



HAL
open science

The (In)Efficiency of interaction

Beniamino Accattoli, Ugo Dal Lago, Gabriele Vanoni

► **To cite this version:**

Beniamino Accattoli, Ugo Dal Lago, Gabriele Vanoni. The (In)Efficiency of interaction. Proceedings of the ACM on Programming Languages, 2021, 5 (POPL), pp.1-33. 10.1145/3434332 . hal-03346750

HAL Id: hal-03346750

<https://inria.hal.science/hal-03346750>

Submitted on 17 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 (In)Efficiency of Interaction

BENIAMINO ACCATTOLI, Inria & LIX, École Polytechnique, UMR 7161, France

UGO DAL LAGO, Università di Bologna & INRIA, Italy

GABRIELE VANONI, Università di Bologna & INRIA, Italy

Evaluating higher-order functional programs through abstract machines inspired by the geometry of the interaction is known to induce *space* efficiencies, the price being *time* performances often poorer than those obtainable with traditional, environment-based, abstract machines. Although families of lambda-terms for which the former is exponentially less efficient than the latter do exist, it is currently unknown how *general* this phenomenon is, and how far the inefficiencies can go, in the worst case. We answer these questions formulating four different well-known abstract machines inside a common definitional framework, this way being able to give sharp results about the relative time efficiencies. We also prove that non-idempotent intersection type theories are able to precisely reflect the time performances of the interactive abstract machine, this way showing that its time-inefficiency ultimately descends from the presence of higher-order types.

Additional Key Words and Phrases: lambda-calculus, abstract machines, geometry of interaction

1 INTRODUCTION

Sometimes, simple objects such as natural numbers can generate theories of marvelous richness, such as number theory. Something similar happens with the λ -calculus, the universally accepted model of purely functional programs. Its definition is simple: three constructors and just one rewriting rule, β -reduction, based on a natural notion of substitution. The theory of β -reduction, however, is surprisingly rich, and still the object of research, despite decades of deep investigations.

In the eighties, Barendregt's book [Barendregt 1984] presented a stable operational and denotational theory, Lévy had already developed his sophisticated optimality theory [Lévy 1978], and languages such as Haskell were using tricky sharing mechanisms in their implementations. In 1987, however, the linear logic [Girard 1987] earthquake came together with a completely new viewpoint on the λ -calculus, requiring to revisit the whole theory. For our story, two of its byproducts are relevant, namely the geometry of interaction [Girard 1989] (shortened to GoI) and game semantics [Abramsky et al. 2000; Hyland and Ong 2000].

GoI and Game Semantics. At the time, GoI was a radically new interpretation of proofs, arising from connections between linear logic and functional analysis, and based on an abstract notion of interactive execution for proofs. Game semantics was introduced to solve the *full abstraction problem for PCF* [Milner 1977], and along the years affirmed itself as the sharpest and most flexible form of semantics for higher-order languages. Roughly, the models known at the time were not able to capture fine computational behaviors—that is, they were not *intensional* enough. Strategies from game semantics, instead, allow to faithfully model these behaviors of λ -terms: program composition is modeled as the interplay between the corresponding strategies—a concrete form of interaction—having the flavor of executions in some sort of abstract machine. In fact, there are two styles of game semantics. One, AJM games, is due to Abramsky, Jagadeesan, and Malacaria [Abramsky et al. 2000], and it is directly inspired by GoI. Another one, HO games, is due to Hyland and Ong [Hyland and Ong 2000], and models interaction in a different, pointer-based, way.

Authors' addresses: Beniamino Accattoli, LIX, Inria & LIX, École Polytechnique, UMR 7161, France, beniamino.accattoli@inria.fr; Ugo Dal Lago, Università di Bologna & INRIA, Italy, ugo.dallago@unibo.it; Gabriele Vanoni, Università di Bologna & INRIA, Italy, gabriele.vanoni2@unibo.it.

2018. 2475-1421/2018/1-ART1 \$15.00

<https://doi.org/>

Game Machines. The computational content of GoI was first explored by [Danos and Regnier \[1995\]](#) and [Mackie \[1995\]](#), who proposed a new form of implementation schema called *interaction abstract machine* (shortened to IAM). The IAM works in a fundamentally different way with respect to environment-based abstract machines, which are the standard and time-efficient way of modeling the implementations of functional languages. The link between game semantics and abstract machines was first explored by [Danos et al. \[1996\]](#). They showed the IAM to be the machinery behind AJM games, and the new *pointer abstract machine* (PAM) the one behind HO games. They also established a correspondence between the two styles of games, providing an indirect relationship between the IAM and the PAM. Finally, from a technical study of the IAM, [Danos and Regnier \[1999\]](#) introduced an optimized machine, the *jumping abstract machine* (JAM), claiming it isomorphic to the PAM despite using very different data structures. In the following, we refer collectively to the IAM, the JAM, and the PAM, as to *game machines* (*interaction machines* would be ambiguous, because of the IAM).

A Blind Spot. Despite the existence of a huge literature about GoI and game semantics, their related abstract machines remain—somewhat surprisingly—not well understood. Game machines are quite sophisticated and their presentations are hard to grasp, sometimes even far from being formally defined. For instance, the PAM has always been presented informally, as an algorithm described in natural language or pseudocode. Additionally, the relationship between these machines is not clear, especially at the level of the relative performances. One of the aims of this paper is taking the first steps towards a proper theory of the efficiency of game machines.

Space and Interactions. It is well known that environment machines can be space inefficient, because the environment (or closure) mechanism they rely on uses space proportional to the number of β -steps, *i.e.* the natural time cost model of the λ -calculus. Using as much space as time is in fact the worst one can do, from a space efficiency point of view. The IAM relies on a different mechanism, that—similarly to offline Turing machines [[Dal Lago and Schöpp 2010](#)]¹—sacrifices time in order to be space-efficient. This phenomenon was first pointed out by [Mackie \[1995\]](#), but it is the extensive work by Schöpp and coauthors [[Dal Lago and Schöpp 2010](#); [Schopp 2007](#); [Schöpp 2014, 2015](#)] that showed that the IAM allows for capturing sub-linear space computations¹, something impossible in environment machines. Along the same lines, one can mention the *Geometry of Synthesis* [[Ghica 2007](#); [Ghica and Smith 2010](#)], in which the geometry of interaction is seen as a compilation scheme towards circuits, and computation space is *finite*, and of paramount importance.

Time and Interactions. About time, instead, not much is known for game machines. Since the early papers on the IAM [[Danos and Regnier 1995](#); [Mackie 1995](#)], it is known that it can be exponentially slower than environment machines. As an example, the family of terms t_n defined as $t_1 := I$ and $t_{n+1} := t_n I$ (where I is the identity combinator) takes time exponential in n to be evaluated by the IAM, but only linear time in any environment machine. Therefore, game and environment machines are fundamentally different devices, and game ones—at least the IAM—*can* be time-inefficient. There remain however various open questions. Is the inefficiency of the IAM a *general* phenomenon, that is, are *all* λ -terms concerned? What about the other game machines? How bad can the aforementioned phenomenon be, quantitatively speaking? On *which* λ -terms does the phenomenon show up? The main objective of this paper is to provide answers to these questions.

¹evaluating a λ -term without fully inspecting it is indeed possible if the term is accessed by way of pointers, in the spirit of so-called offline Turing machines (themselves an essential ingredient in the definition of complexity classes such as LOGSPACE), and this is precisely the way the IAM works; see [[Dal Lago and Schöpp 2016](#)] for a thorough discussion about sub-linear space computations in the λ -calculus.

Jumping is Dizzying. The time inefficiency of the IAM is addressed by Danos and Regnier and Mackie via an optimized machine, the JAM [Danos and Regnier 1999; Mackie 1995]. In which relation the JAM is with other machines is unclear. Danos and Regnier present the JAM as an optimization of the IAM defined on top of proof nets. Then, they claim (without proving it) that if one considers the call-by-name translation of the λ -calculus into proof nets, the JAM is isomorphic to the PAM, while they prove that using the call-by-value translation one obtains the KAM. This is somehow puzzling, since the KAM is a call-by-name machine.

Time, Environments, and Types. Another natural question comes from the study of the relationship between intersection types and environment machines. The non-idempotent variant of intersection types—here shortened to *multi types*—provides a type theoretic understanding of time for environment machines, as shown by de Carvalho [2018], since the execution time of environment machines can be extracted from multi types derivations. It is natural to wonder whether similar connections exist between game machines and multi types, or some other form of type system. That would be particularly useful as a way of comparing the time behaviour of a given term when evaluated by distinct machines.

This Paper. The aim of this work is precisely giving the first sharp results about the time (in)efficiency of the interaction mechanism at work in game machines. We adopt the simplest possible setting, that is, weak call-by-name evaluation on closed terms, called here *Closed CbN*, and we provide four main contributions.

Contribution 1: a Formal Common Framework. Inspired by a very recent reformulation of the IAM on λ -terms (rather than proof nets, as in the original papers) by Accattoli et al. [2020a], called λ IAM, we provide new similar presentations of the JAM and the PAM, called λ JAM and λ PAM. These formulations are easily manageable and comparable, enabling neat formal results about them—in particular, ours is the first formal and manageable definition of the PAM.

Contribution 2: Comparative Complexity. We provide bisimulations between the λ IAM, the λ JAM, the λ PAM, and additionally the KAM, taken as the reference for environment machines. This allows for a precise comparison of the time behavior of the four machines:

- (1) *Hierarchy:* we show that the KAM is never slower than the λ JAM which is never slower than the λ IAM, establishing a sort of hierarchy.
- (2) *λ JAM is (slowly) reasonable:* a close inspection shows that the λ JAM is at most quadratically slower than the KAM. Since the KAM is a *reasonable*² implementation scheme, we obtain that the λ JAM is reasonable as well.
- (3) *λ JAM and λ PAM isomorphism:* we confirm Danos and Regnier’s claim that the λ JAM and the λ PAM are isomorphic (and have the same time behavior), working out the elegant and yet far from trivial isomorphism³.

Contribution 3: λ IAM Time and Multi Types. We show how to extract the length of the λ IAM run on a term t —that is, the time cost—from multi type derivations for t . This study complements de Carvalho’s, showing that multi types can measure also the time of the IAM, not just the KAM. A key point is that, by comparing how multi types measure game and environment machines,

²*Reasonable* is a technical word meaning polynomially related to Turing machines. In our context, a machine for Closed CbN is reasonable if the number of transition it takes on t is polynomial in the number of weak head β -steps to reduce t and in the size $|t|$ of t . For more details about reasonable cost models for the λ -calculus, see the overview in [Accattoli 2018].

³In the note [Danos and Regnier 2004], the authors claim that the PAM “is faster than the KAM in many cases” referring to private communication with Herbelin. Our results falsify the claim, as the PAM—behaving as the λ JAM—is never faster and at most quadratically slower than the KAM.

we obtain a clear insight about the time gap between the two approaches: the time usage of the λ IAM depends on the multi type derivation *and* the size of the involved multi types, while the time usage of the KAM depends only on the former. Therefore, the gap is bigger on terms whose KAM evaluation is much shorter than the involved types⁴.

Contribution 4: a Uniform Proof Technique. The proofs of the three most challenging theorems—namely the bisimulations of the λ IAM and the λ JAM, of the λ JAM and the KAM, and the correctness of cost analyses via the type system—are all proved using the same novel technique. To prove the correctness of the λ IAM, Accattoli et al. [2020a] study a new invariant, the *exhaustible (state) invariant*, expressing a form of coherence of the data structures used by the λ IAM. Our theorems are all proved by adapting the exhaustible invariant to each specific case, providing concise technical developments and conceptual unity. This point contrasts strikingly with the original papers on game machines, whose proof techniques are involved and indirect⁵, and often informal. The exhaustible invariant—while certainly technical—is simpler and provides direct arguments. It seems to be the key tool to study game machines. As a slogan, *interacting is exhausting*.

Our Two Cents about Space. At the end of the paper we also briefly discuss space. Specifically, we provide examples showing that the λ IAM can use more space than the λ JAM, despite the former being considered space-efficient and the latter being as space-inefficient as possible. This fact does not contradict the space efficiency of the λ IAM, as it concerns only specific terms. Our example however shows that space relationships between game machines—if they can be established at all—are subtler than the time ones, and of a less uniform nature.

Our Results, at a Distance. The body of the paper is quite technical and this is inevitable, because abstract machines—for as much as they can be abstract—are low-level tools. It is however easy to provide a high-level perspective. Comparing with the original papers on game machines, our presentations play the role of a *Rosetta stone*, allowing to connect concepts and decode many technical subtleties and invariants. Our exhaustible invariant, additionally, removes the need to resort to game semantics or legal paths when relating the machines. Our complexity study suggests that interaction as modeled by HO games (seen as the λ JAM and the λ PAM) is a time-reasonable process, while as modeled by the GoI and AJM games (seen as the λ IAM) is a time-inefficient process⁶. Our multi type study, however, suggests that the gap between the two is big only on terms whose type derivations are much smaller than the involved types. Focusing on HO games, the quadratic overhead of the λ JAM with respect to the KAM shows that interaction as modeled by HO games is time-reasonable but not time-efficient. Summing up, *interacting takes time, and is exhausting*.

Evaluating without Duplicating. Let us provide a conclusive insight. β -reducing a λ -term (potentially) duplicates arguments, whose different copies may be used differently, typically being applied to different further arguments. The machines in this paper never duplicate parts of the code⁷, but have nonetheless to distinguish different uses of a same piece of code during execution. Each one

⁴Note that even the smallest multi type derivation for the inefficient IAM family t_n given above uses types exponential in n (inside the derivation).

⁵The relationship between the IAM and the PAM in [Danos et al. 1996] is not direct as it goes through both AJM games and HO games. Similarly, the relationship between the IAM and the JAM in [Danos and Regnier 1999] is not self-contained, as it is based on the non-trivial equivalence between regular and legal paths proved in [Asperti et al. 1994].

⁶Whether the IAM is reasonable is unclear. The aforementioned time inefficiency of the IAM is not a proof that it is unreasonable.

⁷Note that not all machines avoid duplications: machines with a single global environment duplicate pieces of the code, see [Accattoli and Barras 2017]. Perhaps surprisingly, performing duplications is not as costly as one may expect. Global environment machines are indeed time-efficient and faster than the game machines studied in this paper.

does it in a clever and sophisticated different way—multi types also fit this view, as they remove duplications altogether by taking all the needed copies in advance (see Sect. 14). This paper can then be seen as a systematic and thorough study of *the art of evaluating without duplicating*.

Related Work. Beyond the works already cited above, game machines are also studied by Curien and Herbelin [1998, 2007] who consider different machines as directly obtained by game semantics, Mackie [2017] who derives a proof net based token machine for System T, Pinto [2001] who develops a parallel implementation of the IAM, and Fernández and Mackie [2002] who extend token machines to call-by-value. A game machine accommodating the additive connectives of linear logic is in [Laurent 2001]. Space-efficiency of variants of the IAM is addressed by Mazza [2015]; Mazza and Terui [2015]. Game machines for languages beyond the λ -calculus, like λ -calculi with algebraic effects, quantum λ -calculi or concurrent calculi are in [Dal Lago et al. 2014, 2015, 2017; Hoshino et al. 2014]. A different kind of machine inspired by the GoI is in [Danos and Regnier 1993; Pedicini and Quaglia 2007]. Interaction and rewriting are mixed in recent work by Muroya and Ghica [2017, 2019]. Clairambault [2011, 2015] uses an abstraction of the PAM to bound evaluation lengths, and similar studies are also done by Aschieri [2017].

The space inefficiency of environment machines has already been observed by Krishnaswami, Benton, and Hoffman, who proposed some techniques to alleviate it in the context of functional-reactive programming and based on linear types [Krishnaswami et al. 2012].

The time efficiency of environment machines has been recently studied in depth. Before 2014, the topic had been mostly neglected—the only two counterexamples being Blelloch and Greiner [1995]; Sands et al. [2002]. Since 2014—motivated by advances on time cost models for the λ -calculus by Accattoli and Dal Lago [2016]—the topic has actively been studied [Accattoli et al. 2014; Accattoli and Barras 2017; Accattoli et al. 2019a; Accattoli and Guerrieri 2019].

Intersection types are a standard tool to study λ -calculi—see standard references such as [Coppo and Dezani-Ciancaglini 1978, 1980; Krivine 1993; Pottinger 1980]. Non-idempotent intersection types, *i.e.* multi types, were first considered by Gardner [1994], and then by de Carvalho [2007, 2018]; Kfoury [2000]; Neergaard and Mairson [2004]—a survey is [Bucciarelli et al. 2017]. De Carvalho’s use of multi types to give bounds to evaluation lengths has also been used in [Accattoli et al. 2020b; Accattoli and Guerrieri 2018; Accattoli et al. 2019b; Bernadet and Graham-Lengrand 2013; de Carvalho et al. 2011; Kesner and Vial 2020].

2 PRELIMINARIES: 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 (*i.e.* the number of !-boxes in linear logic terminology) the hole lies in.

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

We simply write C for a context whenever the level is not relevant. The operation replacing the hole $\langle \cdot \rangle$ with a term t in a context C is noted $C\langle t \rangle$ and called *plugging*.

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

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

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
tu	C	L	T	$\rightarrow_{\bullet 1}$	\underline{t}	$C\langle\langle \cdot \rangle u\rangle$	L	$\bullet \cdot T$
$\lambda x.t$	C	L	$\bullet \cdot T$	$\rightarrow_{\bullet 2}$	\underline{t}	$C\langle\lambda x.\langle \cdot \rangle\rangle$	L	T
\underline{x}	$C\langle\lambda x.D_n\rangle$	$L_n \cdot L$	T	\rightarrow_{var}	$\lambda x.D_n(x)$	\underline{C}	L	$(x, \lambda x.D_n, L_n) \cdot T$
$\lambda x.D_n(x)$	C	L	$(x, \lambda x.D_n, L_n) \cdot T$	\rightarrow_{bt2}	x	$\underline{C\langle\lambda x.D_n\rangle}$	$L_n \cdot L$	T
t	$\underline{C\langle\langle \cdot \rangle u\rangle}$	L	$\bullet \cdot T$	$\rightarrow_{\bullet 3}$	tu	\underline{C}	L	T
t	$\underline{C\langle\lambda x.\langle \cdot \rangle\rangle}$	L	T	$\rightarrow_{\bullet 4}$	$\lambda x.t$	\underline{C}	L	$\bullet \cdot T$
t	$\underline{C\langle\langle \cdot \rangle u\rangle}$	L	$l \cdot T$	\rightarrow_{arg}	\underline{u}	$C\langle t\langle \cdot \rangle\rangle$	$l \cdot L$	T
t	$\underline{C\langle u\langle \cdot \rangle\rangle}$	$l \cdot L$	T	\rightarrow_{bt1}	\underline{u}	$C\langle\langle \cdot \rangle t\rangle$	L	$l \cdot T$

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

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 machines considered in this paper move 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 $C\langle u \rangle = t$. The shape of states depends on the specific machine, but they always include 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. We may write s_t^M to stress the machine, and t is always implicitly considered closed, without further mention. A state is *final* if no transitions apply.

A *run* $\pi : s \rightarrow^* s'$ is a possibly empty sequence of transitions, whose length is noted $|\pi|$. If a and b are transitions labels (that is, $\rightarrow_a \subseteq \rightarrow$ and $\rightarrow_b \subseteq \rightarrow$) then $\rightarrow_{a,b} := \rightarrow_a \cup \rightarrow_b$, $|\pi|_a$ is the number of a transitions in π , and $|\pi|_{\neg a}$ is the size of transitions in π that are not \rightarrow_a . 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. Given a machine M , we write $M(t) \Downarrow$ if M reaches a final state starting from s_t^M , and $M(t) \Uparrow$ otherwise. We say that M *implements Closed CbN* when $M(t) \Downarrow$ if and only if \rightarrow_{wh} terminates on t , for every closed term t .

3 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 et al. [2020a]—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*, that is closer to the practice of functional programming and also the setting underlying the KAM. We first define the machine and then provide explanations and examples. Keep in mind that the λ IAM is an unusual machine, and that finding it hard to grasp is normal—probably, the next sections about the λ JAM and the KAM provide clarifying insights.

λ IAM States. 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—the transitions and all the data structures

are defined 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. A key point is that navigation is done locally, moving only between adjacent positions⁸. The λ IAM has also 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 induction⁹. 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. Intuitively, the machine evaluates the term t until the head abstraction of the head normal form is found (more explanations below). 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 $(\underline{t}, \underline{C}, L, T)$ are queries about the argument of an abstraction.

Next, we explain how the transitions realize three entangled mechanisms. Let us anticipate that the λ JAM shall be obtained by short-circuiting the third mechanism, backtracking, and the KAM by the further removal of the second one, that shall also require to modify the first one.

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 $\rightarrow_{\bullet 4}$ realize the same during the \uparrow phase. Let us illustrate this mechanism with an example (on the right): the first steps of evaluation on the term $(\lambda y. \lambda x. xy)ll$, where l is the identity combinator. One can notice that the λ IAM traverses two β -redexes without altering the token, that is empty both at the beginning and at the end.

	Sub-term	Context	Log	Tape	Dir
	$(\lambda y. \lambda x. xy)ll$	$\langle \cdot \rangle$	ϵ	ϵ	\downarrow
$\rightarrow_{\bullet 1}$	$(\lambda y. \lambda x. xy)l$	$\langle \cdot \rangle l$	ϵ	\bullet	\downarrow
$\rightarrow_{\bullet 1}$	$\lambda y. \lambda x. xy$	$\langle \cdot \rangle ll$	ϵ	$\bullet \bullet$	\downarrow
$\rightarrow_{\bullet 2}$	$\lambda x. xy$	$(\lambda y. \langle \cdot \rangle) ll$	ϵ	\bullet	\downarrow
$\rightarrow_{\bullet 2}$	xy	$(\lambda y. \lambda x. \langle \cdot \rangle) ll$	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 binder is recorded in a new logged position that is added to the tape. Since the machine moves out of a context of

⁸Note that also the transition from the variable occurrence to the binder in \rightarrow_{var} and \rightarrow_{bt2} are local 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 [Accattoli and Barras 2017].

⁹This is similar to the KAM, where closures and environments are defined by mutual induction, but logs and logged positions play a different role. Moreover, there also is a constraint about the length.

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 $\lambda x.D_n(x)$ 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 x and looks for its argument in \uparrow mode. When it has been found, the direction turns \downarrow again.

	Sub-term	Context	Log	Tape	Dir
	\underline{xy}	$(\lambda y.\lambda x.\langle \cdot \rangle)II$	ϵ	ϵ	\downarrow
$\rightarrow_{\bullet 1}$	\underline{x}	$(\lambda y.\lambda x.\langle \cdot \rangle y)II$	ϵ	\bullet	\downarrow
\rightarrow_{var}	$\lambda x.xy$	$(\lambda y.\langle \cdot \rangle)II$	ϵ	$(x, \lambda x.\langle \cdot \rangle y, \epsilon) \cdot \bullet$	\uparrow
$\rightarrow_{\bullet 4}$	$\lambda y.\lambda x.xy$	$\langle \cdot \rangle II$	ϵ	$\bullet \cdot (x, \lambda x.\langle \cdot \rangle y, \epsilon) \cdot \bullet$	\uparrow
$\rightarrow_{\bullet 3}$	$(\lambda y.\lambda x.xy)I$	$\langle \cdot \rangle I$	ϵ	$(x, \lambda x.\langle \cdot \rangle y, \epsilon) \cdot \bullet$	\uparrow
\rightarrow_{arg}	\downarrow	$(\lambda y.\lambda x.xy)I\langle \cdot \rangle$	$(x, \lambda x.\langle \cdot \rangle y, \epsilon)$	\bullet	\downarrow

Mechanism 3: Backtracking. It is started by transition $\rightarrow_{\text{bt}1}$. 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 $\rightarrow_{\text{bt}2}$ is fired.

The \downarrow -phase and the logged position on the tape mean that the λ IAM is backtracking. In fact, in this configuration the machine is not looking for the head variable of the current subterm $\lambda x.t$, 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 shall guarantee that the logged position on the tape always contains a position relative to the active abstraction. In our running example, a backtracking phase, noted with a BT label starts when the IAM looks for the argument of z . Since $\lambda z.z$ has been virtually substituted for x , its argument is actually y . Backtracking is needed to recover the variable a term was virtually substituted for.

