the abstract lemma that says that every state is reachable in a bi-deterministic transition system with only one initial state.

Lemma C.11. *Let \mathbb{T} be an acyclic bi-deterministic transition system on a finite set of states S and with only one initial state s_i . Then all states in 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. \square

In order to prove that the TIAM is acyclic, we need to show that if $t \rightarrow_{wh} u$, 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' . This idea has been already exploited in [5], where bisimulations called *improvements* are used to prove the correctness of the λ IAM, from which we now recall a few definitions.

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

Definition C.12 (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.*

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

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

Proposition C.13 ([5]). *Let \mathcal{R} be an improvement on two DTS \mathcal{S} and \mathcal{Q} , and $s \mathcal{R} q$.*

- 1) Termination equivalence: $s \in S_{\downarrow}$ if and only if $q \in Q_{\downarrow}$.
- 2) Improvement: $|s| \geq |q|$.

b) *Weak Head Contexts:* Next, we need the notion of weak head context H defined as:

$$H := \langle \cdot \rangle \mid Ht$$

Note that if $t \rightarrow_{wh} u$ then $t = H\langle(\lambda x.r)w\rangle$ and $u = H\langle r\{x \leftarrow w\}\rangle$.

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

d) *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 judgement of the derivation π , obtaining $\vdash (t, C) : \mathbb{A}\langle\star\rangle$.

Definition C.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 :

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

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

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

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

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

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

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

Please note that the only states of π which are not mapped to any state of π' are those relative to the judgment $\vdash \lambda x.r : [G'_1 \dots G'_n] \rightarrow A$.

Proposition C.15. \blacktriangleright *is an improvement between TIAM states.*

Proof. ⁸ We inspect the 4 cases of the definition of \blacktriangleright .

- Rule $\text{rdx} \vdash (H\langle(\lambda x.t)u\rangle, K) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\text{rdx}} \vdash (H\langle t\{x\leftarrow u\}\rangle, K) : \mathbb{A}\langle\star\rangle$. Cases for \uparrow (by cases of H):

– $H = \langle\cdot\rangle$. The diagram is closed by rule body :

$$\begin{array}{ccc} \vdash ((\lambda x.t)u, K) : \mathbb{A}\langle\star\rangle & \xrightarrow{\text{TIAM}} & \vdash (\lambda x.t, K\langle\langle\cdot\rangle u\rangle) : T \rightarrow \mathbb{A}\langle\star\rangle & \xrightarrow{\text{TIAM}} & \vdash (t, K\langle(\lambda x.\langle\cdot\rangle)u\rangle) : \mathbb{A}\langle\star\rangle \\ & \blacktriangleright_{\text{rdx}} & & & \blacktriangleright_{\text{body}} \\ \vdash (t\{x\leftarrow u\}, K) : \mathbb{A}\langle\star\rangle & = & & & \vdash (t\{x\leftarrow u\}, K) : \mathbb{A}\langle\star\rangle \end{array}$$

– $H = Gs$. The diagram is closed by rule $\blacktriangleright_{\text{rdx}}$:

$$\begin{array}{ccc} \vdash (G\langle r\rangle s, K) : \mathbb{A}\langle\star\rangle & \xrightarrow{\text{TIAM}} & \vdash (G\langle r\rangle, K\langle\langle\cdot\rangle s\rangle) : T \rightarrow \mathbb{A}\langle\star\rangle \\ & \blacktriangleright_{\text{rdx}} & \blacktriangleright_{\text{rdx}} \\ \vdash (G\langle w\rangle s, K) : \mathbb{A}\langle\star\rangle & \xrightarrow{\text{TIAM}} & \vdash (G\langle w\rangle, K\langle\langle\cdot\rangle s\rangle) : T \rightarrow \mathbb{A}\langle\star\rangle \end{array}$$

Cases for \downarrow (by cases of K):

– $K = \langle\cdot\rangle$. Both machines are stuck.