	Sub-term	Context	Log	Tape	Dir
	$\underline{\lambda z.z}$	$(\lambda y.\lambda x.xy)I\langle \cdot \rangle$	$(x, \lambda x.\langle \cdot \rangle y, \epsilon)$	\bullet	\downarrow
$\rightarrow_{\bullet 2}$	\underline{z}	$(\lambda y.\lambda x.xy)I(\lambda z.\langle \cdot \rangle)$	$(x, \lambda x.\langle \cdot \rangle y, \epsilon)$	ϵ	\downarrow
\rightarrow_{var}	$\lambda z.z$	$(\lambda y.\lambda x.xy)I\langle \cdot \rangle$	$(x, \lambda x.\langle \cdot \rangle y, \epsilon)$	$(z, \lambda z.\langle \cdot \rangle, \epsilon)$	\uparrow
BT $\rightarrow_{\text{bt}1}$	$(\lambda y.\lambda x.xy)I$	$\langle \cdot \rangle I$	ϵ	$(x, \lambda x.\langle \cdot \rangle y, \epsilon) \cdot (z, \lambda z.\langle \cdot \rangle, \epsilon)$	\downarrow
BT $\rightarrow_{\bullet 1}$	$\underline{\lambda y.\lambda x.xy}$	$\langle \cdot \rangle II$	ϵ	$\bullet \cdot (x, \lambda x.\langle \cdot \rangle y, \epsilon) \cdot (z, \lambda z.\langle \cdot \rangle, \epsilon)$	\downarrow
BT $\rightarrow_{\bullet 2}$	$\underline{\lambda x.xy}$	$(\lambda y.\langle \cdot \rangle)II$	ϵ	$(x, \lambda x.\langle \cdot \rangle y, \epsilon) \cdot (z, \lambda z.\langle \cdot \rangle, \epsilon)$	\downarrow
$\rightarrow_{\text{bt}2}$	x	$(\lambda y.\lambda x.\langle \cdot \rangle y)II$	ϵ	$(z, \lambda z.\langle \cdot \rangle, \epsilon)$	\uparrow

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 first occurrence of l , is found.

	Sub-term	Context	Log	Tape	Dir
	x	$(\lambda y.\lambda x.\langle \cdot \rangle y)II$	ϵ	$(z, \lambda z.\langle \cdot \rangle, \epsilon)$	\uparrow
\rightarrow_{arg}	\underline{y}	$(\lambda y.\lambda x.x\langle \cdot \rangle)II$	$(z, \lambda z.\langle \cdot \rangle, \epsilon)$	ϵ	\downarrow
\rightarrow_{var}	$\lambda y.\lambda x.xy$	$\langle \cdot \rangle II$	ϵ	$(y, \lambda x.x\langle \cdot \rangle, (z, \lambda z.\langle \cdot \rangle, \epsilon))$	\uparrow
\rightarrow_{arg}	\downarrow	$(\lambda y.\lambda x.xy)\langle \cdot \rangle I$	$(y, \lambda x.x\langle \cdot \rangle, (z, \lambda z.\langle \cdot \rangle, \epsilon))$	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 , together with the position (t, C) , forms a logged position, *i.e.* the length of L is exactly the level of the code context C ¹⁰. This fact guarantees that the λ IAM never gets stuck because the log is not long enough 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 3.1 (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 and Interpretation. 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 Accattoli et al. [2020a]. The *tape and direction* invariant guarantees that the machine never stops because the log or the tape have not enough logged positions to apply a \rightarrow_{var} , \rightarrow_{bt1} , or a \rightarrow_{arg} transition. Additionally, on states such as $(\lambda x.D(x), C, L, l \cdot T)$, the logged position l has shape $(x, \lambda x.D, L')$, so that transition \rightarrow_{bt2} can always apply—this is a consequence of the *exhaustible state invariant* in Sect. 6, as shown in Accattoli et al. [2020a].

The exhaustible state invariant shall be the technical blueprint for the proof of the relationship between the λ IAM and the λ JAM, amounting to short-circuiting backtracking phases. Similarly, we shall use it to relate the KAM and the λ JAM, and the λ IAM with multi type derivations.

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 [Accattoli et al. 2020a]. For the sake of simplicity, here we restrict to Closed CbN. The λ IAM semantics then reduces to just 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.

THEOREM 3.2 ([ACCATTOLI ET AL. 2020A]). *The λ IAM implements Closed CbN.*

Cost of λ IAM Transitions. For all the abstract machines in this paper we take random access machines (shortened to RAM) with the uniform cost model as the computational model of reference. This is standard in the time analyses of abstract machines for functional languages. Roughly, it amounts to seeing variables and positions as objects whose manipulation takes constant time.

Every λ IAM transition can then be implemented on RAM in constant time but for transition \rightarrow_{var} , whose cost is bounded by the integer n given by D_n (referring to the notation of the rules), as the rule needs to split the log after the first n entries. This is in accordance with the proof nets interpretation of the λ IAM, because transitions \rightarrow_{var} correspond to *sequences* of IAM transitions

¹⁰Then, the length of L is exactly the number of (linear logic) *boxes* in which the code term is contained.

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

Fig. 2. Transitions of the λ Jumping Abstract Machine (λ JAM).

on proof nets—see [Accattoli et al. 2020a]¹¹. Note that n is bound by the size $|t|$ of the (immutable) initial code t . The cost of implementing on RAM a λ IAM run π from t then is $|\pi|_{\text{-var}} + |\pi|_{\text{var}} \cdot |t|$.

Two Useful Properties of the λ IAM. A key property is that the λ IAM is *bi-deterministic*, that is, it is deterministic and also deterministically reversible. Another more technical property is that it verifies a sort of context-freeness with respect to the tape T . Namely, extending the tape preserves the shape of the run and of the final state (up to the extension).

LEMMA 3.3 (λ IAM TAPE LIFT). *Let T be a tape and $\pi : s = (t, C, L, T', d) \rightarrow_{\lambda\text{IAM}}^n (u, D, L', T'', d') = s'$ a run. Then there is a λ IAM run $\pi^T : s^T = (t, C, L, T' \cdot T, d) \rightarrow_{\lambda\text{IAM}}^n (u, D, L', T'' \cdot T, d') = s'^T$.*

4 THE JUMPING ABSTRACT MACHINE, REVISITED

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

Jumping Around the Log. The difference between the λ IAM and the λ JAM is in how they create logged positions, and consequently on how they backtrack. The λ IAM has a

local approach to logs, and backtracks via potentially long sequences of transitions, while the λ JAM follows a *global* approach to logs, and it backtracks in just one *jump*. The transition system is presented in Fig. 2. The details of the two variations over the λ IAM are:

- *Global logged position:* logged positions created by rule \rightarrow_{var} are now global, in that they record the global position of the variable, and not only the position relative to its binder. This way, also the log has to be entirely copied. Differently from the λ IAM, there is some duplication of information.
- *Backtracking is short-circuited:* backtracking is a phase of a λ IAM run which is contained between \rightarrow_{bt1} and \rightarrow_{bt2} transitions. It starts when the machine has to rebuild the history of a redex/substitution and ends when the substituted variable occurrence l is found. The optimization at the heart of the λ JAM comes out from the observation that l is exactly the leftmost position on the log. This way, one could directly jump to that position, instead of doing the backtracking. Of course, this is possible only if positions are saved globally in the log, because the \rightarrow_{jmp} transitions is not local, but global.

The absence of the backtracking phase makes the λ JAM easier to understand than the λ IAM. In particular, the \downarrow and \uparrow phases have now a precise meaning: the former being the quest for the head

¹¹Actually, also transition \rightarrow_{bt2} corresponds to n IAM transitions on proof nets. By implementing logs as bi-linked lists, however, \rightarrow_{bt2} can be implemented in constant time. For \rightarrow_{var} instead, there is no easy way out.

variable of the current subterm, and the latter being the search of the argument of the *only* variable occurrence in the tape. This is the second point of the following lemma.

LEMMA 4.1 (λ JAM BASIC INVARIANTS). *Let $s = (t, C_n, L, T, d)$ be a reachable state. Then*

- (1) Position and log: (t, C_n, L) is a logged position, and
- (2) Tape and direction: if $d = \downarrow$, then T does not contain any logged position, otherwise, if $d = \uparrow$, then T contains exactly one logged position.

We present the λ JAM execution trace of the same term considered for the λ IAM. In particular, the first transitions are identical to the λ IAM execution since no \rightarrow_{var} and \rightarrow_{bt1} rules are involved. Instead, we observe that full context and log are saved at the occurrence of \rightarrow_{var} transitions. We put $l_x := (x, (\lambda y.\lambda x.\langle \cdot \rangle y)l(\lambda z.z), \epsilon)$.

Sub-term	Context	Log	Tape	Dir
$(\lambda y.\lambda x.xy)ll$	$\langle \cdot \rangle$	ϵ	ϵ	\downarrow
$\rightarrow_{\bullet 1} (\lambda y.\lambda x.xy)l$	$\langle \cdot \rangle l$	ϵ	\bullet	\downarrow
$\rightarrow_{\bullet 1} \lambda y.\lambda x.xy$	$\langle \cdot \rangle ll$	ϵ	$\bullet \bullet$	\downarrow
$\rightarrow_{\bullet 2} \lambda x.xy$	$(\lambda y.\langle \cdot \rangle)ll$	ϵ	\bullet	\downarrow
$\rightarrow_{\bullet 2} xy$	$(\lambda y.\lambda x.\langle \cdot \rangle)ll$	ϵ	ϵ	\downarrow
$\rightarrow_{\bullet 1} x$	$(\lambda y.\lambda x.\langle \cdot \rangle y)ll$	ϵ	\bullet	\downarrow

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

Finally, as already explained, backtracking is *jumped*: the λ JAM restores the previously encountered state, saved in the logged position, when exiting from the right-hand side of an application. We put $l_z := (z, (\lambda y.\lambda x.xy)l(\lambda z.\langle \cdot \rangle), l_x)$.

Sub-term	Context	Log	Tape	Dir
$\lambda z.z$	$(\lambda y.\lambda x.xy)l\langle \cdot \rangle$	l_x	l_z	\uparrow
$\rightarrow_{\text{jmp}} x$	$(\lambda y.\lambda x.\langle \cdot \rangle y)l(\lambda z.z)$	ϵ	l_z	\uparrow
$\rightarrow_{\text{arg}} y$	$(\lambda y.\lambda x.x\langle \cdot \rangle)ll$	l_z	ϵ	\downarrow
$\rightarrow_{\text{var}} \lambda y.\lambda x.xy$	$\langle \cdot \rangle ll$	ϵ	$(y, (\lambda y.\lambda x.x\langle \cdot \rangle)ll, l_z)$	\uparrow
$\rightarrow_{\text{arg}} !$	$(\lambda y.\lambda x.xy)\langle \cdot \rangle l$	$(y, (\lambda y.\lambda x.x\langle \cdot \rangle)ll, l_z)$	ϵ	\downarrow

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

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

5 KRIVINE ABSTRACT MACHINE

The Krivine abstract machine [Krivine 2007] (KAM) is a standard environment machine for Closed CbN whose time behavior has been studied thoroughly—in Sect. 10 we recall the literature about it. In particular, it is a time reasonable implementation of Closed CbN, where *reasonably* means

CLOSURES			ENVIRONMENTS		STACKS			STATES
$c ::= (t, C, E)$			$E ::= \epsilon \mid [x \leftarrow c] \cdot E$		$S ::= \epsilon \mid c \cdot S$			$s ::= (t, C, E, S)$
Term	Ctx	Env	Stack		Term	Ctx	Env	Stack
tu	C	E	S	\rightarrow_{app}	t	$C\langle\langle\cdot\rangle u\rangle$	E	$(u, C\langle\langle\cdot\rangle\rangle, E) \cdot S$
$\lambda x.t$	C	E	$c \cdot S$	\rightarrow_{abs}	t	$C\langle\lambda x.\langle\cdot\rangle\rangle$	$[x \leftarrow c] \cdot E$	S
x	C	$E \cdot [x \leftarrow (u, D, E'')] \cdot E'$	S	\rightarrow_{var}	u	D	E''	S

Fig. 3. Data structures and transitions of the Krivine Abstract Machine (KAM).

polynomially related to the time cost model of Turing machines. To be uniform with respect to the other machines, we present the KAM adding information about the context, which is not needed.

Hopping on Arguments. The KAM (in Fig. 3) differs from the λ IAM and λ JAM as it *does record* every β -redex that it encounters—thus explicitly entangling time and space consumption—using two data structures. Log and tape are replaced by the (*local*) *environment* E and the (*applicative*) *stack* S . The basic idea is that, by saving encountered β -redexes in the environment E , when the machine finds a variable occurrence x it simply looks up in E for the argument of the binder λx binding x , avoiding the \uparrow -phase of the λ JAM—note that the KAM has no \uparrow phase and no logs. Mimicking the *jump* terminology, one may say that KAM transition \rightarrow_{var} *hops* directly on the argument, skipping the search for it. The stack S is used to collect encountered arguments that still have to be paired to abstractions to form β -redexes, and then go into the environment E . The intuition is that the stack has an entry for every occurrences of \bullet on the tape of the λ JAM in the \downarrow -phase, but such entries are more informative, they actually record the encountered argument (and a copy of the environment, explained next), and not just acknowledge its presence.

Closures, Stacks, and Environments. The mutually recursive grammars for *closures* and *environments*, plus the independent one for *stacks* are defined in Fig. 3, together with the definition of states. The idea is that every piece of code comes with an environment, forming a closure, which is why environments and closures are mutually defined. Also, when the machine executes a closed term t , every closure (u, C, E) in a reachable state is such that for any free variable x of u there is an entry $[x \leftarrow c]$ in E every, thus E 'closes' u , whence the name *closures*.

Transitions, Initial and Final States. Initial states of the KAM are in the form $s_t = (t, \langle\cdot\rangle, \epsilon, \epsilon)$. The transitions of the KAM are in Fig. 3—their union is noted \rightarrow_{KAM} . The idea is that the \rightarrow_{var} transition looks in the environment for the argument of the variable under evaluation. As for the other machines, the KAM evaluates the term t until the top abstraction of the weak head normal form of t is found, that is a run either never stops or ends in a state s of the shape $s = (\lambda x.u, C, E, \epsilon)$. This is guaranteed by the mentioned and standard (but omitted) invariant ensuring that when the initial term is closed then every variable appearing in the code has an associated closure in the environment, so that the KAM never gets stuck on a \rightarrow_{var} transition. In the next section we shall prove that the λ JAM and the KAM are termination equivalent. We show the KAM execution trace of our running example. Initially, the KAM looks for the head variable keeping track of the encountered arguments. We put $Q := \lambda y.\lambda x.xy$.

Sub-term	Context	Env.	Stack
$(\lambda y.\lambda x.xy)ll$	$\langle \cdot \rangle$	ϵ	ϵ
$\rightarrow_{\text{app}} (\lambda y.\lambda x.xy)l$	$\langle \cdot \rangle l$	ϵ	$(l, Ql\langle \cdot \rangle, \epsilon)$
$\rightarrow_{\text{app}} \lambda y.\lambda x.xy$	$\langle \cdot \rangle ll$	ϵ	$(l, Q\langle \cdot \rangle l, \epsilon) \cdot (l, Ql\langle \cdot \rangle, \epsilon)$
$\rightarrow_{\text{abs}} \lambda x.xy$	$(\lambda y.\langle \cdot \rangle)ll$	$[y \leftarrow (l, Q\langle \cdot \rangle l, \epsilon)]$	$(l, Ql\langle \cdot \rangle, \epsilon)$
$\rightarrow_{\text{abs}} xy$	$(\lambda y.\lambda x.\langle \cdot \rangle)ll$	$[x \leftarrow (l, Ql\langle \cdot \rangle, \epsilon)] \cdot [y \leftarrow (l, Q\langle \cdot \rangle l, \epsilon)] = E$	ϵ
$\rightarrow_{\text{app}} x$	$(\lambda y.\lambda x.\langle \cdot \rangle y)ll$	E	$(y, (\lambda y.\lambda x.x\langle \cdot \rangle)ll, E)$

Thanks to the information saved in the environment, the KAM is able to directly hop to the argument of x , namely the second identity. Moreover, the environment is restored from the closure.

Sub-term	Context	Env.	Stack
x	$(\lambda y.\lambda x.\langle \cdot \rangle y)l(\lambda z.z)$	E	$(y, (\lambda y.\lambda x.x\langle \cdot \rangle)ll, E)$
$\rightarrow_{\text{var}} (\lambda z.z)$	$(\lambda y.\lambda x.xy)l\langle \cdot \rangle$	ϵ	$(y, (\lambda y.\lambda x.x\langle \cdot \rangle)ll, E)$

Then, the computation continues. All application arguments are saved in the stack as closures, *i.e.* together with their environment, and then moved to the environment when a binder λx is encountered (they are also linked to x). Whenever a variable x is reached, its argument is retrieved, together with its environment from the closure linked to that variable x .

Sub-term	Context	Env.	Stack
$(\lambda z.z)$	$(\lambda y.\lambda x.xy)l\langle \cdot \rangle$	ϵ	$(y, (\lambda y.\lambda x.x\langle \cdot \rangle)ll, E)$
$\rightarrow_{\text{abs}} z$	$(\lambda y.\lambda x.xy)l(\lambda z.\langle \cdot \rangle)$	$[z \leftarrow (y, (\lambda y.\lambda x.x\langle \cdot \rangle)ll, E)]$	ϵ
$\rightarrow_{\text{var}} y$	$(\lambda y.\lambda x.x\langle \cdot \rangle)ll$	$[x \leftarrow (l, Ql\langle \cdot \rangle, \epsilon)] \cdot [y \leftarrow (l, Q\langle \cdot \rangle l, \epsilon)]$	ϵ
$\rightarrow_{\text{var}} l$	$(\lambda y.\lambda x.xy)\langle \cdot \rangle l$	ϵ	ϵ

Cost of KAM Transitions. The idea is that environments are implemented as linked lists, so that the duplication and insertion operations in transitions \rightarrow_{app} and \rightarrow_{abs} can be implemented in constant time. Transition \rightarrow_{var} needs to access the environment, whose size is bounded by $|t|$, the size of the initial term t of the run. By adopting smarter implementations of environments, one \rightarrow_{var} transition costs $\log |t|$ —see Accattoli and Barras [2017] for discussions about implementations of the KAM. Then implementing on RAM a KAM run π from t costs $|\pi|_{\text{-var}} + |\pi|_{\text{var}} \cdot \log |t|$.

6 THE EXHAUSTIBLE STATE INVARIANT

Here we present the *exhaustible (state) invariant*. In [Accattoli et al. 2020a], this is a key ingredient for the proof of the λ IAM implementation theorem. In this paper, we give it in various forms to establish the relationships between the various machines. Here we present the basic concepts.

The intuition behind the invariant is that whenever a logged position l occurs in a reachable state, it is there *for a reason*: no logged position occurs in initial states, and transitions only add logged positions to which the machine may come back. In particular, if the state is set in the right way (to be explained), the λ IAM can reach l , *exhausting* it.

Preliminaries. Exhaustible states rest on some *tests* for their logged positions. More specifically, each logged position l in a state s has an associated test state s_l that tunes the data structures of s as to test for the reachability of l . Actually, there shall be *two* classes of test states, one accounting for the logged positions in the tape of s , and one for the those in the log of s . The technical definition of log tests, however, is in Appendix A. They are essential for the proof of the exhaustible invariant, but they are not needed for showing the main consequence of interest in this section, that is, that backtracking always succeeds (Lemma 6.5 below), which is why they are omitted.

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

Definition 6.1 (Tape tests). Let $s = (t, C_n, L_n, T' \cdot l \cdot T'', d)$ be a state. Then the *tape test of s of focus l* is the state $s_l = (t, C_n, L_n, T' \cdot l, \uparrow^{|T' \cdot l|} l)$.

Note that the direction of tape tests is reversed with respect to what stated by the *tape and direction* invariant (Lemma 3.1), and so, in general, they are not reachable states. Such a counter-intuitive fact is needed for the invariant to go through, no more no less. When we shall use the properties of tests to prove properties of the λ IAM (Lemma 6.5 below), we shall extend their tape via the tape lifting property (Lemma 3.3) as to satisfy the invariant and be reachable. Exhausting a logged position l means backtracking to it. We then decorate the backtracking transition $\rightarrow_{\text{bt}1}$ and $\rightarrow_{\text{bt}2}$ as $\rightarrow_{\text{bt}1,l}$ and $\rightarrow_{\text{bt}2,l}$ to specify the involved logged position l . We also need a notion of state positioned in l and having an empty tape, which is meant to be the target state of $\rightarrow_{\text{bt}2,l}$ when exhausting l starting on s_l .

Definition 6.2 (State surrounding a position). Let $l = (t, D, L')$ be a logged position. A state s surrounds l if $s = (t, \underline{C_n \langle D \rangle}, L' \cdot L_n, \epsilon)$ for some C_n and L_n .

The Exhaustibility Invariant. After having introduced all the necessary preliminaries, we can now formulate the property of states that we shall soon prove to be an invariant.

Definition 6.3 (Exhaustible States). \mathcal{E} is the smallest set of states s such that if s_l is a tape or a log test of s then there exists a run $\pi : s_l \xrightarrow{\lambda\text{IAM}}^* \rightarrow_{\text{bt}2,l} s''$, where s'' surrounds l and for the shortest of such runs π it holds that $s'' \in \mathcal{E}$. States in \mathcal{E} are called *exhaustible*.

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

PROPOSITION 6.4 (EXHAUSTIBLE INVARIANT [ACCATTOLI ET AL. 2020A]). *Let s be a λ IAM reachable state. Then s is exhaustible.*

A key consequence is the fact that backtracking always succeeds, as it amounts to exhausting the first logged position on the log.

LEMMA 6.5 (BACKTRACKING ALWAYS SUCCEEDS). *Let s a λ IAM reachable state. If $s \rightarrow_{\text{bt}1,l} s'$ then there exists s'' such that $s' \xrightarrow{\lambda\text{IAM}}^* \rightarrow_{\text{bt}2,l} s''$.*

PROOF. Consider $s = (t, \underline{C \langle u \langle \cdot \rangle \rangle}, l \cdot L, T) \rightarrow_{\text{bt}1,l} (\underline{u}, C \langle \langle \cdot \rangle t \rangle, L, l \cdot T) = s'$. Since s' is reachable then it is exhaustible, and so its tape test $s'_l := (\underline{u}, C \langle \langle \cdot \rangle t \rangle, L, l)$ can be exhausted, that is, there is a λ IAM run $\pi : s'_l \xrightarrow{\lambda\text{IAM}}^* \rightarrow_{\text{bt}2,l} q$ for a state q surrounding l . Note that s'_l is s' where the tape contains only l . Now, we lift π to a run $\pi^T : s' \xrightarrow{\lambda\text{IAM}}^* \rightarrow_{\text{bt}2,l} s''$ using the tape lifting lemma (Lemma 3.3). \square

7 RELATING THE λ -IAM AND THE λ -JAM: JUMPING IS EXHAUSTING

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

Log Tests. For simulating jumps we need log tests. The idea is the same underlying tape tests: they focus on a given logged position in the log. Their definition however requires more than simply stripping down the log, as the new log and the position still have to form a logged position—said differently, the *position and the log* invariant (Lemma 3.1) has to be preserved. Roughly, the log test

s_{l_m} focussing on the m -th logged position l_m in the log of a state $(t, C_n, l_n \cdots l_2 \cdot l_1, T, d)$ is obtained by removing the prefix $l_n \cdots l_{m+1}$ (if any), and moving the current position up by $n - m$ levels. Moreover, the tape is emptied and the direction is set to \uparrow .

In the argument for the simulation of jumps given below, we need only log tests of a very simple form. Namely, given a state $s = (t, \underline{C\langle u \cdot \cdot \rangle}, l \cdot L, T)$ from which the λ JAM jumps, we shall consider the log test $s_l := (t, \underline{C\langle u \cdot \cdot \rangle}, l \cdot L, \epsilon)$, that is, the tape is emptied and (in this case) the position does not change. The more general form of log tests needing the position change is technical and defined in Appendix A—it is unavoidable for proving the invariant, but we fear that giving it here would obfuscate the use of the exhaustible technique, whose idea is instead quite simple.

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

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

LOGGED POSITIONS	$I(x, C\langle \lambda x.D_n \rangle, L_n \cdot L) := (x, \lambda x.D_n, I(L_n))$		
TAPES AND LOGS	$I(\epsilon) := \epsilon$	$I(l \cdot \Gamma) := I(l) \cdot I(\Gamma)$	$I(\bullet \cdot T) := \bullet \cdot I(T)$
STATES	$I(t, C, L, T, d) := (t, C, I(L), I(T), d)$		

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

Definition 7.1 (I-Exhaustible States). \mathcal{E}_I is the smallest set of λ JAM states s such that for any tape or log test s_l of s of focus l , there exists a run $\pi : I(s_l) \xrightarrow{*}_{\lambda\text{IAM}} \xrightarrow{\text{bt2}, I(l)} I(l^\circ)$ such that $l^\circ \in \mathcal{E}_I$. States in \mathcal{E}_I are called *I-exhaustible*.

LEMMA 7.2 (I-EXHAUSTIBLE INVARIANT). *Let s be a λ JAM reachable state. Then s is I-exhaustible.*

Jumping is Exhausting. From the invariant and the tape lifting property of the λ IAM, it follows easily that hops can be simulated via backtracking, from which the relationship between the λ IAM and the λ JAM immediately follows. We write $\rightarrow_{\text{jmp}, l}$ for a \rightarrow_{jmp} transition jumping to l .

LEMMA 7.3 (JUMPS SIMULATION VIA BACKTRACKING). *Let s be a λ JAM reachable and $s \rightarrow_{\text{jmp}, l} s'$. Then $I(s) \xrightarrow{\text{bt1}, I(l)} \xrightarrow{*}_{\lambda\text{IAM}} \xrightarrow{\text{bt2}, I(l)} I(s')$.*

PROOF. Let $l := (x, D, L')$ and consider $s = (t, \underline{C\langle u \cdot \cdot \rangle}, (x, D, L') \cdot L, T) \rightarrow_{\text{jmp}, l} (x, \underline{D}, L', T) = s'$. Since s is reachable then it is I-exhaustible, so its log test $s_l := (t, \underline{C\langle u \cdot \cdot \rangle}, l \cdot L, \epsilon)$ can be exhausted, that is, there is a λ IAM run $\pi : I(s_l) \xrightarrow{*}_{\lambda\text{IAM}} \xrightarrow{\text{bt2}, I(l)} I(x, \underline{D}, L', \epsilon) = s''$. Note that the first transition of π is necessarily $\rightarrow_{\text{bt1}, I(l)}$. Moreover, $I(s_l)$ and s'' are exactly $I(s)$ and $I(s')$ with empty tape. We lift π to a run $\pi^{I(T)} : I(s_l)^{I(T)} \xrightarrow{\text{bt1}, I(l)} \xrightarrow{*}_{\lambda\text{IAM}} \xrightarrow{\text{bt2}, I(l)} s''^{I(T)}$ using Lemma 3.3. Now, $\pi^{I(T)}$ is exactly the λ IAM simulation of the jump, because $I(s_l)^{I(T)} = I(s)$ and $s''^{I(T)} = I(s)$. \square

From the lemma it easily follows a bisimulation between the λ IAM and the λ JAM, showing that the latter is faster. In Appendix A, there is a general theorem relating also potentially diverging runs. Here we give only the more concise statement about complete runs.

THEOREM 7.4 (λ IAM AND λ JAM RELATIONSHIP). *There is a complete λ JAM run π_J from t if and only if there is a complete λ IAM run π_I from t . In particular, the λ JAM implements Closed CbN. Moreover, $|\pi_J| \leq |\pi_I|$ and $|\pi_J|_{\text{var}} \leq |\pi_I|_{\text{var}}$.*

LOGGED CLOSURES					ENVIRONMENTS					
$\hat{c} ::= (t, C_{n+1}, E)^{L_n}$					$E ::= \epsilon \mid [x \leftarrow \hat{c}] \cdot E$					
LOGS					CLOSED POSITIONS					
$L_0 ::= \epsilon \quad L_{n+1} ::= \hat{l} \cdot L_n$					$\hat{l} ::= (t, C_n, L_n)^E$					
CLOSED TAPES					STATES					
$T ::= \epsilon \mid \hat{c} \cdot T \mid \hat{l} \cdot T$					$s ::= (t, C, L, E, T, d)$					
Term	Ctx	Log	Env	Cl. Tape	Term	Ctx	Log	Env	Cl. Tape	
\underline{tu}	C	L	E	T	\underline{t}	$C\langle\langle\cdot\rangle u\rangle$	L	E	$\hat{c} \cdot T$	
$\underline{\lambda x.t}$	C	L	E	$\hat{c} \cdot T$	\underline{t}	$C\langle\lambda x.\langle\cdot\rangle\rangle$	L	$[x \leftarrow \hat{c}] \cdot E$	T	
\underline{x}	$C\langle\lambda x.D_n\rangle$	$L_n \cdot L$	$E' \cdot [x \leftarrow \hat{c}] \cdot E$	T	$\lambda x.D_n(x)$	\underline{C}	L	E	$\hat{l} \cdot T$	
\underline{x}	C	L	E	T	\underline{u}	D	$\hat{l} \cdot L'$	F	T	
where	$\hat{c} := (u, C\langle t\langle\cdot\rangle\rangle, E)^L$ in $\rightarrow_{\bullet 1/\text{app}}$ $\hat{l} := (x, C\langle\lambda x.D\rangle, L_n \cdot L)^{E' \cdot [x \leftarrow \hat{c}] \cdot E}$ in $\rightarrow_{\text{varj}}$ $E = E' \cdot [x \leftarrow (u, D, F)^{L'}] \cdot E''$ and $\hat{l} = (x, C, L)^E$ in $\rightarrow_{\text{hop/varK}}$.									
t	$\underline{C\langle\langle\cdot\rangle u\rangle}$	L	E	$\hat{c} \cdot T$	$\rightarrow_{\bullet 3}$	tu	\underline{C}	L	E	T
t	$\underline{C\langle\lambda x.\langle\cdot\rangle\rangle}$	L	$[x \leftarrow \hat{c}] \cdot E$	T	$\rightarrow_{\bullet 4}$	$\lambda x.t$	\underline{C}	L	E	$\hat{c} \cdot T$
t	$\underline{C\langle\langle\cdot\rangle u\rangle}$	L	E	$\hat{l} \cdot T$	\rightarrow_{arg}	\underline{u}	$C\langle t\langle\cdot\rangle\rangle$	$\hat{l} \cdot L$	E	T
t	$\underline{C\langle u\langle\cdot\rangle\rangle}$	$\hat{l} \cdot L$	E	T	\rightarrow_{jmp}	x	\underline{D}	L'	E'	T
where in the last transition $\hat{l} = (x, D, L')^{E'}$.										

Fig. 4. Data structures and transitions of the Hopping Abstract Machine (HAM).

Exponential Gap. The time gap between the λ IAM and the λ JAM can be exponential, as it is shown by the family of terms t_n ($t_1 := l$ and $t_{n+1} := t_n l$) mentioned in the introduction. The results of this paper provide a nice high-level proof. Next section shows that the time of the λ JAM is polynomial in the time of the KAM, that takes time polynomial in the number of β -steps to evaluate t_n , that is, n . The study of multi types in Sect. 12 instead shows that the time of the λ IAM depends on the size of the smallest type A_n of t_n , which is easily seen to be exponential in n . In fact, using the notation of Sect. 12, $A_1 := [\star] \rightarrow \star$, and $A_{n+1} := [A_n] \rightarrow A_n$.

8 ENTANGLING THE λ JAM AND THE KAM: THE HAM

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

The HAM. To prove that the KAM is a time improvement of the λ JAM, we introduce an intermediate machine, the *Hopping Abstract Machine* (HAM) in Fig. 4, that merges the two. The HAM is a technical tool addressing an inherent difficulty: the λ JAM and the KAM use different data structures and it is impossible to turn a KAM state into a λ JAM state without having to look at the whole run that led to that state, as it is instead possible for the λ JAM and the λ IAM.

The idea behind the HAM is to entangle the data structures of both machines (so that their states get paired by construction), and to allow it to behave non-deterministically either as the λ JAM or as the KAM. The HAM deals with two enriched objects, *logged closures* \hat{c} and *closed (logged) positions* \hat{l} (defined in Fig. 4, overloading some of the notations of the previous sections), obtained by adding a log to closures and an environment to logged positions. Of course, environments and logs have to be redefined as containing these enriched objects. There is also a *(closed) tape* T , that is, a data structure obtained by merging the roles of the stack and the tape and containing both logged closures and closed positions. In fact the closed tape is obtained from the λ JAM tape by upgrading every \bullet entry to a logged closure \hat{c} , and every logged position l to a closed one \hat{l} . Note

that logged closures and closed positions contain the same information (a term, a context, a log, and an environment) but they play different roles.

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

It is evident that by removing environments, turning every logged closure into \bullet , and removing $\rightarrow_{\text{hop/varK}}$ we obtain the λJAM . Similarly, by removing logs, $\rightarrow_{\text{varJ}}$, and the \uparrow transitions, one obtains the KAM. We avoid spelling out these immediate projections. Instead, we see KAM runs inside the HAM as given by the transition $\rightarrow_{\text{HAMK}} := \rightarrow_{\bullet 1/\text{app}} \cup \rightarrow_{\bullet 2/\text{abs}} \cup \rightarrow_{\text{hop/varK}}$. Similarly, the λJAM is seen as transition $\rightarrow_{\text{HAMJ}}$, defined as the union of all HAM transitions but $\rightarrow_{\text{hop/varK}}$.

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

LEMMA 8.1 (HAM TAPE LIFT). *Let $\pi : s = (t, C, L, E, T', d) \rightarrow_{\text{HAM}}^n (u, D, L', E', T'', d') = s'$ be a run and T be a tape. Then there is a run $\pi^T : s^T = (t, C, L, E, T' \cdot T, d) \rightarrow_{\text{HAM}}^n (u, D, L', E', T'' \cdot T, d') = s'^T$.*

9 HOPPING IS ALSO EXHAUSTING

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

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

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

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

Definition 9.1 (HAM Environment tests). Let $s = (t, C \langle \lambda x. D_n \rangle, L_n \cdot L, E' \cdot [x \leftarrow \hat{c}] \cdot E, T, d)$ be a state. Then, $s_{\hat{c}} := (\lambda x. D_n \langle t \rangle, \underline{C}, L, E, \epsilon)$ is an environment test for s of focus \hat{c} .

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

Definition 9.2 (HAM State induced by a logged closure). Given a logged closure $\hat{c} = (u, D \langle t \langle \cdot \rangle \rangle, E)^L$, the state \hat{c}° induced by \hat{c} is defined as $\hat{c}^\circ := (t, \underline{D} \langle \langle \cdot \rangle \rangle u, L, E, \epsilon)$.

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

Definition 9.3 (HAM \uparrow -Exhaustible states). \mathcal{E}_{\uparrow} is the smallest set of those states s such that for any tape or environment test $s_{\hat{c}}$ of s , there exists a run $\pi_{\uparrow} : s_{\hat{c}} \rightarrow_{\uparrow}^* \hat{c}^\circ$ and $\hat{c}^\circ \in \mathcal{E}_{\uparrow}$. States in \mathcal{E}_{\uparrow} are called \uparrow -*exhaustible* (pronounced *up*-exhaustible).

LEMMA 9.4. *Let s be a HAM reachable state. Then s is \uparrow -exhaustible.*

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

LEMMA 9.5 (HOPS SIMULATION VIA \uparrow). *Let s be a HAM reachable state and $s \rightarrow_{\text{hop/varK}} s'$. Then $s \rightarrow_{\text{varJ}} s'' \rightarrow_{\uparrow}^+ s'$.*

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

From the lemma it easily follows a bisimulation between the λ JAM and the KAM, showing that the latter is faster. In Appendix B, there is a theorem relating their runs inside the HAM, considering also potentially diverging runs. Here we give only the more concise statement about complete runs.

THEOREM 9.6 (λ JAM AND KAM RELATIONSHIP). *There is a complete λ JAM run π_J from t if and only if there is a complete KAM run π_K from t . Moreover, $|\pi_J| = |\pi_K| + |\pi_J|_{\uparrow}$ and $|\pi_J|_{\text{var}} = |\pi_K|_{\text{var}}$.*

10 THE λ -JAM IS SLOWLY REASONABLE

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

Locating the λ JAM. We have proved in the previous two sections that a run π_J of the λ JAM from t is such that $|\pi_K| \leq |\pi_J| \leq |\pi_I|$, where π_K and π_I are the runs from t respectively of the KAM and of the λ IAM. However, this tells nothing about the inherent complexity of evaluating a term with the λ JAM. In fact, it is well known that $|\pi_K|$ is polynomial in $\#\beta$ and $|t|$ (namely quadratic in $\#\beta$ and linear in $|t|$), while $|\pi_I|$ can be exponential in both $\#\beta$ and $|t|$ (the typical example of exponential behavior being the family of terms t_n defined as $t_1 := 1$ and $t_{n+1} := t_n 1$). What about the λ JAM? Is it polynomial or exponential? It turns out that the λ JAM is polynomial, and precisely at most quadratically slower than the KAM.

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

LEMMA 10.1. *Let $\pi : (t, \underline{C}, L, T) \rightarrow_{\bullet, 3, 4}^* s$. Then $|\pi| \leq |C| \leq |C\langle t \rangle|$.*

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

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

PROPOSITION 10.2 (DEPTH INVARIANT). *Let $\pi : s_t \rightarrow_{\lambda\text{JAM}}^* s$ be an initial run of the λJAM . Then $\text{depth}(s) = |\pi|_{\text{var}}$. Moreover $\text{depth}(s) \geq \text{depth}(l)$ for every logged position l in s .*

Third, we bound \uparrow phases. The number of jumps in a single phase $s \rightarrow_{\uparrow}^* s'$ of \uparrow transitions is bound by $\text{depth}(s)$, and pairing it up with Lemma 10.1 we obtain a bound on the phase. By the depth invariant the bound can be given relatively to $|\pi|_{\text{var}}$, and a standard argument extends the bound to all \uparrow phases in a run, adding a quadratic dependency. Let $|\pi|_{\uparrow}$ be the number of \rightarrow_{\uparrow} transitions in π .

LEMMA 10.3 (BOUND ON \uparrow PHASES).

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

The Complexity of the KAM. We need to recall the complexity analysis of the KAM from [Accattoli et al. 2014; Accattoli and Barras 2017; Accattoli and Dal Lago 2012]. The length of a complete KAM run π from t verifies $|\pi| = |\pi|_{\text{var}} + 2 \cdot |\pi|_{\text{abs}}$ and we have $|\pi|_{\text{var}} = \mathcal{O}(|\pi|_{\text{abs}}^2)$ (the bound is tight, as there are examples reaching it). Since $|\pi|_{\text{abs}}$ is exactly the number $\#\beta$ of \rightarrow_{wh} steps to evaluate t (the cost model of reference for Closed CbN), we have that $|\pi| = \mathcal{O}(\#\beta^2)$. Now since the cost of implementing KAM single transitions on RAM is bound by $|t|$, the complexity of implementing the KAM is $\mathcal{O}(\#\beta^2 \cdot |t|)$, that is, the KAM is a reasonable machine.

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

THEOREM 10.4 (λJAM COMPLEXITY). *Let t be a closed term such that $t \rightarrow_{\text{wh}}^n u$, u be \rightarrow_{wh} normal, and π_J and π_K be the complete λJAM and KAM runs from t . Then:*

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

11 THE POINTER ABSTRACT MACHINE, REVISITED

The Pointer Abstract Machine (PAM), due to Danos and Regnier [Danos et al. 1996; Danos and Regnier 2004], gives an operational account of the interaction process at work in Hyland and Ong game semantics. The machine is always described rather informally via a pseudo-code algorithm. Here we define it according to our syntactic style, calling it λPAM , and provide its first formal and manageable presentation as an actual abstract machine.

Our result concerning the λPAM is that it is strongly bisimilar to the λJAM . Roughly, the two are the same machine, with exactly the same time behavior, they just use different data structures. This connection is mentioned in [Danos and Regnier 1999], but not proved. We find it instructive to spell it out, as the connection is elegant but far from being evident.

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

index i intuitively realizes a mechanism to retrieve the log associated to p by the λ JAM. We first define the machine and then explain the relationship between the two approaches.

Data Structures. All the data structures of the PAM are defined in Fig. 5. Positions are no longer logged, and noted with p, p' , etc. An *index* i is simply a natural number. *Indexed positions* are pairs (p, i) . A history H is a sequence of indexed variable positions (accumulated from right to left). The idea is that indices are pointers to entries in the history, that is, if the i -th entry of H is (p, j) then j points to a previous entry in H , that is, $j < i$. The tape of the λ PAM is a stack containing variable positions and occurrences of \bullet .

Transitions and Look-Up. Initial states have the form $s_t := (t, \langle \cdot \rangle, \epsilon, 0, \epsilon)$, the transitions of the λ PAM are in Fig. 5, they are labeled exactly as in the λ JAM, and their union is noted $\rightarrow_{\lambda\text{PAM}}$. Transitions \rightarrow_{var} and \rightarrow_{jmp} need to retrieve information from the history H , for which there is some dedicated notation. We use i_k^H, x_k^H, D_k^H to denote, respectively, the index, variable, and context of the k -th indexed position in H .

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

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

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

LEMMA 11.1 (λ PAM INVARIANTS). *Let $s = (t, C_n, H, i, T, d)$ be a reachable PAM state. Then:*

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

An Example. As for the other machines we have considered in this paper, we give the execution trace of the λ PAM on the term $(\lambda y.\lambda x.xy)\text{II}$. The reader can grasp some intuition considering that the PAM is strongly bisimilar to the λ JAM. In particular, the λ PAM considers explicit pointers. Indeed, as we have

already pointed out, λ JAM logs are not actually copied in the λ JAM \rightarrow_{var} transition: what is duplicated is just a pointer to them. The λ PAM handles this mechanism directly in its definition, and can thus be considered as a low-level implementation of the λ JAM. In the following we will explain this in more detail. After having looked for the head variable through the spine of the term, the λ PAM, now in \uparrow mode, queries the argument of x , namely $\lambda z.z$, that then explores. The

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

POSITIONS	$p ::= (t, C)$				TAPES	$T ::= \epsilon \mid \bullet \cdot T \mid p \cdot T$				
HISTORIES	$H ::= \epsilon \mid (p, i) \cdot H$				STATES	$s ::= (t, C, H, i, T, d) \quad i \in \mathbb{N}$				
Sub-term	Context	Hist.	Index	Tape	Sub-term	Context	Hist.	Index	Tape	
tu	C	H	i	T	$\rightarrow_{\bullet 1}$	\underline{t}	$C\langle\langle \cdot \rangle u\rangle$	H	i	$\bullet \cdot T$
$\lambda x.t$	C	H	i	$\bullet \cdot T$	$\rightarrow_{\bullet 2}$	\underline{t}	$C\langle\lambda x. \langle \cdot \rangle \rangle$	H	i	T
x	$C\langle\lambda x. D_n\rangle$	H	i	T	\rightarrow_{var}	$\lambda x. D_n \langle x \rangle$	\underline{C}	H	$\phi_H^n(i)$	$(x, C\langle\lambda x. D_n\rangle) \cdot T$
t	$\underline{C\langle\langle \cdot \rangle u\rangle}$	H	i	$\bullet \cdot T$	$\rightarrow_{\bullet 3}$	tu	\underline{C}	H	i	T
t	$\underline{C\langle\lambda x. \langle \cdot \rangle \rangle}$	H	i	T	$\rightarrow_{\bullet 4}$	$\lambda x.t$	\underline{C}	H	i	$\bullet \cdot T$
t	$\underline{C\langle\langle \cdot \rangle u\rangle}$	H	i	$p \cdot T$	\rightarrow_{arg}	\underline{u}	$C\langle t \langle \cdot \rangle \rangle$	$(p, i) \cdot H$	$ H + 1$	T
t	$\underline{C\langle u \langle \cdot \rangle \rangle}$	H	i	T	\rightarrow_{jmp}	x_i^H	$\underline{D_i^H}$	H	$i - 1$	T

Fig. 5. Data structures and transitions of the λ Pointer Abstract Machine (λ PAM).

argument of its head variable z is y , that has to be found via *backtracking* or *jumping*. We put $p_x = (x, (\lambda y. \lambda x. \langle \cdot \rangle y) \mid (\lambda z. z))$.

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

The jump is simulated by the λ PAM retrieving the position saved in the history at the current index, and then updating the index accordingly, *i.e.* diminishing it by one. Intuitively, this corresponds to the “unpacking” made by the λ JAM in the \rightarrow_{jmp} transition. We put $p_z = (z, (\lambda y. \lambda x. xy) \mid (\lambda z. \langle \cdot \rangle))$ and $p_y = (y, (\lambda y. \lambda x. x \langle \cdot \rangle) \mid)$.

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

History, Indices, and Logs. The history H essentially stores the sequence of \rightarrow_{var} queries, consisting of the position of a variable needing an argument, that the λ PAM has completed, that is, for which it has found the argument. The key point is that it stores them with an index i . Indices are a low-level mechanism to retrieve logs, that are crumbled and shuffled all over H .

Let us explain how a log $(p_1, L_1) \cdot \dots \cdot (p_n, L_n)$ of a reachable λ JAM state is represented by the index i_1 and the history H of the corresponding λ PAM state. There are two ideas:

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

The Bisimulation. The given explanation underlies the following definition of relations \simeq_T , \simeq_{LH} and \simeq between data structures and states of the λ JAM and the λ PAM, that induce a strong

bisimulation. The intended meaning of the relation $L \simeq_{LH} (H, i)$ is that the log L is represented in the history H starting from index i .

Definition 11.2. The relations \simeq_T , \simeq_{LH} and \simeq are defined as follows.

$$\begin{array}{c}
 \text{TAPES} \quad \frac{}{\bar{\epsilon} \simeq_T \bar{\epsilon}} \quad \frac{T_J \simeq_T T_P}{\bullet T_J \simeq_T \bullet T_P} \quad \frac{T_J \simeq_T T_P}{(x, C, L) \cdot T_J \simeq_T (x, C) \cdot T_P} \\
 \text{LOG-HISTORIES} \quad \frac{}{\epsilon \simeq_{LH} (H, 0)} \quad \frac{(x, C) = (x_i^H, D_i^H) \quad L \simeq_{LH} (H, \phi_H(i)) \quad L' \simeq_{LH} (H, i - 1)}{(x, C, L') \cdot L \simeq_{LH} (H, i)} \\
 \text{STATES} \quad \frac{T_J \simeq_T T_P \quad L \simeq_{LH} (H, i)}{(t, C, L, T_J, d) \simeq (t, C, H, i, T_P, d)}
 \end{array}$$

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

THEOREM 11.3 (\simeq IS A STRONG BISIMULATION).

- (1) for every run $\pi_J : s_t^{\lambda JAM} \rightarrow_{\lambda JAM}^* s_J$ there exists a run $\pi_P : s_t^{\lambda PAM} \rightarrow_{\lambda PAM}^* s_P$ such that $s_J \simeq s_P$ and $|\pi_J| = |\pi_P|$ and performing exactly the same transitions;
- (2) for every run $\pi_P : s_t^{\lambda PAM} \rightarrow_{\lambda PAM}^* s_P$ there exists a run $\pi_J : s_t^{\lambda JAM} \rightarrow_{\lambda JAM}^* s_J$ such that $s_J \simeq s_P$ and $|\pi_J| = |\pi_P|$ and performing exactly the same transitions.

Moreover, if $s_J = (t, \underline{C}, L, T_J, \uparrow) \simeq (t, \underline{C}, H, i, T_P, \uparrow) = s_P$ and (x, D, L') is the unique logged position in T_J then $L' \simeq_{LH} (H, |H|)$.

Strong bisimulations trivially preserve termination and the length of runs.

COROLLARY 11.4 (TERMINATION AND λ PAM IMPLEMENTATION). $\lambda JAM(t) \Downarrow$ if and only if $\lambda PAM(t) \Downarrow$, and the two runs use exactly the same transitions. Therefore, the λ PAM implements Closed CbN.

12 SEQUENCE TYPES

Here we introduce a type system that we shall use to measure the length of λ IAM runs.