$$\begin{array}{c} \vdash (r, \langle\cdot\rangle) : \mathbb{A}\langle\star\rangle \\ \blacktriangleright_{\text{rdx}} \\ \vdash (w, \langle\cdot\rangle) : \mathbb{A}\langle\star\rangle \end{array}$$

– $K = G\langle\langle\cdot\rangle s\rangle$. Two subcases depending on the type context. If the focus is on the right of the arrow the diagram is closed by rule rdx .

$$\begin{array}{ccc} \vdash (r, G\langle\langle\cdot\rangle s\rangle) : T \rightarrow \mathbb{A}\langle\star\rangle & \xrightarrow{\text{TIAM}} & \vdash (rs, G) : \mathbb{A}\langle\star\rangle \\ & \blacktriangleright_{\text{rdx}} & \blacktriangleright_{\text{rdx}} \\ \vdash (w, G\langle\langle\cdot\rangle s\rangle) : T \rightarrow \mathbb{A}\langle\star\rangle & \xrightarrow{\text{TIAM}} & \vdash (ws, G) : \mathbb{A}\langle\star\rangle \end{array}$$

If the focus is on the left of the arrow the diagram is closed by rule ext .

$$\begin{array}{ccc} \vdash (r, G\langle\langle\cdot\rangle s\rangle) : \mathbb{L}\langle\mathbb{A}\langle\star\rangle\rangle \rightarrow A & \xrightarrow{\text{TIAM}} & \vdash (s, G\langle r\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle \\ & \blacktriangleright_{\text{rdx}} & \blacktriangleright_{\text{ext}} \\ \vdash (w, G\langle\langle\cdot\rangle s\rangle) : \mathbb{L}\langle\mathbb{A}\langle\star\rangle\rangle \rightarrow A & \xrightarrow{\text{TIAM}} & \vdash (s, G\langle w\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle \end{array}$$

- Rule $\text{body} \vdash (t, H\langle(\lambda x.D)u\rangle) : \mathbb{A}\langle\star\rangle \blacktriangleright_{\text{body}} \vdash (t\{x\leftarrow u\}, H\langle D\{x\leftarrow u\}\rangle) : \mathbb{A}\langle\star\rangle$. 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 $t : T \rightarrow A$, the diagram is trivially closed by rule body .

– $t = x$. Diagram closed by rule arg .

$$\begin{array}{ccc} \vdash (x, H\langle(\lambda x.D)u\rangle) : \mathbb{A}\langle\star\rangle & \xrightarrow{\text{TIAM}} & \vdash (\lambda x.D\langle x\rangle, H\langle\langle\cdot\rangle u\rangle) : \mathbb{L}\langle\mathbb{A}\langle\star\rangle\rangle \rightarrow A' & \xrightarrow{\text{TIAM}} & \vdash (u, H\langle(\lambda x.D\langle x\rangle)\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle \\ & \blacktriangleright_{\text{body}} & & & \blacktriangleright_{\text{arg}} \\ \vdash (u, H\langle D\{x\leftarrow u\}\rangle) : \mathbb{A}\langle\star\rangle & = & & & \vdash (u, H\langle D\{x\leftarrow u\}\rangle) : \mathbb{A}\langle\star\rangle \end{array}$$

Cases of \downarrow (by cases of D):

– $D = \langle\cdot\rangle$. The diagram is closed by rule rdx

$$\begin{array}{ccc} \vdash (t, H\langle(\lambda x.\langle\cdot\rangle)u\rangle) : \mathbb{A}\langle\star\rangle & \xrightarrow{\text{TIAM}} & \vdash (\lambda x.t, H\langle\langle\cdot\rangle u\rangle) : T \rightarrow \mathbb{A}\langle\star\rangle & \xrightarrow{\text{TIAM}} & \vdash ((\lambda x.t)u, H) : \mathbb{A}\langle\star\rangle \\ & \blacktriangleright_{\text{body}} & & & \blacktriangleright_{\text{rdx}} \\ \vdash (t\{x\leftarrow u\}, H) : \mathbb{A}\langle\star\rangle & = & & & \vdash (t\{x\leftarrow u\}, H) : \mathbb{A}\langle\star\rangle \end{array}$$

– $D = E\langle\lambda y.\langle\cdot\rangle\rangle$, $D = E\langle\langle\cdot\rangle r\rangle$ and $D = E\langle r\langle\cdot\rangle\rangle$. The diagram is trivially closed by rule body .