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

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

$$\text{LINEAR TYPES } A, A' ::= \star \mid S \rightarrow A \quad \text{SEQUENCE TYPES } S, S' ::= [A_1, \dots, A_n]$$

Since commutativity is ruled out, we have, e.g., $[A, A'] \neq [A', A]$. We shall use $[\cdot]$ as a generic list constructor not limited to types, thus writing $[2, 1, 12, 4]$ for a list of natural numbers, and also

$$\begin{array}{c}
\frac{}{x : [A] \vdash x : A} \text{T-VAR} \qquad \frac{\Gamma, x : S \vdash t : A}{\Gamma \vdash \lambda x.t : S \rightarrow A} \text{T-}\lambda \qquad \frac{}{\Gamma \vdash \lambda x.t : \star} \text{T-}\lambda_{\star} \\
\frac{\Gamma \vdash t : [A'_1, \dots, A'_n] \rightarrow A \quad [\Delta_i \vdash u : A'_i]_{i \in [1, \dots, n]}}{\Gamma \uplus \sum_{i \in [1, \dots, n]} \Delta_i \vdash tu : A} \text{T-@}
\end{array}$$

Fig. 6. The sequence type system.

use it for lists of judgments or type derivations. Note that there is a ground type \star , which can be thought as the type of normal forms, that in Closed CbN are precisely abstractions. Note also that arrow (linear) types $S \rightarrow A$ can have a sequence only on the left. The empty sequence is noted $[\]$, and the concatenation of two sequences S and S' is noted $S \uplus S'$.

Type judgments have the following form $\Gamma \vdash t : A$, where Γ is a type environment, defined below. The typing rules are in Fig. 6, type derivations are noted π and we write $\pi \triangleright \Gamma \vdash t : A$ for a type derivation π of ending judgement $\Gamma \vdash t : A$. Type environments, ranged over by Γ, Δ are total maps from variables to sequence types such that only finitely many variables are mapped to non-empty sequence types, and we write $\Gamma = x_1 : S_1, \dots, x_n : S_n$ if $\text{dom}(\Gamma) = \{x_1, \dots, x_n\}$ —note that type environments are commutative, what is non-commutative is only the sequence constructor $[\]$. Given two type environments Γ, Δ , the expression $\Gamma \uplus \Delta$ stands for the type environment assigning to every variable x the list $\Gamma(x) \uplus \Delta(x)$. A sequence $\Gamma_{i_1}, \dots, \Gamma_{i_k}$ of type environments is also noted $\{\Gamma_i\}_{i \in [i_1, \dots, i_k]}$, or $\{\Gamma_i\}_{i \in I}$ with $I = [i_1, \dots, i_k]$. Moreover, we use $\sum_{i \in I} \Gamma_i$ for the type environment defined as $\sum_{i \in I} \Gamma_i := [\]$ if $I = [\]$, and $\sum_{i \in I} \Gamma_i := \Gamma_{i_1} \uplus \sum_{i \in I'} \Gamma_i$ if $I = [i_1] \uplus I'$.

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 is a correspondence between sequence types and axioms. We write $|S|$ for the length of S as a sequence.

LEMMA 12.1 (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 $|\Gamma(x)|$ axioms typing x in π , which appear from left to right as leaves of π (seen as an ordered tree) in the order given by $\Gamma(x) = [A_1, \dots, A_k]$ and that the i -th axiom types x with A_i .*

Characterization of Termination. It is well-known that intersection and multi types characterize Closed CbN termination, that is, they type *all* and only those λ -terms that terminate with respect to weak head reduction. If terms are closed, the same result smoothly holds for sequence types, as we now explain. The only point where non-commutativity is delicate for the characterization is in the proof of the typed substitution lemma for subject reduction (and the dual lemma for subject expansion), as substitution may change the order of concatenation in type environments. In our simple setting where terms are closed, however, the term to substitute is closed¹² and—by the relevance lemma—its type derivation comes with no type environment, so the order-of-concatenation problem disappears. Therefore, sequence types characterize termination in Closed CbN too. Thus from now on we essentially identify multi and sequence types.

THEOREM 12.2. *A closed term t has weak head normal form if and only if $\vdash t : \star$.*

Sequence Types and KAM Time. Multi types have been successfully applied in quantitative analyses of normalization, starting with de Carvalho [2007, 2018] who used them to give a bound

¹²It is well known that in Closed CbN the substitutions $t\{x \leftarrow u\}$ associated to reduced β -redexes are such that u is closed. The term t is of course (potentially) open, and its type derivation has a type environment Γ , but the important point here is that the type derivation of u has no type environment, so that the substitution does not concatenate sequence types.

$$\begin{array}{c}
\text{KAM} \\
\frac{}{\vdash^1} \text{T-VAR} \quad \frac{\vdash^w}{\vdash^{w+1}} \text{T-}\lambda \quad \frac{}{\vdash^0} \text{T-}\lambda_\star \\
\frac{\vdash^w \quad [\vdash^{o_i}]_{i \in [1, \dots, n]}}{\vdash^{w+\sum o_i+1}} \text{T-}@
\end{array}
\left|
\begin{array}{c}
\text{\lambda IAM} \\
\frac{}{[A] \vdash^{\|A\|} A} \text{T-VAR} \quad \frac{\Gamma, S \vdash^w A}{\Gamma \vdash^{w+\|S \rightarrow A\|} S \rightarrow A} \text{T-}\lambda \quad \frac{}{\Gamma \vdash^0 \star} \text{T-}\lambda_\star \\
\frac{\Gamma \vdash^w [A'_1, \dots, A'_n] \rightarrow A \quad [\Delta_i \vdash^{o_i} A'_i]_{i \in [1, \dots, n]}}{\Gamma \uplus \sum_{i \in [1, \dots, n]} \Delta_i \vdash^{w+\sum o_i+\|A\|} A} \text{T-}@
\end{array}
\right.$$

Fig. 7. The weight assignments $\mathbf{W}_{\text{KAM}}(\cdot)$, on the left, and $\mathbf{W}_{\lambda\text{IAM}}(\cdot)$, on the right.

to the length of KAM runs. De Carvalho's technique can be re-phrased and distilled as a decoration of type derivations with *weights*, that is, cost annotations, following the scheme of Fig. 7. Please note that the weight assignment is blind to types, and thus relies only on the structure of the type derivation. De Carvalho's result can be formulated as follows.

THEOREM 12.3 (DE CARVALHO). *There is a complete KAM run of length n from t if and only if $\mathbf{W}_{\text{KAM}}(\pi) = n$ for every $\pi \triangleright \vdash t : \star$.*

Sequence Types and λIAM Time. We use the same idea to capture the length of a λIAM run. We keep the same type system but we change the weight assignment to typing rules. First, we have to define a norm on types and sequence types, counts the number of occurrences of \star :

$$\|\star\| = 1 \quad \|S \rightarrow A\| = \|S\| + \|A\| \quad \|[A_1, \dots, A_n]\| = \sum_{1 \leq i \leq n} \|A_i\|$$

Then we define the weight system $\mathbf{W}_{\lambda\text{IAM}}(\cdot)$ in Fig. 7. Observe how this weight system is structurally very similar to $\mathbf{W}_{\text{KAM}}(\cdot)$, the only difference being the fact that whenever the latter adds 1 to the weight, the former adds the number of occurrences of \star in the underlying type. The next section proves the following theorem, that is the λIAM analogous of de Carvalho's theorem.

THEOREM 12.4. *There is a complete λIAM run of length n from t if and only if $\mathbf{W}_{\lambda\text{IAM}}(\pi) = n$ for every $\pi \triangleright \vdash t : \star$.*

13 THE SEQUENCE IAM

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

SIAM. The idea behind the SIAM is simple but a formal definition is a technical nightmare. Let us explain the idea. The machine moves over a fixed type derivation $\pi \triangleright \vdash t : \star$, to be thought as the code. The position of the machine is expressed by an occurrence of a type judgement¹³ J of π . As the λIAM , the SIAM has two possible directions, noted \downarrow and \uparrow ¹⁴. In direction \uparrow the machine looks at the rule above the focused judgement, in direction \downarrow at the rule below. The only "data structure" is a type context \mathbb{B} isolating an occurrence of \star in the type A of the focused judgement (occurrence) $\Gamma \vdash u : A$, defined as follows (careful to not confuse type contexts \mathbb{B} with type environments Γ):

$$\text{TYPE CTXS} \quad \mathbb{B} ::= \langle \cdot \rangle \mid S \rightarrow \mathbb{B} \mid \mathbb{S} \rightarrow A \quad \mid \text{SEQUENCE CTXS} \quad \mathbb{S} ::= [A_1, \dots, A_k, \mathbb{B}, A_{k+1}, \dots, A_n]$$

Summing up, a state s is a quadruple (π, J, \mathbb{B}, d) . If J is in the form $\Gamma \vdash u : A$, we often write s as $\vdash u : \mathbb{B} \langle \star_d \rangle$, where $\mathbb{B} \langle \star \rangle = A$. In fact we will see soon that the type environment is not needed.

¹³A judgement may occur repeatedly in a derivation, which is why we talk about *occurrences* of judgements. To avoid too many technicalities, we usually just write the judgement, leaving implicit that we refer to an occurrence of that judgement.

¹⁴Type derivations are upside-down wrt to the term structure, then direction \downarrow of the λIAM becomes here \uparrow , and \uparrow is \downarrow .

$$\begin{array}{c}
\frac{\frac{\vdash t : S \rightarrow A \quad [\uparrow]}{\vdash tu : \mathbb{B}(\star_{\uparrow})(=A)} \quad \rightarrow_{\bullet 1} \quad \frac{\vdash t : S \rightarrow \mathbb{B}(\star_{\uparrow}) \quad [\uparrow]}{\vdash tu : A}}{\vdash t : S \rightarrow \mathbb{B}(\star_{\downarrow}) \quad [\uparrow]} \quad \rightarrow_{\bullet 3} \quad \frac{\vdash t : S \rightarrow A \quad [\uparrow]}{\vdash tu : \mathbb{B}(\star_{\downarrow})}}{\vdash t : \mathbb{B}(\star_{\downarrow})(=A)} \quad \rightarrow_{\bullet 4} \quad \frac{\vdash t : A}{\vdash \lambda x.t : S \rightarrow \mathbb{B}(\star_{\downarrow})}} \\
\frac{\frac{\vdash t : S \rightarrow A \quad [\uparrow]}{\vdash tu : \mathbb{B}(\star_{\uparrow})(=A)} \quad \rightarrow_{\bullet 1} \quad \frac{\vdash t : S \rightarrow \mathbb{B}(\star_{\uparrow}) \quad [\uparrow]}{\vdash tu : A}}{\vdash t : S \rightarrow \mathbb{B}(\star_{\downarrow}) \quad [\uparrow]} \quad \rightarrow_{\bullet 3} \quad \frac{\vdash t : S \rightarrow A \quad [\uparrow]}{\vdash tu : \mathbb{B}(\star_{\downarrow})}}{\vdash t : \mathbb{B}(\star_{\downarrow})(=A)} \quad \rightarrow_{\bullet 4} \quad \frac{\vdash t : A}{\vdash \lambda x.t : S \rightarrow \mathbb{B}(\star_{\downarrow})}} \\
\frac{\frac{\frac{\overline{\vdash x : \mathbb{B}(\star_{\uparrow})_i (=A_i)} \quad i}{\vdots}}{\vdash \lambda x.C(x) : [\dots A_i \dots] \rightarrow A'} \quad \rightarrow_{\text{var}} \quad \frac{\frac{\overline{\vdash x : A_i} \quad i}{\vdots}}{\vdash \lambda x.C(x) : [\dots \mathbb{B}(\star_{\downarrow})_i \dots] \rightarrow A'}}{\vdash t : [\dots \mathbb{B}(\star_{\downarrow})_i \dots] \rightarrow A' \quad \vdash_i u : A_i (= \mathbb{B}(\star)_i)} \quad \rightarrow_{\text{arg}} \quad \frac{\vdash t : [\dots A_i \dots] \rightarrow A' \quad \vdash_i u : \mathbb{B}(\star_{\uparrow})_i}{\vdash tu : A'}}{\vdash t : [\dots A_i \dots] \rightarrow A' \quad \vdash_i u : \mathbb{B}(\star_{\downarrow})_i (=A_i)} \quad \rightarrow_{\text{bt1}} \quad \frac{\vdash t : [\dots \mathbb{B}(\star_{\uparrow})_i \dots] \rightarrow A' \quad \vdash_i u : A_i}{\vdash tu : A'}} \\
\frac{\frac{\frac{\overline{\vdash x : A_i (= \mathbb{B}(\star)_i)} \quad i}{\vdots}}{\vdash \lambda x.C(x) : [\dots \mathbb{B}(\star_{\uparrow})_i \dots] \rightarrow A'} \quad \rightarrow_{\text{bt2}} \quad \frac{\frac{\overline{\vdash x : \mathbb{B}(\star_{\downarrow})_i} \quad i}{\vdots}}{\vdash \lambda x.C(x) : [\dots A_i \dots] \rightarrow A'}}{\vdash t : [\dots \mathbb{B}(\star_{\downarrow})_i \dots] \rightarrow A' \quad \vdash_i u : A_i (= \mathbb{B}(\star)_i)} \quad \rightarrow_{\text{arg}} \quad \frac{\vdash t : [\dots A_i \dots] \rightarrow A' \quad \vdash_i u : \mathbb{B}(\star_{\uparrow})_i}{\vdash tu : A'}}{\vdash t : [\dots A_i \dots] \rightarrow A' \quad \vdash_i u : \mathbb{B}(\star_{\downarrow})_i (=A_i)} \quad \rightarrow_{\text{bt1}} \quad \frac{\vdash t : [\dots \mathbb{B}(\star_{\uparrow})_i \dots] \rightarrow A' \quad \vdash_i u : A_i}{\vdash tu : A'}}
\end{array}$$

Fig. 8. The transitions of the Sequence IAM (SIAM).

Transitions. The SIAM starts on the final judgement of π , with empty type context $\mathbb{B} = \langle \cdot \rangle$ and direction \uparrow . It moves from judgement to judgement, following occurrences of \star around π . The transitions are in Fig. 8, their union noted $\rightarrow_{\text{SIAM}}$, as we now explain them—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 judgement 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 judgement assigns type $S \rightarrow A$ to $\lambda x.t$, and the type context is $S \rightarrow \mathbb{B}$, that is, it selects an occurrence of \star in the target type $A = \mathbb{B}(\star)$. The transition then simply moves to the judgement above, stripping down the type context to \mathbb{B} , 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: $[\uparrow]$ simply denotes the right premise 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 $S \rightarrow A'$ isolating \star inside the i -th type A in S . The transition then moves to the final judgement of the i -th derivation in 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 12.1. Consider a T- λ rule occurrence whose right-hand type of the conclusion is $S \rightarrow A'$. The premise has shape $\Gamma, x : S \vdash t : A'$, and by the lemma there is a bijection between the sequence of linear types in S and the axioms on x , respecting the order in S . The left side of \rightarrow_{bt2} focuses on the i -th type A in S and the SIAM moves to the judgement 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 $T\text{-}\lambda_{\star}$. Accordingly, when the SIAM reaches one of these rules it is in a final state. Exactly as the λIAM , the SIAM is bi-deterministic.

PROPOSITION 13.1. *The SIAM is bi-deterministic for each type derivation $\pi \triangleright \vdash t : \star$.*

An example. We present below the very same example analyzed in Section 3. 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.

$$\begin{array}{c}
 \frac{x : [[\star] \rightarrow \star] \vdash x : [\star\downarrow_{16}] \rightarrow \star\uparrow_6 \quad y : [\star] \vdash y : \star\uparrow_{17}}{y : [\star], x : [[\star] \rightarrow \star] \vdash xy : \star\uparrow_5} \\
 \frac{y : [\star] \vdash \lambda x.x y : [[\star\uparrow_{15}] \rightarrow \star\downarrow_7] \rightarrow \star\uparrow_4}{\vdash \lambda y.\lambda x.x y : [\star\downarrow_{18}] \rightarrow [[\star\uparrow_{14}] \rightarrow \star\downarrow_8] \rightarrow \star\uparrow_3} \quad \frac{}{\vdash I : \star\uparrow_{19}} \quad \frac{z : [\star] \vdash z : \star\uparrow_{11}}{\vdash \lambda z.z : [\star\downarrow_{12}] \rightarrow \star\uparrow_{10}} \\
 \hline
 \vdash (\lambda y.\lambda x.x y)I : [[\star\uparrow_{13}] \rightarrow \star\downarrow_9] \rightarrow \star\uparrow_2 \quad \vdash (\lambda z.z) : \star\uparrow_1 \\
 \hline
 \vdash (\lambda y.\lambda x.x y)I(\lambda z.z) : \star\uparrow_1
 \end{array}$$

One can immediately notice that every occurrence of \star is visited exactly once. Moreover, the sequence of the visited subterms is the same as the one obtained in the example of Section 3.

14 $\lambda\text{-IAM}$ TIME VIA EXHAUSTING SEQUENCE TYPES

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

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

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

In the Appendix F we provide the execution of the term $(\lambda x.xx)I$, that actually duplicates the sub-term I , for all the machines presented in the paper.

Relating Logs and Tapes with Typed Positions. In the λIAM , the log $L = l_1 \cdot \dots \cdot l_n$ has a logged position for every argument u_1, \dots, u_n in which the position of the current state is contained. In the SIAM this is simply given by the sub-derivations for u_1, \dots, u_n in which the current judgement occurrence J is contained—the way in which l_k identifies a copy of u_k in the λIAM corresponds on the type derivation π to the index i of the sub-derivation (in the sequence of sub-derivations) typing u_k in which J is located. Note that the λIAM manipulates the log only via transitions \rightarrow_{arg} and \rightarrow_{bt1} , that on the SIAM 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. 8, it is evident that there is a logged position on the λ IAM tape for every type sequence S in which it lies the hole $\langle \cdot \rangle$ of the current type context \mathbb{B} of the SIAM.

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

PROPOSITION 14.1. *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 SIAM states on π and λ IAM states on t .*

Weights and the Length of SIAM Runs via Acyclicity. We now turn to the proof of the correctness of the weight assignment $\mathbf{W}_{\lambda\text{IAM}}(\pi)$, that is, the fact that it correctly measures the length of λ IAM complete runs. While the weight assignment for the λ IAM is similar to de Carvalho's one for the KAM, the proof of its correctness is completely different, and it must be, as we know explain.

The KAM performs an evaluation that essentially mimics cut-elimination and so the number of KAM transitions to normal form is obtained via a refined, quantitative form of subject reduction. One may say that it is obtained in a *step-by-step* manner. The λ IAM, instead, does not mimic subject reduction. It walks over the type derivation *without ever changing it*, potentially passing many times over the same judgement (because of backtracking). Correctness of weights cannot then be obtained via a refined subject reduction property, because the reduced derivation gives rise to a different run, and not to a sub-run. It must instead follow from a *global* analysis of a fixed derivation, that we now develop. The proof technique is an original contribution of this paper.

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

Note a subtlety: by the bisimulation with the λ IAM (Prop. 14.1) we know that the run of the SIAM terminates, but we do not know whether it reaches all states. What we have to prove, then, is that there are no unreachable loops, that is, loops that are not reachable from an initial state. The next easy lemma guarantees that this is enough.

LEMMA 14.2. *Let 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.*

We show the absence of loops using a sort of subject reduction property. We first show that if the SIAM loops on $\pi \triangleright \vdash t : \star$ and $t \rightarrow_{\text{wh}} u$, then there is a type derivation $\pi' \triangleright \vdash u : \star$ on which the SIAM loops—that is, SIAM looping is preserved by reduction of the underlying term. This is done by defining a relation \blacktriangleright between the SIAM states on π and on π' —see Appendix E.3.

PROPOSITION 14.3. \blacktriangleright *is a loop-preserving bisimulation between SIAM states.*

Then, by the trivial fact that the SIAM does not loop on \rightarrow_{wh} -normal terms (as they are typed using just one rule, namely $T\text{-}\lambda_{\star}$), we obtain that it never loops.

COROLLARY 14.4. *Let $\pi \triangleright \vdash t : \star$ be a type derivation. Then the SIAM does not loop on π .*

The correctness of the weights for the length of SIAM runs immediately follows, and, via the strong bisimulation in Prop. 14.1, it transfers to the λ IAM.

THEOREM 14.5 (λ IAM TIME VIA SEQUENCE TYPES). *Let t be a closed term that is \rightarrow_{wh} -normalizable, σ the complete λ IAM run from s_t , and $\pi \triangleright t : \star$ a type derivation for t . Then $|\sigma| = \mathbf{W}_{\lambda\text{IAM}}(\pi)$.*

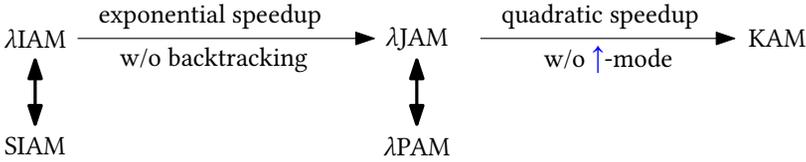
15 OUR TWO CENTS ABOUT SPACE

Here we provide an interesting example about space usage, with the only purpose of stressing that the situation is subtler than for time. Among the machines we have presented, the λ IAM is the only one tuned for space efficiency, as shown by the literature [Dal Lago and Schöpp 2010; Ghica 2007; Ghica and Smith 2010; Mazza 2015; Mazza and Terui 2015; Schöpp 2007]. In fact, the space used by the λ JAM (thus the λ PAM) and the KAM is proportional to their time, *i.e.* their space usage is inflationary. Nonetheless, there are terms for which the λ JAM outperforms the λ IAM in space consumption, showing that the space relationship between the λ IAM and the λ JAM is less smooth than the time one.

PROPOSITION 15.1. *Let r_k^h be defined as $r_k^h := (\lambda x_1 \dots \lambda x_k . \lambda y . y (\lambda z_1 \dots \lambda z_h . \lambda z . z)) t_1 \dots t_k (\lambda w . w u_1 \dots u_h)$. The the λ IAM space consumption for the evaluation of r_k^h is 2 logged positions plus $h + k$ occurrences of \bullet , while the λ JAM needs 2 logged positions plus $\max\{h, k + 1\}$ occurrences of \bullet .*

16 CONCLUSIONS

In this paper, we analysed the relative time performances of three game machines, namely the IAM the JAM and the PAM, establishing a series of results which can be summarized as follows:



Here, thicker arrows represent a stronger correspondence, *i.e.*, the λ PAM and λ JAM are isomorphic, the λ JAM improves the λ IAM with a possibly exponential advantage, while the KAM improves the λ JAM (thus the λ PAM) with a quadratic advantage.

Besides settling the question about the relative efficiency of the main game machines, we also prove non-idempotent intersection types to be able to precisely characterize the time performance of the λ IAM when run on the typed term, in analogy with the classic results on environment machines by de Carvalho [2018]. This way, the time behavior of two heterogeneous machines, namely the KAM and the λ IAM, on a given normalizing term t can be captured by just comparing two different ways of weighting the same sequence type derivation, the former attributing weight 1 to any instance rule in the type derivation, the latter taking into account the size of the underlying type in an essential way. In other words, *the bigger the types, the more inefficient the λ IAM*.

Among the topics for future work, we can certainly mention the extension of the results obtained here to *call-by-value* game machines, which seems within reach. A study on the relative *space* efficiency of game machines is more elusive, as the partial results in Section 15 show.

ACKNOWLEDGMENTS

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

- Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. 2000. Full Abstraction for PCF. *Inf. Comput.* 163, 2 (2000), 409–470. <https://doi.org/10.1006/inco.2000.2930>
- Beniamino Accattoli. 2018. (In)Efficiency and Reasonable Cost Models. *Electr. Notes Theor. Comput. Sci.* 338 (2018), 23–43.

- Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. 2014. Distilling abstract machines. In *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming, Gothenburg, Sweden, September 1-3, 2014*, Johan Jeuring and Manuel M. T. Chakravarty (Eds.). ACM, 363–376. <https://doi.org/10.1145/2628136.2628154>
- Beniamino Accattoli and Bruno Barras. 2017. Environments and the complexity of abstract machines. In *Proceedings of the 19th PPDP*. 4–16.
- Beniamino Accattoli, Andrea Condoluci, Giulio Guerrieri, and Claudio Sacerdoti Coen. 2019a. Crumbling Abstract Machines. In *Proceedings of the 21st PPDP*. 4:1–4:15.
- Beniamino Accattoli and Ugo Dal Lago. 2012. On the Invariance of the Unitary Cost Model for Head Reduction. In *23rd International Conference on Rewriting Techniques and Applications (RTA'12)*, RTA 2012, May 28 - June 2, 2012, Nagoya, Japan. 22–37. <https://doi.org/10.4230/LIPIcs.RTA.2012.22>
- Beniamino Accattoli and Ugo Dal Lago. 2016. (Leftmost-Outermost) Beta Reduction is Invariant, Indeed. *Logical Methods in Computer Science* 12, 1 (2016).
- Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. 2020a. The Machinery of Interaction. In *Proceedings of the 22nd PPDP*. Article 4.
- Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. 2020b. Tight typings and split bounds, fully developed. *J. Funct. Program.* 30 (2020), e14.
- Beniamino Accattoli and Giulio Guerrieri. 2018. Types of Fireballs. In *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings (Lecture Notes in Computer Science)*, Sukyoung Ryu (Ed.), Vol. 11275. Springer, 45–66. https://doi.org/10.1007/978-3-030-02768-1_3
- Beniamino Accattoli and Giulio Guerrieri. 2019. Abstract machines for Open Call-by-Value. *Sci. Comput. Program.* 184 (2019). <https://doi.org/10.1016/j.scico.2019.03.002>
- Beniamino Accattoli, Giulio Guerrieri, and Maico Leberle. 2019b. Types by Need. In *Proceedings of the 28th ESOP*. 410–439.
- Federico Aschieri. 2017. Game Semantics and the Geometry of Backtracking: a New Complexity Analysis of Interaction. *J. Symb. Log.* 82, 2 (2017), 672–708. <https://doi.org/10.1017/jsl.2016.48>
- Andrea Asperti, Vincent Danos, Cosimo Laneve, and Laurent Regnier. 1994. Paths in the lambda-calculus. In *Proceedings of LICS '94*. 426–436.
- Hendrik Pieter Barendregt. 1984. *The lambda calculus: its syntax and semantics*. North-Holland.
- Alexis Bernadet and Stéphane Graham-Lengrand. 2013. Non-idempotent intersection types and strong normalisation. *Logical Methods in Computer Science* 9, 4 (2013).
- Guy E. Blelloch and John Greiner. 1995. Parallelism in Sequential Functional Languages. In *FPCA*. 226–237.
- Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. 2017. Non-idempotent intersection types for the Lambda-Calculus. *Logic Journal of the IGPL* 25, 4 (2017), 431–464.
- Pierre Clairambault. 2011. Estimation of the Length of Interactions in Arena Game Semantics. In *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings (Lecture Notes in Computer Science)*, Martin Hofmann (Ed.), Vol. 6604. Springer, 335–349. https://doi.org/10.1007/978-3-642-19805-2_23
- Pierre Clairambault. 2015. Bounding linear head reduction and visible interaction through skeletons. *Log. Methods Comput. Sci.* 11, 2 (2015). [https://doi.org/10.2168/LMCS-11\(2:6\)2015](https://doi.org/10.2168/LMCS-11(2:6)2015)
- Mario Coppo and Mariangiola Dezani-Ciancaglini. 1978. A new type assignment for λ -terms. *Arch. Math. Log.* 19, 1 (1978), 139–156. <https://doi.org/10.1007/BF02011875>
- Mario Coppo and Mariangiola Dezani-Ciancaglini. 1980. An extension of the basic functionality theory for the λ -calculus. *Notre Dame Journal of Formal Logic* 21, 4 (1980), 685–693. <https://doi.org/10.1305/ndjfl/1093883253>
- Pierre-Louis Curien and Hugo Herbelin. 1998. Computing with Abstract Böhm Trees. In *Proceedings of the 3rd FLOPS*.
- Pierre-Louis Curien and Hugo Herbelin. 2007. Abstract machines for dialogue games. (2007). arXiv:0706.2544 <http://arxiv.org/abs/0706.2544>
- Ugo Dal Lago, Claudia Faggian, Ichiro Hasuo, and Akira Yoshimizu. 2014. The geometry of synchronization. In *Proceedings of CSL-LICS '14*. 35:1–35:10.
- Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. 2015. Parallelism and Synchronization in an Infinitary Context. In *Proceedings of the 30th LICS*. 559–572.
- Ugo Dal Lago and Ulrich Schöpp. 2010. Functional Programming in Sublinear Space. In *Proc. of ESOP 2010*. 205–225. https://doi.org/10.1007/978-3-642-11957-6_12
- Ugo Dal Lago and Ulrich Schöpp. 2016. Computation by interaction for space-bounded functional programming. *Information and Computation* 248 (2016), 150–194.
- Ugo Dal Lago, Ryo Tanaka, and Akira Yoshimizu. 2017. The geometry of concurrent interaction: Handling multiple ports by way of multiple tokens. In *Proceedings of the 32nd LICS*. 1–12.

- Vincent Danos, Hugo Herbelin, and Laurent Regnier. 1996. Game Semantics & Abstract Machines. In *Proceedings of the 11th LICS*. 394–405.
- V. Danos and L. Regnier. 1993. Local and asynchronous beta-reduction (an analysis of Girard’s execution formula). In *Proceedings of the 8th LICS*. 296–306.
- V. Danos and L. Regnier. 1995. *Proof-nets and the Hilbert space*. Cambridge University Press, 307–328.
- Vincent Danos and Laurent Regnier. 1999. Reversible, irreversible and optimal lambda-machines. *Theoretical Computer Science* 227, 1 (1999), 79–97.
- Vincent Danos and Laurent Regnier. 2004. *Head Linear Reduction*. Technical Report.
- Daniel de Carvalho. 2007. *Sémantiques de la logique linéaire et temps de calcul*. Thèse de Doctorat. Université Aix-Marseille II.
- Daniel de Carvalho. 2018. Execution time of λ -terms via denotational semantics and intersection types. *Math. Str. in Comput. Sci.* 28, 7 (2018), 1169–1203.
- Daniel de Carvalho, Michele Pagani, and Lorenzo Tortora de Falco. 2011. A semantic measure of the execution time in linear logic. *Theoretical Computer Science* 412, 20 (2011), 1884–1902.
- Maribel Fernández and Ian Mackie. 2002. Call-by-Value lambda-Graph Rewriting Without Rewriting. In *Proceedings of the 1st ICGT*. 75–89.
- Philippa Gardner. 1994. Discovering Needed Reductions Using Type Theory. In *Theoretical Aspects of Computer Software (TACS '94) (Lecture Notes in Computer Science)*, Vol. 789. 555–574. https://doi.org/10.1007/3-540-57887-0_115
- Dan R. Ghica. 2007. Geometry of Synthesis: A Structured Approach to VLSI Design. In *Proceedings of the 34th POPL*. 363–375.
- Dan R. Ghica and Alex I. Smith. 2010. 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*. 301–324. <https://doi.org/10.1016/j.entcs.2010.08.018>
- Jean-Yves Girard. 1987. Linear logic. *Theoretical Computer Science* 50, 1 (1987), 1–101.
- Jean-Yves Girard. 1989. Geometry of Interaction I: Interpretation of System F. In *Studies in Logic and the Foundations of Mathematics*, R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo (Eds.). Vol. 127. Elsevier, 221–260.
- Naohiko Hoshino, Koko Muroya, and Ichiro Hasuo. 2014. Memoryful Geometry of Interaction: From Coalgebraic Components to Algebraic Effects (*Proceedings of CSL-LICS '14*). ACM, 52:1–52:10.
- J. M. E. Hyland and C.-H. Luke Ong. 2000. On Full Abstraction for PCF: I, II, and III. *Inf. Comput.* 163, 2 (2000), 285–408. <https://doi.org/10.1006/inco.2000.2917>
- Delia Kesner and Pierre Vial. 2020. Consuming and Persistent Types for Classical Logic. In *Proceedings of the 35th LICS*. 619–632.
- Assaf J. Kfoury. 2000. A linearization of the Lambda-calculus and consequences. *Journal of Logic and Computation* 10, 3 (2000), 411–436. <https://doi.org/10.1093/logcom/10.3.411>
- Neelakantan R. Krishnaswami, Nick Benton, and Jan Hoffmann. 2012. Higher-order functional reactive programming in bounded space. In *Proc. of POPL 2012*. 45–58. <https://doi.org/10.1145/2103656.2103665>
- Jean-Louis Krivine. 1993. *Lambda-calculus, types and models*. Masson.
- Jean-Louis Krivine. 2007. A Call-by-name Lambda-calculus Machine. *Higher Order Symbol. Comput.* 20, 3 (2007), 199–207.
- Olivier Laurent. 2001. A Token Machine for Full Geometry of Interaction. In *Proceedings of the 5th TLCA*. 283–297.
- Jean-Jacques Lévy. 1978. *Réductions correctes et optimales dans le lambda-calcul*. PhD Thesis. Université Paris 7.
- Ian Mackie. 1995. The Geometry of Interaction Machine. In *Proceedings of the 22nd POPL*. 198–208.
- Ian Mackie. 2017. A Geometry of Interaction Machine for Gödel’s System T. In *Proceedings of the 24th WoLLIC*. 229–241.
- Damiano Mazza. 2015. Simple Parsimonious Types and Logarithmic Space. In *Proceedings of the 24th CSL*. 24–40.
- Damiano Mazza, Luc Pellissier, and Pierre Vial. 2018. Polyadic Approximations, Fibrations and Intersection Types. *Proceedings of the ACM on Programming Languages* 2, POPL:6 (2018).
- Damiano Mazza and Kazushige Terui. 2015. Parsimonious Types and Non-uniform Computation. In *Proceedings of the 42nd ICALP*. 350–361.
- Robin Milner. 1977. Fully Abstract Models of Typed lambda-Calculi. *Theor. Comput. Sci.* 4, 1 (1977), 1–22. [https://doi.org/10.1016/0304-3975\(77\)90053-6](https://doi.org/10.1016/0304-3975(77)90053-6)
- Koko Muroya and Dan R. Ghica. 2017. The Dynamic Geometry of Interaction Machine: A Call-by-Need Graph Rewriter. In *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*. 32:1–32:15. <https://doi.org/10.4230/LIPIcs.CSL.2017.32>
- Koko Muroya and Dan R. Ghica. 2019. The Dynamic Geometry of Interaction Machine: A Token-Guided Graph Rewriter. *Log. Methods Comput. Sci.* 15, 4 (2019). <https://lmcs.episciences.org/5882>
- Peter Möller Neergaard and Harry G. Mairson. 2004. Types, potency, and idempotency: why nonlinearity and amnesia make a type system work. In *Proceedings of the 9th ICFP*. 138–149.
- Marco Pedicini and Francesco Quaglia. 2007. PELCR: Parallel environment for optimal lambda-calculus reduction. *ACM Trans. Comput. Log.* 8, 3 (2007), 14. <https://doi.org/10.1145/1243996.1243997>

- Jorge Sousa Pinto. 2001. Parallel Implementation Models for the lambda-Calculus Using the Geometry of Interaction. In *Proc. of TLCA 2001 (LNCS)*, Samson Abramsky (Ed.), Vol. 2044. Springer, 385–399. https://doi.org/10.1007/3-540-45413-6_30
- Garrel Pottinger. 1980. A Type Assignment for The Strongly Normalizable λ -terms. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, J.P. Seldin and J.R. Hindley (Eds.). Academic Press, 561–578.
- David Sands, Jörgen Gustavsson, and Andrew Moran. 2002. Lambda Calculi and Linear Speedups. In *The Essence of Computation*. 60–84.
- Ulrich Schopp. 2007. Stratified Bounded Affine Logic for Logarithmic Space. In *Proc. of LICS 2007*. 411–420.
- Ulrich Schöpp. 2014. On the Relation of Interaction Semantics to Continuations and Defunctionalization. *Logical Methods in Computer Science* 10, 4 (2014).
- Ulrich Schöpp. 2015. From Call-by-Value to Interaction by Typed Closure Conversion. In *Proc. of PPDP 2015 (LNCS)*, Vol. 9458. Springer, 251–270.
- Takeshi Tsukada, Kazuyuki Asada, and C.-H. Luke Ong. 2017. Generalised species of rigid resource terms. In *Proceedings of the 32nd LICS*. 1–12.

A PROOF OF SECT. 7 (λ -IAM AND THE λ -JAM: JUMPING IS EXHAUSTING)

First of all, we have to finally define log tests.

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

Let (u, C_{n+1}) be a position. Then, for every decomposition of n into two natural numbers m, k with $m + k = n$, we can find contexts C_m and C_k such that $t = C_m \langle r C_k \langle u \rangle \rangle$. Then, the $m + 1$ -outer context of the position (u, C_{n+1}) is the context $O_{m+1} := C_m \langle r \langle \cdot \rangle \rangle$ of level $m + 1$ and the $m + 1$ -outer position is $(C_k \langle u \rangle, O_{m+1})$.

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

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

We also need to recall a lemma about log tests from [Accattoli et al. 2020a], to be used in the proof of the I-exhaustible invariant.

LEMMA A.2 (INVARIANCE PROPERTIES OF LOG TESTS). *Let $s = (t, C_n, L_n, T, d)$ be a λ IAM state. Then:*

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

LEMMA A.3 (I-EXHAUSTIBLE INVARIANT). *Let t be a closed term and $\pi : s_t \rightarrow_{\lambda\text{JAM}}^k s$ a λJAM run. Then s is I-exhaustible.*

PROOF. By induction on k . For $k = 0$ there is nothing to prove because the tape has no logged positions (so it does not decompose) and s has no outer state. Then suppose $\pi' : s_0 \rightarrow_{\lambda\text{IAM}}^{k-1} s'$ and that the run continues with $s' \rightarrow_{\lambda\text{JAM}} s$. By *i.h.*, s' is I-exhaustible.

Terminology: when a test state satisfies the clause in the definition of I-exhaustible states we say that it is *positive*.

There are many cases to take into account, depending on the transition used to move from s' to s —the cases for \bullet are rather trivial, the other ones instead are subtle, the subtlest one being the jump, that is, transition \rightarrow_{jmp} (it is the last case). First, suppose that $d = \downarrow$. Cases of $s' \rightarrow_{\lambda\text{IAM}} s$:

- (1) Transition $\rightarrow_{\bullet 1}$:

$$s' = (\underline{r}w, C, L, T) \rightarrow_{\bullet 1} (\underline{r}, C \langle \langle \cdot \rangle w \rangle, L, \bullet \cdot T) = s.$$

- *Log tests.* Positivity of log tests follows from Lemma A.2.3 and the *i.h.*: s' is a head translation of s , and the lemma states that they have the same log tests, which are positive because s' is I-exhaustible by *i.h.*
- *Tape tests.* The direction is \downarrow and by Lemma 4.1, the tape of s has no logged positions, and so there are not tape tests.

(2) Transition $\rightarrow_{\bullet 2}$:

$$s' = (\underline{\lambda x.r}, C, L, \bullet \cdot T) \rightarrow_{\bullet 2} (\underline{r}, C\langle \lambda x.\langle \cdot \rangle \rangle, L, T) = s$$

Exactly as the previous case.

(3) Transition \rightarrow_{var} :

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

• *Log tests.* By Lemma A.2.4, all log tests of s are also log tests of s' . Since the latter is I-exhaustible by *i.h.*, then all these tests are positive.

• *Tape tests.* Let $l := (x, C\langle \lambda x.D_n \rangle, L_n \cdot L)$. The only tape state of s is $s_l := (\underline{\lambda x.D_n\langle x \rangle}, C, L, l)$ and the one-step run

$$\begin{aligned} \sigma : I(s_l) &= (\underline{\lambda x.D_n\langle x \rangle}, C, I(L), I(l)) \\ &= (\underline{\lambda x.D_n\langle x \rangle}, C, I(L), (x, \lambda x.D_n, I(L_n))) \\ &\xrightarrow{\text{bt2}, I(l)} (x, \underline{C\langle \lambda x.D_n \rangle}, I(L_n) \cdot I(L), \epsilon) \\ &= I(x, \underline{C\langle \lambda x.D_n \rangle}, L_n \cdot L, \epsilon) = I(l^\circ) \end{aligned}$$

exhausts l as required. Now, we prove that l° is I-exhaustible. Note that l° is s' with empty tape, so they have the same log tests, which are positive because s' is I-exhaustible by *i.h.*, and l° has no tape test.

Now, suppose that $d = \uparrow$. Cases of $s' \rightarrow_{\lambda\text{IAM}} s$:

(1) Transition $\rightarrow_{\bullet 3}$:

$$s' = (u, \underline{D\langle \cdot \rangle r}, L, \bullet \cdot T) \rightarrow_{\bullet 3} (ur, \underline{D}, L, T) = s.$$

(a) *Log tests.* Positivity of log tests follows from Lemma A.2.3 and the *i.h.*: s' is a head translation of s , and the lemma states that they have the same log tests, which are positive because s' is I-exhaustible by *i.h.*

(b) *Tape tests.* The direction of s is \uparrow and by Lemma 4.1, the tape of s has exactly one logged positions l , and so just one tape test s_l . Note that s' also has a tape test s'_l and that by *i.h.* it is positive, that is, there is a run $\sigma : I(s'_l) \xrightarrow{\lambda\text{IAM}}^* \xrightarrow{\text{bt2}, I(l)} I(l^\circ)$ with l° I-exhaustible. Since the direction of s_l and s'_l is \downarrow , we have a run $\rho : I(s_l) \rightarrow_{\bullet 1} I(s'_l) \xrightarrow{\lambda\text{IAM}}^* \xrightarrow{\text{bt2}, I(l)} I(l^\circ)$ prefixing σ with a step and exhausting s_l .

(2) Transition $\rightarrow_{\bullet 4}$

$$s' = (u, \underline{D\langle \lambda x.\langle \cdot \rangle \rangle}, L, T) \rightarrow_{\bullet 4} (\lambda x.u, \underline{D}, L, \bullet \cdot T) = s.$$

This case is exactly as the previous one.

(3) Transition \rightarrow_{arg} :

$$s' = (u, \underline{D\langle \langle \cdot \rangle r \rangle}, L, l \cdot T) \rightarrow_{\text{arg}} (\underline{r}, D\langle u\langle \cdot \rangle \rangle, l \cdot L, T) = s.$$

(a) *Log tests.* The log tests of s are those of s' plus $s_l = (r, \underline{D\langle u\langle \cdot \rangle \rangle}, l \cdot L, \epsilon)$. The former are positive because of the *i.h.*, while about the latter, observe that

$$\begin{aligned} I(s_l) &= I(r, \underline{D\langle u\langle \cdot \rangle \rangle}, l \cdot L, \epsilon) \\ &= (r, \underline{D\langle u\langle \cdot \rangle \rangle}, I(l) \cdot I(L), \epsilon) \\ &\xrightarrow{\text{bt1}} (\underline{u}, \underline{D\langle \langle \cdot \rangle r \rangle}, I(L), I(l)) \\ &= I(\underline{u}, \underline{D\langle \langle \cdot \rangle r \rangle}, L, l) = s'_l. \end{aligned} \tag{1}$$

Note that s'_l is a tape test of s' . By *i.h.*, there is a run $\sigma : I(s'_l) \xrightarrow{\lambda\text{IAM}}^* \xrightarrow{\text{bt2}, I(l)} I(l^\circ)$ such that l° is I-exhaustible. Now, the run for the test of interest is $\rho : I(s_l) \rightarrow_{\lambda\text{IAM}} I(s'_l) \xrightarrow{\lambda\text{IAM}}^* \xrightarrow{\text{bt2}, I(l)} I(l^\circ)$, obtained by prefixing σ with the step in (1).

- (b) *Tape tests.* The direction is \downarrow of s and by Lemma 4.1, the tape of s has no logged positions, and so there are not tape tests for of s .
- (4) Transition \rightarrow_{jmp} :

$$s' = (u, \underline{D\langle r\langle \cdot \rangle \rangle}, (x, C, L') \cdot L, T) \rightarrow_{\text{jmp}} (x, \underline{C}, L', T) = s.$$

Let $l := (x, C, L')$.

- (a) *Log tests:* By *i.h.*, s' is I-exhaustible, and since $s'_l = (u, \underline{D\langle r\langle \cdot \rangle \rangle}, (x, C, L') \cdot L, \epsilon)$ is a log test of s' , then it is positive and there exist a run

$$\sigma : I(s'_l) \xrightarrow{*}_{\lambda\text{IAM}} \rightarrow_{\text{bt2}, I(l)} I(l^\circ)$$

where l° is I-exhaustible. By Lemma A.2.2, s and l° have the same log tests, which are then positive.

- (b) *Tape tests.* Since the direction of s is \uparrow , by Lemma 4.1 $|T|_l = 1$, there is only one possible decomposition: $T = T' \cdot l \cdot T''$. Then the only tape test of s is

$$s_l = (\underline{x}, C, L', T' \cdot l)$$

and the only tape test of s' is

$$s'_l = (\underline{u}, D\langle r\langle \cdot \rangle \rangle, (x, C, L') \cdot L, T' \cdot l)$$

that by *i.h.* is positive and so there is a run $\sigma : I(s'_l) \xrightarrow{*}_{\lambda\text{IAM}} \rightarrow_{\text{bt2}, I(l)} I(l^\circ)$ with l° I-exhaustible.

Now, we show that $I(s_l) \xrightarrow{+}_{\lambda\text{IAM}} I(s'_l)$, that proves the positivity of the tape tests, using an argument analogous to the one for the log tests. Let $l' := (x, C, L')$ and consider the state $s'_{l'} := (u, \underline{D\langle r\langle \cdot \rangle \rangle}, l' \cdot L, \epsilon)$, that it is a log test of s' . By *i.h.*, it is positive, thus there is a run $\rho : I(s'_{l'}) \xrightarrow{*}_{\lambda\text{IAM}} \rightarrow_{\text{bt2}, I(l')} I(l'^\circ)$. By reversibility, we obtain a run $\rho' : I(l'^\circ)^\perp \xrightarrow{+}_{\lambda\text{IAM}} I(s'_{l'})^\perp$, where $^\perp$ is the operation on states that changes the direction. Explicitly, we have:

$$\sigma' : I(l'^\circ)^\perp = I(\underline{x}, C, L', \epsilon) \xrightarrow{+}_{\lambda\text{IAM}} I(\underline{u}, D\langle r\langle \cdot \rangle \rangle, (x, C, L') \cdot L, \epsilon) = I(s'_{l'})^\perp$$

By Lemma 3.3, we can lift the run to states extended with the tape $T' \cdot l$, obtaining:

$$\rho'' : I(s_l) = I(\underline{x}, C, L', T' \cdot l) \xrightarrow{+}_{\lambda\text{IAM}} I(\underline{u}, D\langle r\langle \cdot \rangle \rangle, (x, C, L') \cdot L, T' \cdot l) = I(s'_l)$$

The run $I(s_l) \xrightarrow{+}_{\lambda\text{IAM}} I(l^\circ)$ obtained by concatenating ρ'' and σ exhausts s_l .

□

THEOREM A.4 (λIAM AND λJAM RELATIONSHIP).

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

PROOF.

- (1) We proceed by induction on the length of π_J . If $|\pi_J| = 0$ there is nothing to prove. Now, let us consider $\pi_J : s_t \xrightarrow{*}_{\lambda\text{JAM}} s' \rightarrow_{\lambda\text{JAM}} s$. Considering the property true for the reduction $\sigma_J : s_t \xrightarrow{*}_{\lambda\text{JAM}} s'$, we prove that it is true for π_J . In particular, there exists a reduction $\sigma_I : I(s_t) \xrightarrow{*}_{\lambda\text{IAM}} I(s')$ such that $|\sigma_I| \geq |\sigma_J|$ and $|\sigma_I|_{\text{var}} \geq |\sigma_J|_{\text{var}}$. We proceed considering all the possible transitions from s' to s .