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

$$\begin{array}{ccc} \vdash (t, H\langle(\lambda x.D\langle x\rangle)\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle & \xrightarrow{\text{TIAM}} & \vdash (\lambda x.D\langle x\rangle, H\langle\langle\cdot\rangle t\rangle) : \mathbb{L}\langle\mathbb{A}\langle\star\rangle\rangle \rightarrow A' & \xrightarrow{\text{TIAM}} & \vdash (x, H\langle(\lambda x.D)t\rangle) : \mathbb{A}\langle\star\rangle \\ & \blacktriangleright_{\text{arg}} & & & \blacktriangleright_{\text{body}} \\ \vdash (t, H\langle D\{x\leftarrow t\}\rangle) : \mathbb{A}\langle\star\rangle & = & & & \vdash (t, H\langle D\{x\leftarrow t\}\rangle) : \mathbb{A}\langle\star\rangle \end{array}$$

⁸Also this proof requires colors.

- 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 $w := H\langle r\{x\leftarrow u\}\rangle$.

$$\begin{array}{ccc} \vdash (t, K\langle s\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle & \xrightarrow{\text{TIAM}} & \vdash (s, K\langle\langle\cdot\rangle t\rangle) : \mathbb{L}\langle\mathbb{A}\langle\star\rangle\rangle \rightarrow A \\ \blacktriangleright_{\text{ext}} & & \blacktriangleright_{\text{rdx}} \\ \vdash (t, K\langle w\langle\cdot\rangle\rangle) : \mathbb{A}\langle\star\rangle & \xrightarrow{\text{TIAM}} & \vdash (w, K\langle\langle\cdot\rangle t\rangle) : \mathbb{L}\langle\mathbb{A}\langle\star\rangle\rangle \rightarrow A \end{array}$$

□

Corollary C.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 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 $\mathbb{T}_{\pi'}$. Since improvements preserve non-termination (Prop. C.13.1), also $s_{\pi'}$ diverges. Since $s_{\pi'}$ has a finite number of states, there must be a cycle. □

Corollary C.17. *For each type derivation $\pi \triangleright \vdash t : \star$, \mathbb{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 C.18. *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 C.11. □

APPENDIX D PROOFS FROM SECTION VII

Proposition D.1 (Space of Single Extracted States). *Let $s = (\pi, J, \mathbb{A}, d)$ be a reachable TIAM state. Then $|\mathbb{A}|_{\text{b}} = |T_{\text{ext}}(s)|_{\text{sp}}$ and $|J|_{\text{b}} = |L_{\text{ext}}(s)|_{\text{sp}}$, and thus $|s|_{\text{b}} = |\text{ext}(s)|_{\text{sp}}$. Moreover,*

- 1) *if $L_{\text{ext}}(s) = l_1..l_n$, 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 $|l_i|_{\text{sp}} = h_i$;*
- 2) *for each extracted tape position l , i.e. for each \mathbb{G} such that $\mathbb{A} = \mathbb{G}\langle\mathbb{L}\langle\mathbb{A}'\langle\star\rangle\rangle \rightarrow A$, then $|l|_{\text{sp}} = |\mathbb{L}| \cdot X$.*

Proof. We proceed by induction on the length of the run $\sigma : s_0 \xrightarrow{*}_{\text{TIAM}} s$. 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 $|\langle\cdot\rangle|_{\text{b}} = 0 = |T_{\text{ext}}(s)|_{\text{sp}}$ and $|J|_{\text{b}} = 0 = |L_{\text{ext}}(s)|_{\text{sp}}$. Otherwise, $\sigma : s_0 \xrightarrow{n}_{\text{TIAM}} s' \xrightarrow{1}_{\text{TIAM}} s$. We analyze the different cases of the last transition.

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

$$s' = \frac{\vdash t : T \rightarrow A \quad \vdash}{s' = \vdash tu : \mathbb{A}\langle\star_\uparrow\rangle (= A) \rightarrow_{\bullet 1}} \quad \frac{\vdash t : T \rightarrow \mathbb{A}\langle\star_\uparrow\rangle \quad \vdash}{\vdash tu : A} = s$$

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

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

$$s' = \frac{\overline{\vdash x : \mathbb{A}\langle\star_\uparrow\rangle_i (= A_i)}^i \quad \vdots}{\vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle A_i\rangle \rightarrow A'} \rightarrow_{\text{var}} \frac{\overline{\vdash x : A_i}^i \quad \vdots}{\vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle \mathbb{A}\langle\star_\downarrow\rangle_i\rangle \rightarrow A'} = s$$

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 $s_{\text{ext}}(s') = (\underline{x}, D\langle\lambda x.C\rangle, L_{\text{ext}}(i) \cdot L_{\text{ext}}(J), T_{\text{ext}}(\mathbb{A}_i))$ and $s_{\text{ext}}(s) = (\lambda x.C\langle x\rangle, \underline{D}, L_{\text{ext}}(J), (x, \lambda x.C, L_{\text{ext}}(i)) \cdot T_{\text{ext}}(\mathbb{A}_i))$. By *i.h.* we have $k \cdot X = |L_{\text{ext}}(i)|_{\text{sp}}$. Then $|\mathbb{L}\langle \mathbb{A}_i\rangle \rightarrow A'|_{\text{b}} = X + k \cdot X + |\mathbb{A}_i|_{\text{b}} = X + |L_{\text{ext}}(i)|_{\text{sp}} + |T_{\text{ext}}(s')|_{\text{sp}} = |x, \lambda x.C, L_{\text{ext}}(i)|_{\text{sp}} + |T_{\text{ext}}(s')|_{\text{sp}} = |T_{\text{ext}}(s)|_{\text{sp}}$. About the log, it suffices to note that $|J|_{\text{b}} =_{i.h.} |L_{\text{ext}}(s)|_{\text{sp}}$.

- Case $\rightarrow_{\text{bt}2}$.

$$s' = \frac{\overline{\vdash x : A_i (= \mathbb{A}\langle\star_\uparrow\rangle_i)}^i \quad \vdots}{\vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle \mathbb{A}\langle\star_\uparrow\rangle_i\rangle \rightarrow A'} \rightarrow_{\text{bt}2} \frac{\overline{\vdash x : \mathbb{A}\langle\star_\downarrow\rangle_i}^i \quad \vdots}{\vdash \lambda x.C\langle x\rangle : \mathbb{L}\langle A_i\rangle \rightarrow A'} = s$$

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 $s_{\text{ext}}(s') = (\underline{\lambda x.C\langle x\rangle}, D, L_{\text{ext}}(J), (x, \lambda x.C, L_{\text{ext}}(i)) \cdot T_{\text{ext}}(\mathbb{A}_i))$