- Transitions $\rightarrow_{\bullet 1}, \rightarrow_{\bullet 2}, \rightarrow_{\bullet 3}, \rightarrow_{\bullet 4}, \rightarrow_{\text{arg}}$. This group of transitions behaves identically, modulo $I(\cdot)$ in the two machines. Then $|\pi_J| = 1 + |\sigma_J| \leq_{i.h.} 1 + |\sigma_I| = |\pi_I|$ and $|\pi_J|_{\text{var}} = |\sigma_J|_{\text{var}} \leq_{i.h.} |\sigma_I|_{\text{var}} = |\pi_I|_{\text{var}}$.
 - Transition \rightarrow_{var} . This transition behaves identically, modulo $I(\cdot)$, in the two machines. Therefore, $|\pi_J| = 1 + |\sigma_J| \leq_{i.h.} 1 + |\sigma_I| = |\pi_I|$, and exactly the same sequence of (in)equalities holds with respect to $|\cdot|_{\text{var}}$.
 - Transition \rightarrow_{jmp} . This is the only non trivial case. If $s \rightarrow_{\text{jmp}, l} s'$ then by the simulation of jumps via backtracking (Lemma 7.3) we have a run $\rho_I : I(s) \rightarrow_{\text{bt}1, I(l)}^* \xrightarrow{\lambda\text{IAM}} \xrightarrow{\text{bt}2, I(l)} I(s')$. Then we define π_I as σ_I followed by ρ_I , so that $|\pi_J| = 1 + |\sigma_J| \leq_{i.h.} 1 + |\sigma_I| < |\rho_I| + |\sigma_I| = |\pi_I|$ and $|\pi_J|_{\text{var}} = |\sigma_J|_{\text{var}} \leq_{i.h.} |\sigma_I|_{\text{var}} \leq |\rho_I|_{\text{var}} + |\sigma_I|_{\text{var}} = |\pi_I|_{\text{var}}$.
- (2) By induction on the length of π_I . If $|\pi_I|_t = 0$ there is nothing to prove. Now, let us consider $\pi_I : s_t \xrightarrow{\lambda\text{IAM}}^* s_1 \xrightarrow{\lambda\text{IAM}} s$. Considering the property true for the reduction $\pi'_I : s_t \xrightarrow{\lambda\text{IAM}}^* s_1$, we prove that it is true for π_I . By *i.h.*, there are runs $\pi'_J : s_t \xrightarrow{\lambda\text{JAM}}^* s_2$ and $\sigma'_I : s_1 \xrightarrow{\lambda\text{IAM}}^* I(s_2)$ such that $\pi'_I \sigma'_I = I(\pi'_I)$. If σ'_I is non-empty then by determinism of the λIAM we are done, because σ'_I has to pass through s and the suffix σ_I of σ'_I starting on s proves the statement. If σ'_I is empty then $s_1 = I(s_2)$. Then consider the cases of transition $s_1 \xrightarrow{\lambda\text{IAM}} s$:
- Transitions $\rightarrow_{\bullet 1}, \rightarrow_{\bullet 2}, \rightarrow_{\bullet 3}, \rightarrow_{\bullet 4}, \rightarrow_{\text{arg}}, \rightarrow_{\text{var}}$. The λJAM can do the same step and close the diagram, as these transitions behaves identically, modulo $I(\cdot)$, in the two machines.
 - Transition $\rightarrow_{\text{bt}2}$. Impossible because then the state $s_1 = I(s_2)$ would have direction \downarrow , have a logged position on the tape, and be the projection of a λJAM state—by the direction and tape invariant of the λJAM such states have no logged positions on the tape.
 - Transition $\rightarrow_{\text{bt}1}$. Then the λJAM can make a jump and we can close the diagram using the simulation of jumps via backtracking (Lemma 7.3), as in the previous point of the theorem.
- (3) The first two points of the theorem provide the proof that I is a bisimulation between the λIAM and the λJAM . Clearly, I preserves termination. □

Theorem A.4 immediately implies the following more concise statement given in the body of the paper (as Theorem 7.4).

COROLLARY A.5 (λIAM AND λJAM RELATIONSHIP). *There is a complete λJAM run π_J from t if and only if there is a complete λIAM run π_I from t . In particular, the λJAM implements Closed CbN. Moreover, $|\pi_J| \leq |\pi_I|$ and $|\pi_J|_{\text{var}} \leq |\pi_I|_{\text{var}}$.*

B PROOFS OF SECT. 9 (HOPPING IS ALSO EXHAUSTING)

LEMMA B.1 (HAM BASIC INVARIANTS). *Let $s = (t, C_n, L, E, T, d)$ be a HAM reachable state. Then*

- (1) Position and log: $(t, C_n, L)^E$ is a closed position, and
- (2) Tape and direction: if $d = \downarrow$, then T does not contain any closed positions, otherwise, if $d = \uparrow$, then T contains exactly one closed position.

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

Tape Tests. By the *tape and direction* invariant, there is exactly one closed position \hat{l} on the tape in direction \uparrow and none in direction \downarrow . Essentially, we test only the logged closures added to the tape in a \downarrow phase, which—in \uparrow -states—are those on the right of \hat{l} on the tape. Moreover, in a \uparrow -state we test them starting on \hat{l} (which records a \downarrow -state), not from the current position.

Definition B.2 (HAM Tape tests). Let $s = (t, C, L, E, T, d)$ be a HAM state. Tape tests of s are defined depending on whether there is a closed position \hat{l} on T , that is, on whether $|T|_{\hat{l}}$ is 1 or 0.

- If $d = \downarrow$ then $s_{\hat{c}} := (t, \underline{C}, L, E, T')$ is a tape test of s of focus \hat{c} for each decomposition $T = T' \cdot \hat{c} \cdot T''$ of the tape.
- If $d = \uparrow$ then $s_{\hat{c}} := (x, \underline{D}, L', E', T_1)$ is a tape test of s of focus \hat{c} for each decomposition $T = T' \cdot \hat{l} \cdot T_1 \cdot c \cdot T_2$ with $\hat{l} = (x, D, L')^{E'}$ of the tape.

LEMMA B.3 (INVARIANCE PROPERTIES OF HAM ENVIRONMENT TESTS). *Let $s = (t, C_n, L_n, E, T, d)$ be a state. Then:*

1. Direction: *the dual (t, C_n, L_n, E, T, d^1) of s induces the same environment tests;*
2. Tape: *the state (t, C_n, L_n, E, T', d) obtained from s replacing T with an arbitrary tape T' induces the same environment tests;*
3. Weak shift: *let weak contexts be defined by $W ::= \langle \cdot \rangle \mid Wu \mid uW$. Then*
 - (1) *if $t = W\langle r \rangle$, then for every L' and T' the state $(r, C_n\langle W \rangle, L', E, T', d)$ induces the same environment tests of s .*
 - (2) *if $C_n = D_h\langle E_k \rangle$ with E_k weak context, then for every L' and T' the state $(E_k\langle t \rangle, C_h, L', E, T', d)$ induces the same environment tests of s .*
4. Inclusion: *if $C_n = C_m\langle \lambda x.C_i \rangle$, $L_n = L_i \cdot L_m$ and $E = E' \cdot [x \leftarrow \hat{c}] \cdot E''$ then the environment tests of $(\lambda x.C_i\langle t \rangle, C_m, L_m, E'', T', d)$ are environment tests of s .*

As in Sect. 8, we consider the λ JAM and the KAM as special instances of the HAM. In particular the λ JAM always uses the $\rightarrow_{\text{varJ}}$ transition, while the KAM always the transition $\rightarrow_{\text{hop/varK}}$. This way, states can be compared without any kind of projection.

LEMMA B.4 (LOGGED CLOSURES AND CLOSED POSITIONS WERE VISITED). *Let $\pi : s_t \rightarrow_{\text{HAM}}^* s$.*

- (1) Logged closures: *if $\hat{c} = (u, C\langle t\langle \cdot \rangle \rangle, E)^L$ is a logged closure in s then π passes through a state $(\underline{t}, C\langle \langle \cdot \rangle u \rangle, L, E, T)$ for some tape T .*
- (2) Closed positions: *if $\hat{l} = (u, C, L)^E$ is a closed position in s then π passes through a state $(\underline{u}, C, L, E, T)$ for some tape T .*

PROOF. By induction on the length of π , together with an immediate inspection of the transitions using the *i.h.* \square

LEMMA B.5 (\uparrow -EXHAUSTIBLE INVARIANT). *Let s be a HAM reachable state. Then s is \uparrow -exhaustible.*

PROOF. By induction on k . For $k = 0$ there is nothing to prove because $s = s_t$ has no tests. Then suppose $s_t \xrightarrow{\text{HAM}}^{k-1} s' \xrightarrow{\text{HAM}} s$. By *i.h.* $s' = (u, C, L, E, T, d)$ is \uparrow -exhaustible, and with this hypothesis we need to conclude that s is \uparrow -exhaustible, too. There are many cases to take into account, depending on the transition used to move from s' to s .

Terminology: when a test satisfies the clause for tests in the definition of \uparrow -exhaustibility, we say that it is *positive*.

First, suppose that $d = \downarrow$. Cases of $s' \xrightarrow{\text{HAM}} s$:

- (1) Transition $\rightarrow_{\bullet 1/\text{app}}$:

$$s' = (\underline{tu}, C, L, E, T) \xrightarrow{\bullet 1/\text{app}} (\underline{t}, C\langle \langle \cdot \rangle u \rangle, L, E, (u, C\langle t\langle \cdot \rangle \rangle, E)^L \cdot T) = s$$

- *Environment tests.* It follows by the *i.h.*, since the environment tests of s are the same of those of s' .
- *Tape tests.* The first tape test of s is trivially positive since $s_{\hat{c}} = \hat{c}^\circ$. All other tape tests of s are in the form $s_{\hat{c}} = (t, \underline{C}\langle \langle \cdot \rangle u \rangle, L, E, (u, C\langle t\langle \cdot \rangle \rangle, E)^L \cdot T')$, where $T' \cdot \hat{c}$ is a prefix of T . Clearly

$$s_{\hat{c}} = (t, \underline{C}\langle \langle \cdot \rangle u \rangle, L, E, (u, C\langle t\langle \cdot \rangle \rangle, E)^L \cdot T') \xrightarrow{\bullet 3} (tu, \underline{C}, L, E, T') = s'_{\hat{c}}$$

and $s'_{\hat{c}}$ is a tape test for s' . By *i.h.*, $s'_{\hat{c}}$ is positive. Hence, since

$$s_{\hat{c}} \rightarrow_{\uparrow} s'_{\hat{c}}$$

also $s_{\hat{c}}$ is positive.

(2) Transition $\rightarrow_{\bullet 2/\text{abs}}$:

$$s' = (\underline{\lambda x.t}, C, L, E, \hat{c} \cdot T) \rightarrow_{\bullet 2/\text{abs}} (\underline{t}, C\langle \lambda x.\langle \cdot \rangle \rangle, L, [x \leftarrow \hat{c}] \cdot E, T) = s$$

- *Environment tests.* The environment tests of s are those of s' plus $s_{\hat{c}} := (\lambda x.t, \underline{C}, L, E, \epsilon)$. Note that $s_{\hat{c}}$ is also a tape test of s' , which by *i.h.* is positive.
- *Tape tests.* Each tape test of s is in the form $s_{\hat{c}'} = (t, \underline{C}\langle \lambda x.\langle \cdot \rangle \rangle, L, [x \leftarrow \hat{c}] \cdot E, T')$, where $T' \cdot \hat{c}'$ is a prefix of T . Clearly

$$s_{\hat{c}'} = (t, \underline{C}\langle \lambda x.\langle \cdot \rangle \rangle, L, [x \leftarrow \hat{c}] \cdot E, T') \rightarrow_{\bullet 4} (\lambda x.t, \underline{C}, L, E, \hat{c} \cdot T') = s'_{\hat{c}'}$$

and $s'_{\hat{c}'}$ is a tape test for s' . Thus, by *i.h.* $s'_{\hat{c}'}$ is positive, and so is $s_{\hat{c}'}$.

(3) Transition $\rightarrow_{\text{varj}}$:

$$(\underline{x}, C\langle \lambda x.D_n \rangle, L_n \cdot L, E' [x \leftarrow \hat{c}] E, T) \rightarrow_{\text{varj}} (\lambda x.D_n \langle x \rangle, \underline{C}, L, E, \hat{l} \cdot T)$$

where $\hat{l} := (x, C\langle \lambda x.D \rangle, L_n \cdot L)^{E' [x \leftarrow \hat{c}] E}$.

- *Environment tests.* It follows by the *i.h.*, since all the environment tests of s are environment tests of s' .
- *Tape tests.* It follows by the *i.h.*, since the tape tests of s are the same of those of s' .

(4) Transition $\rightarrow_{\text{hop/varK}}$:

$$s' = (\underline{x}, C, L, E, T) \rightarrow_{\text{hop/varK}} (\underline{u}, D\langle t \langle \cdot \rangle \rangle, (x, C, L)^E \cdot L', F, T) = s$$

where $E = E' \cdot [x \leftarrow (u, D\langle t \langle \cdot \rangle \rangle, F)^{L'}] \cdot E''$.

- *Environment tests.* By Lemma B.4, we have that the run π passed through a state $s'' := (\underline{t}, D\langle \langle \cdot \rangle u \rangle, L', F, T')$ for some T' . Note that s'' is a weak shift of s as defined in Lemma A.2.3, and so s'' and s have the same environment tests, which are then positive by *i.h.*
- *Tape tests.* Note that for each prefix $T' \cdot \hat{c}$ of T we have

$$s_{\hat{c}} = (u, \underline{D}\langle t \langle \cdot \rangle \rangle, (x, C, L)^E \cdot L', F, T') \rightarrow_{\text{jmp}} (x, \underline{C}, L, E, T') = s'_{\hat{c}}$$

and by *i.h.* $s'_{\hat{c}}$ is positive. Then $s_{\hat{c}}$ is positive.

Then, suppose that $d = \uparrow$. Cases of $s' \rightarrow_{\text{HAM}} s$:

(1) Transition $\rightarrow_{\bullet 3}$:

$$(t, \underline{C}\langle \langle \cdot \rangle u \rangle, L, E, \hat{c} \cdot T) \rightarrow_{\bullet 3} (tu, \underline{C}, L, E, T)$$

- *Environment tests.* It follows by the *i.h.*, because all the environment tests of s are environment tests of s' by Lemma A.2.3.
- *Tape tests.* By *i.h.* since the tape tests of s are the same of those of s' .

(2) Transition $\rightarrow_{\bullet 4}$:

$$(t, \underline{C}\langle \lambda x.\langle \cdot \rangle \rangle, L, [x \leftarrow \hat{c}] \cdot E, T) \rightarrow_{\bullet 4} (\lambda x.t, \underline{C}, L, E, \hat{c} \cdot T)$$

- *Environment tests.* It follows by the *i.h.*, because all the environment tests of s are environment tests of s' .
- *Tape tests.* By *i.h.* since the tape tests of s are the same of those of s' (\hat{c} appears on the left of the enriched logged position in the tape, and so needs not to be tested).

(3) Transition \rightarrow_{arg} :

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

- *Environment tests.* By *i.h.* since the environment tests of s are the same of those of s' by Lemma A.2.3.
- *Tape tests.* Since the direction of s is \downarrow , by the tape and direction invariant (Lemma B.1) there are no closed position on T , and the tape tests of s are in the form $s_{\hat{c}} := (u, \underline{C\langle t\langle \cdot \rangle \rangle}, \hat{l} \cdot L, E, T')$, where $T' \cdot \hat{c}$ is a prefix of T . If $\hat{l} = (x, D, L')^{E'}$, then

$$s_{\hat{c}} = (u, \underline{C\langle t\langle \cdot \rangle \rangle}, (x, D, L')^{E'} \cdot L, E, T') \rightarrow_{\text{jmp}} (x, \underline{D}, L', E', T').$$

Those states in the form $(x, \underline{D}, L', E', T')$ are exactly the tape tests $s'_{\hat{c}}$ of s' . Thus, by *i.h.* they are positive, and so are the tests $s_{\hat{c}}$.

(4) *Jumping.*

$$s' = (t, \underline{C\langle u\langle \cdot \rangle \rangle}, \hat{l} \cdot L, E, T) \rightarrow_{\text{jmp}} (x, \underline{D}, L', E', T) = s$$

where $\hat{l} = (x, D, L')^{E'}$.

- *Environment tests.* By Lemma B.4, the run π passes through a state $s'' := (\underline{x}, D, L', E', T')$ for some T' . Note that s'' and s differ only for direction and tape, and so by Lemma B.3 they have the same environment tests, which are positive by the *i.h.*
- *Tape tests.* It follows by the *i.h.*, since the tape tests of s are the same of those of s' .

□

THEOREM B.6 (λ JAM AND KAM RELATIONSHIP VIA THE HAM). *Let s_t be a HAM initial state.*

- (1) KAM to λ JAM: *for every run $\pi_K : s_t \rightarrow_{\text{HAM}_K}^* s$ there exists a run $J(\pi_K) : s_t \rightarrow_{\text{HAM}_J}^* s$ such that $|J(\pi_K)| = |\pi_K| + |J(\pi_K)|_{\uparrow}$ and $|J(\pi_K)|_{\text{var}_J} = |\pi_K|_{\text{hop/var}_K}$.*
- (2) λ JAM to KAM: *for every run $\pi_J : s_t \rightarrow_{\text{HAM}_J}^* s$ there exist a run $\pi_K : s_t \rightarrow_{\text{HAM}_K}^* s'$ and a run $\sigma_J : s \rightarrow_{\uparrow}^* s'$ such that $\pi_J \sigma_J = J(\pi_K)$.*
- (3) Termination: *$\rightarrow_{\text{HAM}_K}$ terminates if and only if $\rightarrow_{\text{HAM}_J}$ terminates.*

PROOF.

- (1) We proceed by induction on the length of π_K . If $|\pi_K| = 0$ there is nothing to prove. Now, let us consider $\pi_K : s_t \rightarrow_{\text{HAM}_K}^* s' \rightarrow_{\text{HAM}_K} s$. Considering the property true for the reduction $\sigma_K : s_t \rightarrow_{\text{HAM}_K}^* s'$, we prove that it is true for π_K . In particular, there exists a reduction $J(\sigma_K) : s_t \rightarrow_{\text{HAM}_J}^* s'$ such that $|J(\sigma_K)| = |\sigma_K| + |J(\sigma_K)|_{\uparrow}$ and $|J(\sigma_K)|_{\text{var}_J} = |\sigma_K|_{\text{hop/var}_K}$. We proceed considering all the possible $\rightarrow_{\text{HAM}_K}$ transitions from s' to s .

- Transitions $\rightarrow_{\bullet 1/\text{app}}$ and $\rightarrow_{\bullet 2/\text{abs}}$. These transitions belong also to $\rightarrow_{\text{HAM}_J}$, so the statement trivially holds. In particular, $|J(\pi_K)| = 1 + |J(\sigma_K)| =_{i.h.} 1 + |\sigma_K| + |J(\sigma_K)|_{\uparrow} = |\pi_K|$.
- Transition $\rightarrow_{\text{hop/var}_K}$. By Lemma 9.5, we have a run $\rho : s' \rightarrow_{\text{var}_J} s'' \rightarrow_{\uparrow}^* s$. Then we define $J(\pi_k)$ as the concatenation of $J(\sigma_k)$ and ρ , for which

$$|J(\pi_k)| = |J(\sigma_k)| + 1 + |\rho| =_{i.h.} |\sigma_K| + |J(\sigma_k)|_{\uparrow} + 1 + |\rho_J| = |\sigma_K| + 1 + |J(\pi_k)|_{\uparrow} = |\pi_K| + |J(\pi_k)|_{\uparrow}$$

$$\text{and } |J(\pi_k)|_{\text{var}_J} = 1 + |J(\sigma_k)|_{\text{var}_J} =_{i.h.} 1 + |\sigma_K|_{\text{hop/var}_K} = |\pi_K|_{\text{hop/var}_K}.$$

- (2) By induction on the length of π_J . If $|\pi_J| = 0$ there is nothing to prove. Now, let us consider $\pi_J : s_t \rightarrow_{\text{HAM}_J}^* s_1 \rightarrow_{\text{HAM}_J} s$. Considering the property true for the reduction $\pi'_J : s_t \rightarrow_{\text{HAM}_J}^* s_1$, we prove that it is true for π_J . By *i.h.*, there are runs $\pi'_K : s_t \rightarrow_{\text{HAM}_K}^* s_2$ and $\sigma'_J : s_1 \rightarrow_{\uparrow}^* s_2$ such that $\pi'_J \sigma'_J = J(\pi'_K)$. If σ'_J is non-empty then by determinism of the λ JAM we are done, because σ'_J has to pass through s and the suffix σ_J of σ'_J starting on s proves the statement. If σ'_J is empty then $s_1 = s_2$ and in particular s_2 has direction \downarrow , because it is reached by $\rightarrow_{\text{HAM}_K}$. Then consider the cases of transition $s_1 \rightarrow_{\lambda\text{JAM}} s$:

- Transitions $\rightarrow_{\bullet 1/\text{app}}$ and $\rightarrow_{\bullet 2/\text{abs}}$. These transitions belong also to $\rightarrow_{\text{HAM}_K}$, so $\rightarrow_{\text{HAM}_K}$ can do the same step and close the diagram.

- Transition $\rightarrow_{\text{var}j}$. Then we can close the diagram via the reasoning used at the previous point of the theorem, based on Lemma 9.5.
- (3) Two directions:
- $\rightarrow_{\text{HAM}_K}$ termination implies $\rightarrow_{\text{HAM}_J}$ termination: an omitted standard invariant ensures that if terms are closed then, whenever the code is a variable x , the environment is defined on x . This fact forbids $\rightarrow_{\text{HAM}_K}$ to get stuck on $\rightarrow_{\text{hop/var}K}$ transitions. So $\rightarrow_{\text{HAM}_K}$ final states have the shape $(\underline{\lambda x.t}, C, L, E, \epsilon)$, which are also $\rightarrow_{\text{HAM}_J}$ final states. Then if $\rightarrow_{\text{HAM}_K}$ terminates $\rightarrow_{\text{HAM}_J}$ terminates.
 - $\rightarrow_{\text{HAM}_J}$ termination implies $\rightarrow_{\text{HAM}_K}$ termination: we prove the contrapositive statement. Suppose that $\rightarrow_{\text{HAM}_K}$ diverges starting from s_t . Note that it has to make an infinity of $\rightarrow_{\text{hop/var}K}$ transitions, because without them—that is considering only $\rightarrow_{\bullet 1/\text{app}}$ and $\rightarrow_{\bullet 2/\text{abs}}$ —the size of the code strictly decreases. By the first point of the theorem, projecting the diverging $\rightarrow_{\text{HAM}_K}$ run we obtain a diverging $\rightarrow_{\text{HAM}_J}$ run, because the projection maps the infinity of $\rightarrow_{\text{hop/var}K}$ transitions to an infinity of $\rightarrow_{\text{var}j}$ transitions. □

Theorem B.6 immediately implies the following more concise statement given in the body of the paper (as Theorem 9.6).

COROLLARY B.7 (λ JAM AND KAM RELATIONSHIP). *There is a complete λ JAM run π_J from t if and only if there is a complete KAM run π_K from t . Moreover, $|\pi_J| = |\pi_K| + |\pi_J|_{\uparrow}$ and $|\pi_J|_{\text{var}} = |\pi_K|_{\text{var}}$.*

PROOF. It follows immediately from the previous theorem by the two obvious (and omitted) strong bisimulations between the KAM and the transition subrelation $\rightarrow_{\text{HAM}_K}$ of the HAM, and between the λ JAM and the transition subrelation $\rightarrow_{\text{HAM}_J}$ of the HAM. □

C PROOFS OF SECT. 10 (THE λ -JAM IS SLOWLY REASONABLE)

PROPOSITION C.1 (DEPTH INVARIANT). *Let $\pi : s_t \rightarrow_{\lambda\text{JAM}}^* s$ be an initial run of the λ JAM. Then $\text{depth}(s) = |\pi|_{\text{var}}$. Moreover $\text{depth}(s) \geq \text{depth}(l)$ for every logged position l in s .*

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

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

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

Then $\text{depth}(s) = \text{depth}(L_n \cdot L) + 1 = \text{depth}(s') + 1 =_{i.h.} |\sigma|_{\text{var}} + 1 = |\pi|_{\text{var}}$. For the *moreover* part, let $l := (x, C\langle \lambda x.D_n \rangle, L_n \cdot L)$ and consider a logged position $l' \neq l$ in s . By *i.h.* $\text{depth}(l') \leq \text{depth}(s') < \text{depth}(s)$. For l , instead, by definition of $\text{depth}(\cdot)$ we have $\text{depth}(l) = \text{depth}(s)$.

- Transitions $\rightarrow_{\bullet 3}$, $\rightarrow_{\bullet 4}$, and \rightarrow_{jmp} : the result holds by *i.h.*, since the polarity has not changed and neither has the depth of the tape. For the *moreover* part, the every logged position of s is in s' , and so it follows by the *i.h.*
- Transition \rightarrow_{arg} : the result follows by *i.h.*, since the depth of the tape of s is the same of the depth of the log of s' .

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

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

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

LEMMA C.2 (BOUND ON \uparrow PHASES).