and $s_{\text{ext}}(s) = (x, \underline{D}\langle \lambda x.C \rangle, L_{\text{ext}}(i) \cdot L_{\text{ext}}(J), T_{\text{ext}}(\mathbb{A}_i))$. $|\mathbb{A}_i|_{\text{b}} = |\mathbb{L}\langle \mathbb{A}\langle \star \rangle_i \rangle| \rightarrow A'_{\text{b}} - |\mathbb{L}| \cdot \mathbb{X} =_{i.h.} |T_{\text{ext}}(s')|_{\text{sp}} - |(x, \lambda x.C, L_{\text{ext}}(i))|_{\text{sp}} = |(x, \lambda x.C, L_{\text{ext}}(i)) \cdot T_{\text{ext}}(\mathbb{A}_i)|_{\text{sp}} - |(x, \lambda x.C, L_{\text{ext}}(i))|_{\text{sp}} = |T_{\text{ext}}(\mathbb{A}_i)|_{\text{sp}}$. About the log, since $|L_{\text{ext}}(i)|_{\text{sp}} = k \cdot \mathbb{X}$ by *i.h.* and $|J|_{\text{b}} =_{i.h.} |L_{\text{ext}}(J)|_{\text{sp}}$, then $|J|_{\text{b}} = |J|_{\text{b}} + k \cdot \mathbb{X} = |L_{\text{ext}}(J)|_{\text{sp}} + |L_{\text{ext}}(i)|_{\text{sp}} = |L_{\text{ext}}(s)|_{\text{sp}}$.

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

$$s' = \frac{\frac{\dots \vdash u : \mathbb{A}\langle \star \rangle_i \dots}{\vdash t : \mathbb{L}\langle \mathbb{A}\langle \star \rangle_i \rangle \rightarrow A} \quad \frac{\dots \vdash u : \mathbb{A}\langle \star \rangle_i \dots}{\vdash u : T(= \mathbb{L}\langle \mathbb{A}\langle \star \rangle_i \rangle)} \text{T-MANY}}{\vdash tu : A} \quad \rightarrow_{\text{arg}} \quad \frac{\frac{\dots \vdash u : \mathbb{A}\langle \star \rangle_i \dots}{\vdash t : T \rightarrow A} \quad \frac{\dots \vdash u : \mathbb{A}\langle \star \rangle_i \dots}{\vdash u : \mathbb{L}\langle \mathbb{A}\langle \star \rangle_i \rangle} \text{T-MANY}}{\vdash tu : A} = s$$

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

- Case \rightarrow_{bt1} . Equivalent to the previous one. □

Lemma D.2. *Let G be a type. Then*

$$\|G\| = \max_{\mathbb{G}|G=\mathbb{G}\langle \star \rangle} |\mathbb{G}|_{\text{b}}$$

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

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

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

$$\begin{aligned} \|G\| &= \|[G_1, \dots, G_n]\| = \mathbb{X} + \max_i \{\|G_i\|\} = \mathbb{X} + \max_i \left\{ \max_{\mathbb{G}_i|G_i=\mathbb{G}_i\langle \star \rangle} |\mathbb{G}_i|_{\text{b}} \right\} = \max_i \left\{ \mathbb{X} + \max_{\mathbb{G}_i|G_i=\mathbb{G}_i\langle \star \rangle} |\mathbb{G}_i|_{\text{b}} \right\} \\ &= \max_i \left\{ \{|\mathbb{G}_1 \cdots \mathbb{G}_i \cdots \mathbb{G}_n|_{\text{b}} | G_i = \mathbb{G}_i\langle \star \rangle\} \right\} = \max_{\mathbb{G}|G=\mathbb{G}\langle \star \rangle} |\mathbb{G}|_{\text{b}} \end{aligned}$$

□

Lemma D.3 (Weights bound extracted tapes). *Let $\pi : \Gamma \vdash^w t : G$ be a weighted derivation and \mathcal{J} be the set of all the judgments occurring in π . Then*

$$w \geq \max_{\Delta \vdash u : G' \in \mathcal{J}} \|G'\|$$

Proof. We proceed by induction on π .