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

PROOF.

- (1) We can split π in many subruns $\pi_1 \dots \pi_n$ consisting only of $\rightarrow_{\bullet,3,\bullet,4}$ sequences and interleaved by \rightarrow_{jmp} transitions, i.e. $\pi = \pi_1 \cdot \rightarrow_{\text{jmp}} \pi_2 \cdot \rightarrow_{\text{jmp}} \dots \pi_n$. By Lemma 10.1, each π_i is such that $|\pi_i| \leq |C(t)|$. Moreover, note that the log is untouched by π_i and that the number of \rightarrow_{jmp} transitions is bound by the depth of the first logged position in L , itself bound by $\text{depth}(s)$ by Lemma 10.2. Then $|\pi| \leq \text{depth}(s) \cdot |C(t)|$.
- (2) The run π has shape $\pi_1 \downarrow \pi_1 \uparrow \pi_2 \downarrow \pi_2 \uparrow \dots \pi_n \downarrow \pi_n \uparrow$ where $\pi_i \downarrow$ is made out of \rightarrow_{\downarrow} transitions and $\pi_i \uparrow$ is made out of \rightarrow_{\uparrow} transitions. By the previous point, we have $|\pi_i \uparrow| \leq \text{depth}(s_i \uparrow) \cdot |t|$ where $s_i \uparrow$ is the source state of $\pi_i \uparrow$. By Lemma 10.2, $\text{depth}(s_i \uparrow) = \sum_{j=1}^i |\pi_j \downarrow|_{\text{var}}$. Now, $|\pi|_{\uparrow} = \sum_{i=1}^n |\pi_i \uparrow| \leq \sum_{i=1}^n \text{depth}(s_i \uparrow) \cdot |t| = |t| \cdot \sum_{i=1}^n \sum_{j=1}^i |\pi_j \downarrow|_{\text{var}} \leq |t| \cdot \sum_{i=1}^n |\pi|_{\text{var}} \leq |t| \cdot |\pi|_{\text{var}}^2$.

□

THEOREM C.3 (λJAM COMPLEXITY). *Let t be a closed term such that $t \rightarrow_{\text{wh}}^n u$, u be \rightarrow_{wh} normal, and π_J and π_K be the complete λJAM and KAM runs from t . Then:*

- (1) The λJAM is quadratically slower than the KAM : $|\pi_K| \leq |\pi_J| = O(|\pi_K|^2 \cdot |t|)$.
- (2) The λJAM is (slowly) reasonable: $|\pi_J| = O(n^4 \cdot |t|)$, and the cost of implementing π_J on a RAM is also $O(n^4 \cdot |t|)$.

PROOF. (1) By Theorem B.6.1, $|\pi_J| = |\pi_K| + |\pi_J|_{\uparrow}$ and $|\pi_K|_{\text{var}} = |\pi_J|_{\text{var}}$. By Lemma 10.3.2, $|\pi_J|_{\uparrow} = |\pi_J|_{\text{var}}^2 \cdot |t| = |\pi_K|_{\text{var}}^2 \cdot |t| \leq |\pi_K|^2 \cdot |t|$, from which the statement follows.

- (2) The previous point gives $|\pi_J| = O(|\pi_K|^2 \cdot |t|)$ where π_K is the corresponding run on the KAM . As recalled in Sect. 10, $|\pi_K| = O(n^2)$, from which we obtain $|\pi_J| = O(n^4 \cdot |t|)$.

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

□

D PROOFS OF SECT. 11 (THE POINTER ABSTRACT MACHINE)

LEMMA D.1 (λPAM INVARIANTS). *Let $s = (t, C, H, i, T, d)$ be a reachable λPAM state. Then:*

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

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

LEMMA D.2 (LOGS AND HISTORIES). *Let $L \simeq_{\text{LH}} (H, i)$.*

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

PROOF.

- (1) By induction on n :
- *Base case*: if $n = 0$, then it is trivially satisfied since $L_n = \epsilon$ and $\phi_H^n(i) = i$, so that $L' = L_n \cdot L' \simeq_{LH} (H, i) = (H, \phi_H^n(i))$, as required.
 - *Inductive case*: if $n > 0$ first of all note that $\phi_H^m(i)$ is defined for all $m \leq n$ by the depth invariant (Lemma 11.1). Then, $L = L_{n-1} \cdot e \cdot L'$ and by *i.h.* $e \cdot L' \simeq_{LH} (H, \phi_H^{n-1}(i))$. By definition of \simeq_{LH} this is possible only if $L' \simeq_{LH} (H, \phi_H(\phi_H^{n-1}(i)))$, *i.e.* $L' \simeq_{LH} (H, \phi_H^n(i))$.
- (2) By induction on L . Cases:
- *Empty*, *i.e.* $L = \epsilon$. We have that the hypothesis is $\epsilon \simeq_{LH} (H, 0)$, because it is the only derivable relation for empty logs. Then $\epsilon \simeq_{LH} ((p, j) \cdot H, 0)$.
 - *Non-empty*, *i.e.* $L = (x, C, L'') \cdot L'$. By hypothesis, $l \cdot L' \simeq_{LH} (H, i)$, which implies that
 - (a) $(x, C) = (x_i^H, D_i^H)$,
 - (b) $L' \simeq_{LH} (H, \phi_H(i))$, and
 - (c) $L'' \simeq_{LH} (H, i - 1)$.
 By *i.h.*, we obtain $L' \simeq_{LH} ((p, j) \cdot H, \phi_H(i))$ and $L'' \simeq_{LH} ((p, j) \cdot H, i - 1)$, and clearly $(x, C) = (x_i^{(p,j) \cdot H}, D_i^{(p,j) \cdot H})$. Then $L = (x, C, L'') \cdot L' \simeq_{LH} ((p, j) \cdot H, i)$.

□

THEOREM D.3 (\simeq IS A STRONG BISIMULATION).

- (1) for every run $\pi_J : s_t^{\lambda JAM} \xrightarrow{*}_{\lambda JAM} s_J$ there exists a run $\pi_P : s_t^{\lambda PAM} \xrightarrow{*}_{\lambda PAM} s_P$ such that $s_J \simeq s_P$ and $|\pi_J| = |\pi_P|$ and performing exactly the same transitions;
- (2) for every run $\pi_P : s_t^{\lambda PAM} \xrightarrow{*}_{\lambda PAM} s_P$ there exists a run $\pi_J : s_t^{\lambda JAM} \xrightarrow{*}_{\lambda JAM} s_J$ such that $s_J \simeq s_P$ and $|\pi_J| = |\pi_P|$ and performing exactly the same transitions.

Moreover, if $s_J = (t, \underline{C}, L, T_J, \uparrow) \simeq (t, \underline{C}, H, i, T_P, \uparrow) = s_P$ and (x, D, L') is the unique logged position in T_J then $L' \simeq_{LH} (H, |H|)$.

PROOF. We prove the first point, the second point is symmetrical (simply replacing the use of Lemma 11.1—in the case of transition \rightarrow_{var} below—with Lemma 4.1). By induction on $|\pi_J|$. If π_J is empty then simply take π_P as the empty run. Otherwise, by *i.h.* there exists a λPAM run $\pi_P : s_t^{\lambda PAM} \xrightarrow{*}_{\lambda PAM} s_P$ such that $s_J \simeq s_P$ and $|\pi_J| = |\pi_P|$. Note that $s_J \simeq s_P$ implies $s_J = (t, E, L, T_J, d)$ and $s_P = (t, E, T_P, H, i, d)$ with $T_J \simeq_T T_P$ and $L \simeq_{LH} (H, i)$.

Let's consider the possible extensions of π_J with a further transition. Cases:

- Transitions $\rightarrow_{\bullet 1}, \rightarrow_{\bullet 2}, \rightarrow_{\bullet 3}, \rightarrow_{\bullet 4}$: we show one such case, the other are analogous.

$$\begin{array}{ccc}
 (\underline{ur}, E, L, T_J) & \rightarrow_{\bullet 1} & (\underline{u}, E \langle \langle \cdot \rangle r \rangle, L, \bullet \cdot T_J) \\
 \simeq & & \simeq \\
 (\underline{ur}, E, H, i, T_P) & \rightarrow_{\bullet 1} & (\underline{u}, E \langle \langle \cdot \rangle r \rangle, H, i, \bullet \cdot T_P)
 \end{array}$$

For $\rightarrow_{\bullet 3}$ and $\rightarrow_{\bullet 4}$ the *moreover* part follows from the *i.h.*

- Transition \rightarrow_{var} . We are in the following situation:

$$s_J = (\underline{x}, C \langle \lambda x.D_n \rangle, L_n \cdot L', T_J) \simeq (\underline{x}, C \langle \lambda x.D_n \rangle, H, i, T_P) = s_P$$

with $T_J \simeq_T T_P$ and $L = L_n \cdot L' \simeq_{LH} (H, i)$. The λPAM can do a \rightarrow_{var} transition (guaranteed by the depth invariant of Lemma 11.1), but we have to verify that the two target states are still \simeq -related. By Lemma D.2.1, we have $L' \simeq_{LH} (H, \phi_H^n(i))$. Then:

$$\begin{array}{ccc}
 (\underline{x}, C \langle \lambda x.D_n \rangle, L_n \cdot L', T_J) & \rightarrow_{\text{var}} & (\lambda x.D_n \langle x \rangle, \underline{C}, L', (x, C \langle \lambda x.D_n \rangle, L_n \cdot L') \cdot T_J) \\
 \simeq & & \simeq \\
 (\underline{x}, C \langle \lambda x.D_n \rangle, H, i, T_P) & \rightarrow_{\text{var}} & (\lambda x.D_n \langle x \rangle, \underline{C}, H, \phi_H^n(i), (x, C \langle \lambda x.D_n \rangle) \cdot T_P)
 \end{array}$$

Now, the *moreover* part. We have to prove that $L_n \cdot L' \simeq_{LH} (H, |H|)$. By hypothesis $L_n \cdot L' \simeq_{LH} (H, i)$ and by Lemma 11.1, we have $i = |H|$.

- Transition \rightarrow_{arg} . We are in the following situation:

$$s_J = (t, \underline{C\langle\langle \cdot \rangle u\rangle}, L, l \cdot T'_J) \simeq (t, \underline{C\langle\langle \cdot \rangle u\rangle}, H, i, p \cdot T'_P) = s_P$$

with $l \cdot T'_J \simeq_T p \cdot T'_P$ and $L \simeq_{LH} (H, i)$. The λ PAM can do a \rightarrow_{arg} transition, but we have to verify that the two target states are still \simeq -related. Namely, we have to show that $l \cdot L \simeq_{LH} ((p, i) \cdot H, |H| + 1)$. Let us set $H' := (p, i) \cdot H$ and $j := |H'| = |H| + 1$. We check that all three hypothesis of the rule defining \simeq_{LH} hold:

- (1) Since $L \simeq_{LH} (H, i)$, Lemma D.2.2 gives $L \simeq_{LH} ((p, i) \cdot H, i)$. Note that $\phi_{H'}(j) = i$, that is, $L \simeq_{LH} (H', \phi_{H'}(j))$.
- (2) Since $l \cdot T'_J \simeq_T p \cdot T'_P$, if $l = (x, D, L')$, then $p = (x, D)$ and thus $(x, D) = (x_j^H, D_j^H) = (x, D)$.
- (3) By *i.h.*, the logged position $l = (x, D, L')$ on the λ JAM tape verifies $L' \simeq_{LH} (H, |H|)$. By Lemma D.2.2, $L' \simeq_{LH} ((p, i) \cdot H, |H|)$, that is, $L' \simeq_{LH} (H', j - 1)$.

Then the two target states match:

$$\begin{array}{ccc} (t, \underline{C\langle\langle \cdot \rangle u\rangle}, L, l \cdot T'_J) & \rightarrow_{\text{arg}} & (\underline{u}, C\langle t \langle \cdot \rangle \rangle, l \cdot L, T'_J) \\ \simeq & & \simeq \\ (t, \underline{C\langle\langle \cdot \rangle u\rangle}, H, i, p \cdot T'_P) & \rightarrow_{\text{arg}} & (\underline{u}, C\langle t \langle \cdot \rangle \rangle, H', j, T'_P) \end{array}$$

- Transition \rightarrow_{jmp} . We are in the following situation:

$$s_J = (t, \underline{C\langle u \langle \cdot \rangle \rangle}, (x, D, L') \cdot L', T_J) \simeq (t, \underline{C\langle u \langle \cdot \rangle \rangle}, H, i, T_P) = s_P$$

with $T_J \simeq_T T_P$ and $(x, D, L') \cdot L' \simeq_{LH} (H, i)$. The λ PAM can do a \rightarrow_{jmp} transition, but we have to verify that the two target states are still \simeq -related. Note that, since $(x, D, L') \cdot L' \simeq_{LH} (H, i)$, we have $(x, D) = (x_i^H, D_i^H)$ and $L' \simeq_{LH} (H, i - 1)$. Therefore:

$$\begin{array}{ccc} (t, \underline{C\langle u \langle \cdot \rangle \rangle}, (x, D, L') \cdot L', T_J) & \rightarrow_{\text{jmp}} & (x, \underline{D}, L', T_J) \\ \simeq & & \simeq \\ (t, \underline{C\langle u \langle \cdot \rangle \rangle}, H, i, T_P) & \rightarrow_{\text{jmp}} & (x_i^H, \underline{D}_i^H, H, i - 1, T_P) \end{array}$$

The *moreover* part follows from the *i.h.*

□

E PROOFS OF SECT. 14 (TYPED INTERACTIONS ARE EXHAUSTING)

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

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

E.1 S-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 3. 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.

$$\begin{array}{c}
\frac{x : [[\star] \rightarrow \star] \vdash x : [\star\downarrow_{16}] \rightarrow \star\uparrow_6 \quad y : [\star] \vdash y : \star\uparrow_{17}}{y : [\star], x : [[\star] \rightarrow \star] \vdash xy : \star\uparrow_5} \\
\frac{y : [\star] \vdash \lambda x. xy : [[\star\uparrow_{15}] \rightarrow \star\downarrow_7] \rightarrow \star\uparrow_4}{\vdash \lambda y. \lambda x. xy : [\star\downarrow_{18}] \rightarrow [[\star\uparrow_{14}] \rightarrow \star\downarrow_8] \rightarrow \star\uparrow_3} \quad \frac{}{\vdash I : \star\uparrow_{19}} \quad \frac{z : [\star] \vdash z : \star\uparrow_{11}}{\vdash \lambda z. z : [\star\downarrow_{12}] \rightarrow \star\uparrow_{10}} \\
\hline
\vdash (\lambda y. \lambda x. xy)I : [[\star\uparrow_{13}] \rightarrow \star\downarrow_9] \rightarrow \star\uparrow_2 \quad \frac{}{\vdash (\lambda y. \lambda x. xy)I(\lambda z. z) : \star\uparrow_1}
\end{array}$$

We start by defining the notions of typed tests used to define S-exhaustible states. Somewhat surprising, while in the λ IAM tape tests are easy to define and log tests require some syntactical gymnastics, here it is the other way around.

Type Positions and Generalized States. To define tests, we have to consider a slightly more general notion of SIAM state. In Sect. 13, a state is a quadruple (π, J, \mathbb{B}, d) where J is an occurrence of a judgement $\Gamma \vdash u : A$ in π , d is a direction, and \mathbb{B} is a type context isolating an occurrence of \star in A . The generalization simply is to consider type contexts \mathbb{B} such that $\mathbb{B}\langle A' \rangle = A$ for some A' , that is, not necessarily isolating \star . A pair (A', \mathbb{B}) such that $\mathbb{B}\langle A' \rangle = A$ is called a position in A .

Note that the SIAM can be naturally adapted to this more general notion of state, that follows an arbitrary formula A' , not necessarily \star and it amounts to simply replace \star with A' .

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

SIAM Tests. Given a SIAM state $s = (\pi, J, (A, \mathbb{B}), d)$, the underlying idea is that the judgement occurrence J encodes the log of the λ IAM, while the type context \mathbb{B} 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) an element A' of a sequence S related to s , and that these sequence elements play the role of logged positions in the λ IAM. These sequence elements are of two kinds:

- (1) *Elements containing J :* those in which the focused judgment J itself is contained, corresponding to the logged positions in the log of the λ IAM. Note that the positions on the log are those for which the λ IAM has previously found the corresponding arguments. In the SIAM these arguments are exactly those in which the focused judgment is contained.
- (2) *Elements appearing in \mathbb{B} :* 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 π' and a position (A', \mathbb{B}') in the right-hand type of π' .

Definition E.1 (Focus). A focus f in a derivation π is a pair $f = (J, (A, \mathbb{B}))$ of a judgement occurrence J and of a type position (A, \mathbb{B}) in the right-hand type $\mathbb{B}\langle A \rangle$ of J .

The intuition is that exhausting a test $s_{J, (A, \mathbb{B})}$ 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 E.2 (Judgement tests). Let $s = (\pi, J, (A, \mathbb{B}), d)$ be a SIAM state. Let r_i be i -th T-@ rule found traversing π by descending from the focused judgement J towards the final judgement of π .

Let J_i be the judgement of the sequence S_i in the right premise of r_i traversed in such a descent (careful: J_i is the j -th judgement of S_i for some j , that is, the index i denotes the connection with rule r_i , not the position in S_i). Let J_i be $\Gamma \vdash t : A'$. 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 SIAM and the λ IAM, having type context $\langle \cdot \rangle$ corresponds to the fact that the log tests of the λ IAM always have an empty tape.

Example E.3 (Judgement test). Let us give an example of judgement test in the context of the given example of SIAM run. If we consider the state $\uparrow 11$, we find its log tests going down in the type derivation for each T-@ rule traversed from the right hand side. In this case we immediately find the judgment $\vdash \lambda z.z : [\star] \rightarrow \star$. Then, $\vdash \lambda z.z : \langle [\star] \rightarrow \star \rangle \downarrow$ is a log test for $\uparrow 11$. Since between $\vdash \lambda z.z : [\star] \rightarrow \star$ and the root of the derivation we do not cross any other suitable T-@ rule, there are no other log tests for $\uparrow 11$.

Type (Context) Tests. While judgement tests depend only on the judgement occurrence J of a state $s = (\pi, J, (A, \mathbb{B}), d)$, type context tests—dually—fix J and depend only on the type context \mathbb{B} of s , that is, they all focus on sequence elements of the form $(J, (A', \mathbb{B}'))$ where $\mathbb{B}' \langle A' \rangle = \mathbb{B} \langle A \rangle$ and $\mathbb{B} = \mathbb{B}' \langle \mathbb{B}'' \rangle$ for some type context \mathbb{B}'' . Namely, there is one type context test (shortened to *type test*) for every sequence in which the hole of \mathbb{B} 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{B}_n of level $n \in \mathbb{N}$ as follows:

$$\begin{aligned} \mathbb{B}_0 &:= \langle \cdot \rangle \mid S \rightarrow \mathbb{B}_0 \\ \mathbb{B}_{n+1} &:= [\dots \mathbb{B}_n \dots] \rightarrow A \mid S \rightarrow \mathbb{B}_{n+1} \end{aligned}$$

Clearly, every type context \mathbb{B} can be seen as a type context \mathbb{B}_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{B} is a context \mathbb{B}' such that $\mathbb{B}' \langle \mathbb{B}'' \rangle = \mathbb{B}$ for some \mathbb{B}'' . Given \mathbb{B} of level $n > 0$, there is a smallest prefix context $\mathbb{B}|_i$ of level $0 < i \leq n$, and it has the form $\mathbb{B}' \langle [\dots \langle \cdot \rangle \dots] \rightarrow A \rangle$ for a type context \mathbb{B}' of level $i - 1$.

Definition E.4 (Type tests). Let $s = (\pi, J, (A, \mathbb{B}), d)$ be a SIAM state and n be the level of \mathbb{B} . The sequence of directed prefixes $\text{DiPref}(\mathbb{B})$ of \mathbb{B} is the sequence of pairs (\mathbb{B}', d') , where \mathbb{B}' is a prefix of \mathbb{B} , defined as follows:

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

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

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

Example E.5 (Type test). Let us now give examples of type tests in the example of SIAM run that we provided. We compute the tape tests of $\uparrow 13$. Its type is

$$[[\langle \star \rangle] \rightarrow \star] \rightarrow \star$$

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

Definition E.6 (State respecting a focus). Let $f = (J, (A, \mathbb{B}))$ be a focus. A SIAM state s respects f if it is an axiom $\vdash x : \langle A \rangle_{\downarrow}$ for some variable x (the typing context of s , which is omitted by convention, is $x : A$).

Definition E.7 (S-Exhaustible states). The set \mathcal{E}_S of S-exhaustible states is the smallest set such that if $s \in \mathcal{E}_S$, then for each type or judgement test of s_f of focus f there exists a run $\pi : s_f \xrightarrow{*}_{\text{SIAM}} \rightarrow_{\text{bt2}} s'$ where s' respects f and for the shortest such run $s' \in \mathcal{E}_S$.

LEMMA E.8 (S-EXHAUSTIBLE INVARIANT). *Let t be a closed term, $\pi \triangleright \Gamma \vdash t : A$ a sequence type derivation for it, and $\pi : \vdash t : \langle A \rangle_{\uparrow} \xrightarrow{k}_{\text{SIAM}} s$ an initial SIAM run. Then s is S-exhaustible.*

PROOF. By induction on k . For $k = 0$ there is nothing to prove because the initial state $s_0 = \vdash t : \langle A \rangle_{\uparrow}$ has no judgement nor type tests. Then suppose $\pi' : s_0 \xrightarrow{k-1}_{\text{SIAM}} s'$ and that the run continues with $s' \rightarrow_{\text{SIAM}} s$. By *i.h.*, s' is S-exhaustible.

Terminology: when a test state satisfies the clause in the definition of S-exhaustible states we say that it is *positive*.

Cases of $s' \rightarrow_{\text{SIAM}} s$:

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

$$s' = \vdash tu : \mathbb{B}\langle \star \uparrow \rangle (= A) \xrightarrow{\bullet 1} \frac{\vdash t : S \rightarrow A \quad [\vdash]}{\vdash tu : A} = s$$

– *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 $s'_f \rightarrow_{\text{SIAM}} s_f$. Since the machine is deterministic also s_f is positive. Let us now consider a type test s_f of direction \downarrow . We observe that there is a corresponding type test s'_f of s' , that it is positive by *i.h.*, and that $s_f \rightarrow s'_f$. Then s_f is positive.

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

$$s' = \frac{\frac{\vdash x : \mathbb{B}\langle \star \uparrow \rangle_i (= A_i) \quad i}{\vdots}}{\vdash \lambda x. C\langle x \rangle : [\dots A_i \dots] \rightarrow A'} \xrightarrow{\text{var}} \frac{\frac{\vdash x : A_i \quad i}{\vdots}}{\vdash \lambda x. C\langle x \rangle : [\dots \mathbb{B}\langle \star \downarrow \rangle_i \dots] \rightarrow A'} = s$$

– *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{B} . Let s^j be the type test of s associated to the j -th triple in $\text{DiPref}([\dots \mathbb{B} \dots] \rightarrow A')$. Three cases, depending on the index j of s^j :

- (1) $j = 1$: then s^1 is $\vdash \lambda x. C\langle x \rangle : \mathbb{B}\langle \star \uparrow \rangle, [\dots \langle \cdot \rangle \dots] \rightarrow A'$. Note that $s^1 \rightarrow_{\text{bt2}} \vdash x : \mathbb{B}\langle \star \downarrow \rangle, \langle \cdot \rangle$, which has no type tests and has the same judgement tests of s' , which by *i.h.* are positive. Hence, s^1 is S-exhaustible.