- Case T-VAR. This case is trivial.

$$\frac{}{x : [A] \vdash x : A} \text{T-VAR}$$

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

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

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

$$\frac{\Gamma, x : T \vdash^v t : A}{\Gamma \vdash \lambda x.t : T \rightarrow A} \text{T-}\lambda$$

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

$$\frac{\Gamma \vdash t : T \rightarrow A \quad \Delta \vdash u : T}{\Gamma \uplus \Delta \quad \vdash \quad tu : A} \text{ T-@}$$

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

$$\frac{\Gamma_i \vdash t : G_i \quad 1 \leq i \leq n}{[\uplus_{i=1}^n \Gamma_i] \quad \vdash \quad t : [G_1, \dots, G_n]} \text{ T-MANY}$$

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

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

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

□

Lemma D.4 (Weights bound also extracted logs). *Let $\pi : \Gamma \vdash t : G$ be a weighted derivation. Then $w \geq v + |J|_b$ for every weighted judgement $J \vdash^v$ in π .*

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

$$w \geq v' + |J'|_b \geq X + v + |J|_b - X = v + |J|_b.$$

□

Let $\text{states}(\pi)$ be the set of TIAM states during the execution of π .

Theorem D.5 (λ IAM space bounds). *Let $\pi \triangleright \vdash^w t : \star$ be a weighted tree types derivation. Then $|\text{ext}(s)|_{\text{sp}} \leq w$ for every $s \in \text{states}(\pi)$.*

Proof. We prove the bound using $|s|_b$ instead of $|\text{ext}(s)|_{\text{sp}}$, and obtain the statement because $|s|_b$ instead of $|\text{ext}(s)|_{\text{sp}}$ by Prop. VII.3. Let $s = (\pi, J, \mathbb{A}, d) \in \text{states}(\pi)$ be a TIAM state and let v be its weight. By Lemma VII.6, $w \geq |J|_b + v$. By Lemma VII.5, $v \geq \|\mathbb{A}(\star)\|$, and by Lemma VII.4 $\|\mathbb{A}(\star)\| \geq |\mathbb{A}|_b$. Then $v \geq |\mathbb{A}|_b$. Therefore, $w \geq |J|_b + |\mathbb{A}|_b = |s|_b$. □

Proposition D.6 (Weight witness). *Let $\pi : \Gamma \vdash^w t : G$ be a weighted derivation and $G \neq [\cdot]$. Then there exists a TIAM state s over π such that $w = |s|_b$.*

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

- Case T-VAR:

$$\frac{}{x : [A] \vdash x : A} \text{ T-VAR}$$

There is no log and thus $s_L = 0$, and by Lemma VII.4, $\|A\| = \max_{\mathbb{A}|A=\mathbb{A}(\star)} |\mathbb{A}|_b$.

- Case T- λ_\star :

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

There is no log and thus $s_L = 0$, and $\|\star\| = 0 = |\langle \cdot \rangle|_b$.

- Case T- λ :

$$\frac{\Gamma, x : T \vdash t : A}{\Gamma \quad \vdash \quad \lambda x.t : T \rightarrow A} \text{ T-}\lambda$$

There are two sub-cases:

- 1) $w = v \geq \|T \rightarrow A\|$: then the statement follows by the *i.h.*

- 2) $w = \|T \rightarrow A\| > v$, by Lemma VII.4, there is a state $s = (\pi, J, \mathbb{A}, d)$ over the concluding judgement $J = \Gamma \vdash \lambda x.t : T \rightarrow A$ for which $\|T \rightarrow A\| = |\mathbb{A}|_b$. Since for the concluding judgement $|J|_b = 0$, we obtain

$$w = \|T \rightarrow A\| = |\mathbb{A}|_b = |\mathbb{A}|_b + |J|_b = |s|_b.$$

- Case T-@:

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

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

- Case T-MANY:

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

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 $|\mathbb{G}|_b + |J|_b = m$. 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 $|\mathbb{G}|_b + |J|_b = \mathbb{X} + m = \mathbb{X} + \max_i \{v_i\}$.

- Case T-NONE:

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

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

Corollary D.7 (λ IAM exact bound via tree types derivations). *Let $\pi \triangleright \vdash^w t : \star$ be a tree types derivation and σ the complete λ IAM run on t . Then $|\sigma|_s = w$.*

Proof. By Theorem D.5, $|\sigma|_s \leq w$. By Prop. VII.8, there exists a state s of the TIAM over π such that $|s|_b = w$. By Prop. VII.3, $|\text{ext}(s)|_{\text{sp}} = |s|_b$. Therefore, $|\sigma|_s = w$. □

APPENDIX E PROOFS FROM SECTION VIII

We give the type schema for the Turing's fixed point combinator Θ , needed for the encoding of Turing machines, defined as follows:

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

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.

Consider a list of types $\bar{A} := A_k, \dots, A_0$, type 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:

$$\Theta : \mathbb{F}_0^{\bar{A}} := \mathbb{T}_0^{\bar{A}} \rightarrow A_0 \quad \text{where} \quad \mathbb{T}_0^{\bar{A}} := [\mathbb{Y}_0^{\bar{A}}]$$

$$\text{and} \quad \mathbb{Y}_0^{\bar{A}} := [\cdot] \rightarrow A_0$$

Considering that $\Theta t \rightarrow_{wh} (\lambda y.y(\Theta y))t \rightarrow_{wh} t(\Theta t)$, $y : \mathbb{T}_0^{\bar{A}}$ and $t : \mathbb{T}_0^{\bar{A}}$. 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:

$$\Theta : \mathbb{F}_{n+1}^{\bar{A}} = \mathbb{T}_{n+1}^{\bar{A}} \rightarrow A_{n+1}$$

$$\text{where} \quad \mathbb{T}_{n+1}^{\bar{A}} = [\mathbb{Y}_{n+1}^{\bar{A}}, [\mathbb{T}_n^{\bar{A}}]]$$

$$\text{and} \quad \mathbb{Y}_{n+1}^{\bar{A}} = [A_n] \rightarrow A_{n+1}$$

Again, we note that the $n+2$ leaves of $\mathbb{T}_{n+1}^{\bar{A}}$ correspond to the fact that $t : \mathbb{T}_{n+1}^{\bar{A}}$ is evaluated $n+2$ times. Moreover, notice that \mathbb{T}_n , seen as a tree type, is sparse, having a topology identical to the one in Figure 6b. Moreover, tree types can be used to derive the space consumption of the λ IAM when used to evaluate Θ .

Lemma E.1. *For each $n \geq 0$, and for each list of types \bar{A} such that $|\bar{A}| \geq n+1$, $[\mathbb{T}_n^{\bar{A}}] \vdash^{ \geq (2n+1)\mathbb{X} } y : \mathbb{T}_n^{\bar{A}}$.*

Case $n + 1$:

$$\begin{array}{c}
\frac{\frac{\frac{}{x : [A] \vdash^0 x : A} \text{T-VAR}}{x : [[A] \rightarrow A] \vdash^X x : [A]} \text{T-VAR}}{x : [[A] \rightarrow A, [A]] \vdash^X xx : A} \text{T-}\lambda}{\vdash^{2X} \lambda x.xx : [[A] \rightarrow A, [A]] \rightarrow A} \text{T-}\lambda \\
\frac{\frac{\frac{\frac{\frac{}{x : [A] \vdash^0 x : A} \text{T-VAR}}{x : [[A]] \vdash^X x : [A]} \text{T-M}}{x : [[A] \rightarrow A, [A]] \vdash^X xx : A} \text{T-}\lambda}}{\vdash^{2(n+1)X} t_{n+1} := (\lambda x.xx)t_n : A =: [A'] \rightarrow A'} \text{T-}\@}{\frac{\frac{i.h.}{\vdash^{2nX} t_n : A} \text{T-M}}{\vdash^{2nX} t_n : [A] \rightarrow A} \text{T-}\@} \text{T-}\@} \text{T-M}
\end{array}$$

Since $X = \log(|t_n|) = \log(n)$, we have that the space consumption of the λ IAM is $\Theta(n \log n)$, *i.e.* quasi-logarithmic in $\#\beta$.