- (2) j is even: for s^j (of direction \Downarrow) there is a corresponding type test s'^{j-1} of odd index of s' , having direction \Uparrow and such that $s'^{j-1} \rightarrow_{\text{var}} s^j$. Thus one can conclude by *i.h.* and determinism of the SIAM.
- (3) $j \neq 1$ is odd: for s^j (of direction \Uparrow) there is a corresponding type test s'^{j-1} of even index of s' , having direction \Downarrow and such that $s^j \rightarrow_{\text{bt2}} s'^{j-1}$. Thus one can conclude by *i.h.*
- Case \rightarrow_{bt2} .

$$s' = \frac{\frac{\vdash x : A_i (= \mathbb{B}\langle \star \rangle_i)}{\vdots} \quad \frac{\vdash x : \mathbb{B}\langle \star \rangle_i}{\vdots}}{\vdash \lambda x. C(x) : [\dots \mathbb{B}\langle \star \rangle_i \dots] \rightarrow A'} \rightarrow_{\text{bt2}} \frac{\vdash \lambda x. C(x) : [\dots A_i \dots] \rightarrow A'}{\vdash \lambda x. C(x) : [\dots \mathbb{B}\langle \star \rangle_i \dots] \rightarrow A'} = s$$

- *Judgement tests.* The first type test of s' is $s'^1 := \vdash \lambda x. C(x) : \mathbb{B}\langle \star \rangle_i \Uparrow, [\dots \langle \cdot \rangle \dots] \rightarrow A'$. Note that $s'^1 \rightarrow_{\text{bt2}} \vdash x : \mathbb{B}\langle \star \rangle_i \Downarrow, \langle \cdot \rangle =: s''$ and that s'' exhausts s'^1 , and it is the first such state. Since s'^1 is positive, s'' is S-exhaustible. Note that s'' has the same judgment tests of s , which are then positive.
- *Type tests.* For each odd type test s^i of s (whose direction is \Uparrow), the corresponding even type test s'^{i+1} of s' has direction \Downarrow , is positive by *i.h.*, and such that $s^i \rightarrow_{\text{var}} s'^{i+1}$. 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 $s'^{i+1} \rightarrow_{\text{bt2}} s^i$. Then s^i is positive by determinism of the SIAM.

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

$$s' = \frac{\vdash t : [\dots \mathbb{B}\langle \star \rangle_i \dots] \rightarrow A \quad \vdash_i u : A'_i (= \mathbb{B}\langle \star \rangle_i \Downarrow)}{\vdash tu : A} \rightarrow_{\text{arg}} \frac{\vdash t : [\dots A'_i \dots] \rightarrow A \quad \vdash_i u : \mathbb{B}\langle \star \rangle_i \Uparrow}{\vdash tu : A} = s$$

- *Judgement tests.* Judgement tests of s are those of s' , which are positive by *i.h.*, plus $s^u := \vdash u : \mathbb{B}\langle \star \rangle_i \Downarrow, \langle \cdot \rangle$. Please note that $s^u \rightarrow_{\text{bt1}} \vdash t : \mathbb{B}\langle \star \rangle_i \Uparrow, [\dots \langle \cdot \rangle \dots] \rightarrow A =: s''$. Now, s'' 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 SIAM. For each even type test s^i of s' (whose direction is \Downarrow), the corresponding odd type test s'^{i+1} of s' has direction \Uparrow , is positive by *i.h.*, and such that $s^i \rightarrow_{\text{bt1}} s'^{i+1}$. Then s^i is positive.

- Case \rightarrow_{bt1} .

$$s' = \frac{\vdash t : [\dots A'_i \dots] \rightarrow A \quad \vdash_i u : \mathbb{B}\langle \star \rangle_i \Downarrow (= A'_i)}{\vdash tu : A} \rightarrow_{\text{bt1}} \frac{\vdash t : [\dots \mathbb{B}\langle \star \rangle_i \dots] \rightarrow A \quad \vdash_i u : A'_i}{\vdash tu : A} = s$$

- *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 $s^1 := \vdash t : \mathbb{B}\langle \star \rangle_i \Uparrow, [\dots \langle \cdot \rangle \dots] \rightarrow A$. Please note that $s'^u := \vdash u : \mathbb{B}\langle \star \rangle_i \Downarrow, \langle \cdot \rangle$ is a judgement test of s' such that $s'^u \rightarrow_{\text{bt1}} s^1$. By *i.h.*, s'^u is positive. By determinism of the SIAM, s^1 is positive. For each odd type test s^i of s (whose direction is \Uparrow), the corresponding even type test s'^{i-1} of s' has direction \Downarrow , is positive by *i.h.*, and such that $s'^{i-1} \rightarrow_{\text{bt1}} s^i$. Then s^i is positive by determinism of the SIAM. For each even type test s^i of s' (whose direction is \Downarrow), the corresponding odd type test s'^{i-1} of s' has direction \Uparrow , is positive by *i.h.*, and such that $s^i \rightarrow_{\text{arg}} s'^{i-1}$. Then s^i is positive.

□

E.2 Extracting λ IAM States from SIAM S-Exhaustible States, and the λ IAM/SIAM Strong Bisimulation

From S-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 S-exhaustible states. Indeed, the induction principle used to define S-exhaustibility allows recursive definition on S-exhaustible states to be well-behaved.

Definition E.9 (Extraction of logged positions). Let s be an S-exhaustible SIAM state in a derivation π , t be the final term in π , and s_f be a judgement or type test of s . Since s is S-exhaustible, there is an exhausting run $s_f \rightarrow_{\text{SIAM}}^+ s' \in \mathcal{E}_S$. 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 E.10 (Extraction of logs, tapes, and states). Let $s = (\pi, J, (A, \mathbb{B}), d)$ be an S-exhaustible SIAM state where t is the final term in π , and J is $\Gamma \vdash u : \mathbb{B}\langle A \rangle$. The λ IAM state extracted from s is $s_{\text{ext}}(s) := (u, C_s, L_{\text{ext}}(s), T_{\text{ext}}(s), d)$ where

- *Context:* C_s is the only term context such that $t = C_s\langle u \rangle$;
- *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:* $T_{\text{ext}}(s) = T_{\text{ext}}^s(\mathbb{B}, 0)$ where $T_{\text{ext}}^s(\mathbb{B}, i)$ is the auxiliary function defined by induction on \mathbb{B} as follows.

$$\begin{aligned} T_{\text{ext}}^s(\langle \cdot \rangle, i) &:= \epsilon \\ T_{\text{ext}}^s(S \rightarrow \mathbb{B}, i) &:= \bullet \cdot T_{\text{ext}}^s(\mathbb{B}, i) \\ T_{\text{ext}}^s([\dots \mathbb{B} \dots] \rightarrow A', i) &:= l_{\text{ext}}(s_f^i) \cdot T_{\text{ext}}^s(\mathbb{B}, i + 1) \end{aligned}$$

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

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

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

LEMMA E.11. *Let s be an S-exhaustible SIAM 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-@ 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 E.12 (SIAM- 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 S-exhaustible SIAM states on π and λ IAM states on t . Moreover, if $s_\pi \simeq_{\text{ext}} s_\lambda$ then s_π is SIAM 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 SIAM transitions are simulated by the λ IAM, the other half is essentially identical.

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

$$\begin{array}{c}
\frac{\vdash t : S \rightarrow A \quad [\vdash]}{s' = \vdash tu : \mathbb{B}\langle \star \uparrow \rangle (= A)} \quad \rightarrow_{\bullet 1} \quad \frac{\vdash t : S \rightarrow \mathbb{B}\langle \star \uparrow \rangle \quad [\vdash]}{\vdash tu : A} = s \\
\approx_{\text{ext}} \\
s_{\text{ext}}(s) = (\underline{tu}, C_{s'}, L_{\text{ext}}(s'), T_{\text{ext}}(s')) \quad \rightarrow_{\bullet 1} \quad (\underline{t}, C\langle \langle \cdot \rangle r \rangle, L_{\text{ext}}(s'), \bullet \cdot T_{\text{ext}}(s')) = s_{\lambda}
\end{array}$$

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, $s \approx_{\text{ext}} s_{\lambda}$.

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

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

First of all, C_s has shape $C\langle \lambda x. D_n \rangle$ for some n has the descending path from the focused judgement to the final judgement passes through the showed T- λ rule. Then $C_{s'} = C$.

About the log, by Lemma E.11 there is a correspondance between the level of term contexts and the length of the extracted log, so that $L_{\text{ext}}(s)$ has at least length n , that is, $L_{\text{ext}}(s') = L_n \cdot L$, and $L_{\text{ext}}(s) = L$.

About the tape, note that $T_{\text{ext}}(s) = l_{\text{ext}}(s_f^1) \cdot T_{\text{ext}}^s(\mathbb{B}, 1)$ 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 $\vdash \lambda x. C \langle x \rangle : \mathbb{B}\langle \star \rangle_{i \uparrow}, [\dots \langle \cdot \rangle \dots] \rightarrow A'$. Note that $s_f^1 \rightarrow_{\text{bt}2} \vdash x : \mathbb{B}\langle \star \rangle_{i \uparrow}, \langle \cdot \rangle = s''$, 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, $l_{\text{ext}}(s_f^1) = (x, \lambda x. D_n, L_n)$.
- (2) $T_{\text{ext}}^s(\mathbb{B}, 1) = T_{\text{ext}}(s')$, that is, $T_{\text{ext}}^s(\mathbb{B}, 1) = T_{\text{ext}}^{s'}(\mathbb{B}, 0)$. Note that $T_{\text{ext}}^s(\mathbb{B}, 1)$ and $T_{\text{ext}}^{s'}(\mathbb{B}, 0)$ 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{B} . Here one has to repeat the reasoning done in the $\rightarrow_{\text{bt}2}$ case of the proof of the S-exhaustible invariant (Lemma E.8), that shows that the tape test of index $i > 1$ for s and the one of index $i - 1$ of s' exhaust on the same state, and thus induce the same logged position. Then $T_{\text{ext}}^s(\mathbb{B}, 1) = T_{\text{ext}}(s')$.

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, $s \approx_{\text{ext}} s_{\lambda}$.

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

$$s' = \frac{\frac{\vdash x : A_i (= \mathbb{B}\langle \star \rangle_i)}{\vdots} \quad i}{\vdash \lambda x. C(x) : [\dots \mathbb{B}\langle \star \rangle_i \dots]} \rightarrow A' \quad \rightarrow_{\text{bt}2} \quad \frac{\frac{\vdash x : \mathbb{B}\langle \star \rangle_i}{\vdots} \quad i}{\vdash \lambda x. C(x) : [\dots A_i \dots]} = s$$

$$\simeq_{\text{ext}}$$

$$s_{\text{ext}}(s') = (\lambda x. D_n(x), C_{s'}, L_{\text{ext}}(s'), \underbrace{(x, \lambda x. D_n, L_n) \cdot T_{\text{ext}}^{s'}(\mathbb{B}, 1)}_{=T_{\text{ext}}(s')}) \rightarrow_{\text{bt}2} (x, C_{s'}(\lambda x. D_n), L_n \cdot L_{\text{ext}}(s'), T_{\text{ext}}^{s'}(\mathbb{B}, 1)) = s' \lambda$$

About the tape of $s_{\text{ext}}(s')$, note that $T_{\text{ext}}(s') = l_{\text{ext}}(s_f^1) \cdot T_{\text{ext}}^{s'}(\mathbb{B}, 1)$ 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 $s_f^1 \vdash \lambda x. C(x) : \mathbb{B}\langle \star \rangle_{i\uparrow}, [\dots \langle \cdot \rangle \dots] \rightarrow A'$. Note that $s_f^1 \rightarrow_{\text{bt}2} \vdash x : \mathbb{B}\langle \star \rangle_{i\downarrow}, \langle \cdot \rangle = s''$, 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 $C_s = C_{s'}(\lambda x. D_n)$ and $L_{\text{ext}}(s) = L_n \cdot L_{\text{ext}}(s')$.

About the tape, for s we have to prove that $T_{\text{ext}}^{s'}(\mathbb{B}, 1) = T_{\text{ext}}(s) = T_{\text{ext}}^s(\mathbb{B}, 0)$. This is done as for \rightarrow_{var} , mimicking the reasoning in the proof of the S-exhaustible invariant (Lemma E.8).

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} .

$$s' = \frac{\frac{\vdash t : [\dots \mathbb{B}\langle \star \rangle_i \dots] \rightarrow A \quad \vdash_i u : A'_i (= \mathbb{B}\langle \star \rangle_{i\downarrow})}{\vdash tu : A}}{\simeq_{\text{ext}}} \rightarrow_{\text{arg}} \frac{\frac{\vdash t : [\dots A'_i \dots] \rightarrow A \quad \vdash_i u : \mathbb{B}\langle \star \rangle_{i\uparrow}}{\vdash tu : A}}{= s}$$

$$s_{\text{ext}}(s') = (t, \underbrace{D\langle \langle \cdot \rangle \rangle u}_{=C_{s'}}, L_{\text{ext}}(s'), \underbrace{l_{\text{ext}}(s_f^1) \cdot T_{\text{ext}}^{s'}(\mathbb{B}, 1)}_{=T_{\text{ext}}(s')}) \rightarrow_{\text{arg}} (\underline{u}, D\langle \langle \cdot \rangle \rangle, l_{\text{ext}}(s_f^1) \cdot L_{\text{ext}}(s'), T_{\text{ext}}^{s'}(\mathbb{B}, 1)) = s' \lambda$$

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 S-exhaustible invariant. Similarly, one obtains that $T_{\text{ext}}^{s'}(\mathbb{B}, 1) = T_{\text{ext}}(s) = T_{\text{ext}}^s(\mathbb{B}, 0)$.

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

$$s' = \frac{\frac{\vdash t : [\dots A'_i \dots] \rightarrow A \quad \vdash_i u : \mathbb{B}\langle \star \rangle_{i\downarrow} (= A'_i)}{\vdash tu : A}}{\simeq_{\text{ext}}} \rightarrow_{\text{bt}1} \frac{\frac{\vdash t : [\dots \mathbb{B}\langle \star \rangle_i \dots] \rightarrow A \quad \vdash_i u : A'_i}{\vdash tu : A}}{= s}$$

$$s_{\text{ext}}(s') = (u, \underbrace{D\langle \langle \cdot \rangle \rangle}_{=C_{s'}} u, \underbrace{l_{\text{ext}}(s_f^1) \cdot L}_{=L_{\text{ext}}(s')}, T_{\text{ext}}(s')) \rightarrow_{\text{bt}1} (\underline{t}, D\langle \langle \cdot \rangle \rangle u, L, l_{\text{ext}}(s_f^1) \cdot T_{\text{ext}}(s')) = s' \lambda$$

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'^1_f of s' . Second, that $T_{\text{ext}}^s(\mathbb{B}, 1) = T_{\text{ext}}(s') = T_{\text{ext}}^{s'}(\mathbb{B}, 0)$. Both points follow exactly the reasoning done in the proof of the S-exhaustible invariant. \square

E.3 The SIAM is acyclic and thus weights measure its time

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 E.13. *Let T be an acyclic bi-deterministic transition system on a finite set of states \mathcal{S} and with only one initial state s_i . Then all states in \mathcal{S} are reachable from s_i , and reachable only once.*

PROOF. Let us consider a generic state $s \in \mathcal{S}$ and show that it is reachable from s_i . If $s = s_i$ we are done. Otherwise, since the system is bi-deterministic we can go deterministically backwards from s . Since states are finite and there are no cycles, then the reduction 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 SIAM is acyclic, we need to show that if $t \rightarrow_{wh} u$, then cycles are preserved between the sequence type derivation π for t and the sequence type derivation π' for u . One way to show this fact is building a bisimulation between states of π and states of π' , since bisimulations preserve (non)termination. This idea has been already exploited by [Accattoli et al. \[2020a\]](#), in order to prove the correctness of the λ IAM.

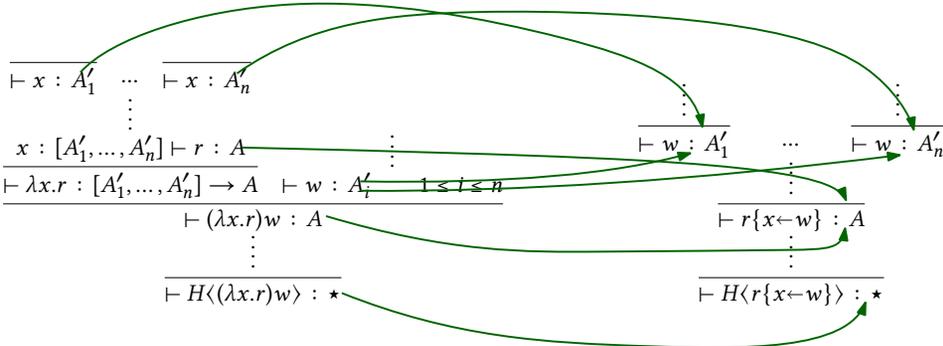
Weak Head Contexts. First of all, we need the notion of weak head context H defined as:

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

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

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

- (1) each axiom for x in π is \blacktriangleright -related with the 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' .



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

Definition E.14 (Bisimulation \blacktriangleright). The definition of \blacktriangleright for $\vdash (t, C) : \mathbb{B}\langle\star\rangle$ 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 :

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

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

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

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

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

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

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

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

PROPOSITION E.15. \blacktriangleright is a loop-preserving bisimulation between SIAM states.

PROOF. ¹⁵ We inspect the 4 cases of the definition of \blacktriangleright .

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

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

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

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

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

Cases for \downarrow (by cases of K):

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

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

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

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

¹⁵Also this proof requires colors.

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

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

- Rule body: $\vdash (t, H\langle(\lambda x.D)u\rangle) : \mathbb{B}(\star) \blacktriangleright_{\text{body}} \vdash (t\{x\leftarrow u\}, H\langle D\{x\leftarrow u\}\rangle) : \mathbb{B}(\star)$. Cases of \uparrow (by cases of t):
 - $t = rw$. Trivially closed by rule body.
 - $t = \lambda y.r$. If $t : \star$ both machines are stuck. If $t : S \rightarrow A$, the diagram is trivially closed by rule body.
 - $t = x$. Diagram closed by rule arg.

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

Cases of \downarrow (by cases of D):

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

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

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

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

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

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

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

□

COROLLARY E.16. *If $\pi \triangleright \vdash H\langle(\lambda x.r)w\rangle : \star$ contains a cycle, the also $\pi' \triangleright \vdash H\langle r\{x\leftarrow w\}\rangle : \star$ contains a cycle.*

PROOF. If the run of the SIAM on $\pi \triangleright \vdash H\langle(\lambda x.r)w\rangle : \star$ loops then there exists a state s_π such that a computation starting from s_π diverges. Every state but $\vdash (\lambda x.r, H\langle\langle\cdot\rangle\rangle w) : \mathbb{B}(\star)$, which however is not final, is related by \blacktriangleright to a state $s_{\pi'}$ of $T_{\pi'}$. By preservation of (non)termination (see [Accattoli et al. 2020a]), also $s_{\pi'}$ diverges. Since $s_{\pi'}$ has a finite number of states, there must be a cycle. □

COROLLARY E.17. *For each type derivation $\pi \triangleright \vdash t : \star$, T_π has no cycles.*

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

THEOREM E.18. *For every closed term t , the IAM takes n steps in t iff $\mathbf{W}_{\lambda\text{IAM}}(\pi) = n$ for every $\pi \triangleright \vdash t : \star$.*

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

F DUPLICATION EXAMPLE

We provide an example that illustrates how duplication is handled by the different abstract machines. We consider the λ -term $t := (\lambda x.xx)(\lambda y.y)$.

F.1 The λIAM

The first steps of the computation are needed to reach the head variable, namely x .

	Sub-term	Context	Log	Tape	Dir
	$(\lambda x.xx)(\lambda y.y)$	$\langle \cdot \rangle$	ϵ	ϵ	\downarrow
$\rightarrow_{\bullet 1}$	$\lambda x.xx$	$\langle \cdot \rangle(\lambda y.y)$	ϵ	\bullet	\downarrow
$\rightarrow_{\bullet 2}$	xx	$(\lambda x.\langle \cdot \rangle)(\lambda y.y)$	ϵ	ϵ	\downarrow
$\rightarrow_{\bullet 1}$	x	$(\lambda x.\langle \cdot \rangle x)(\lambda y.y)$	ϵ	\bullet	\downarrow

Once the head variable x has been found, the machine switches to upward mode \uparrow in order to find its argument $\lambda y.y$.

	Sub-term	Context	Log	Tape	Dir
	x	$(\lambda x.\langle \cdot \rangle x)(\lambda y.y)$	ϵ	\bullet	\downarrow
\rightarrow_{var}	$\lambda x.xx$	$\langle \cdot \rangle(\lambda y.y)$	ϵ	$(x, \lambda x.\langle \cdot \rangle x, \epsilon) \cdot \bullet$	\uparrow
\rightarrow_{arg}	$\lambda y.y$	$(\lambda x.xx)\langle \cdot \rangle$	$(x, \lambda x.\langle \cdot \rangle x, \epsilon)$	\bullet	\downarrow

Intuitively, the first occurrence of x has been substituted for $\lambda y.y$, thus forming a new virtual β -redex $(\lambda y.y)x$. Indeed, a \bullet is on top of the tape, thus allowing the λIAM to inspect $\lambda y.y$, reaching its head variable y .

	Sub-term	Context	Log	Tape	Dir
	$\lambda y.y$	$(\lambda x.xx)\langle \cdot \rangle$	$(x, \lambda x.\langle \cdot \rangle x, \epsilon)$	\bullet	\downarrow
$\rightarrow_{\bullet 2}$	y	$(\lambda x.xx)(\lambda y.\langle \cdot \rangle)$	$(x, \lambda x.\langle \cdot \rangle x, \epsilon)$	ϵ	\downarrow
\rightarrow_{var}	$\lambda y.y$	$(\lambda x.xx)\langle \cdot \rangle$	$(x, \lambda x.\langle \cdot \rangle x, \epsilon)$	$(y, \lambda y.\langle \cdot \rangle, \epsilon)$	\uparrow

Once the head variable y has been found, the machine, in upward mode \uparrow , starts looking for the argument of y from its binder $\lambda y.y$. However, $\lambda y.y$ was not the left side of an application forming a β -redex. Indeed, it was virtually substituted for the first occurrence of x , in the log, thus creating the virtual redex $(\lambda y.y)x$. Its argument is thus the second occurrence of x . The λIAM is able to retrieve it, walking again the path towards the variable $\lambda y.y$ has been virtually substituted for, namely the first occurrence of x , saved in the log. This is what we call backtracking.

	Sub-term	Context	Log	Tape	Dir
	$\lambda y.y$	$(\lambda x.xx)\langle \cdot \rangle$	$(x, \lambda x.\langle \cdot \rangle x, \epsilon)$	$(y, \lambda y.\langle \cdot \rangle, \epsilon)$	\uparrow
\rightarrow_{bt1}	$\lambda x.xx$	$\langle \cdot \rangle(\lambda y.y)$	ϵ	$(x, \lambda x.\langle \cdot \rangle x, \epsilon) \cdot (y, \lambda y.\langle \cdot \rangle, \epsilon)$	\downarrow
\rightarrow_{bt2}	x	$(\lambda x.\langle \cdot \rangle x)(\lambda y.y)$	ϵ	$(y, \lambda y.\langle \cdot \rangle, \epsilon)$	\uparrow
\rightarrow_{arg}	x	$(\lambda x.x\langle \cdot \rangle)(\lambda y.y)$	$(y, \lambda y.\langle \cdot \rangle, \epsilon)$	ϵ	\downarrow

Notice that we are able to backtrack because we saved the occurrence of the substituted variable in the token, otherwise the machine would not be able to know which occurrence of x is the right one. Of course, when the first occurrence of x is reached the IAM, now again in upward mode \uparrow , finds immediately its argument, that is the second occurrence of x . At this point the machine looks for the argument of this last occurrence of x , finding, of course, again $\lambda y.y$.

	Sub-term	Context	Log	Tape	Dir
	\underline{x}	$(\lambda x.x(\cdot))(\lambda y.y)$	$(y, \lambda y.(\cdot), \epsilon)$	ϵ	\downarrow
\rightarrow_{var}	$\lambda x.xx$	$(\cdot)(\lambda y.y)$	ϵ	$(x, \lambda x.x(\cdot), (y, \lambda y.(\cdot), \epsilon))$	\uparrow
\rightarrow_{arg}	$\underline{\lambda y.y}$	$(\lambda x.xx)(\cdot)$	$(x, \lambda x.x(\cdot), (y, \lambda y.(\cdot), \epsilon))$	ϵ	\downarrow

The computation then stops, signaling that t has weak head normal form. Please notice that the position on the log has now a nested structure. Indeed it carries information about the virtual substitutions already performed.

F.2 The λ JAM

The execution on the λ JAM is very similar to the λ IAM one. Logged positions now save the whole term and log.

	Sub-term	Context	Log	Tape	Dir
	$(\lambda x.xx)(\lambda y.y)$	(\cdot)	ϵ	ϵ	\downarrow
$\rightarrow_{\bullet 1}$	$\underline{\lambda x.xx}$	$(\cdot)(\lambda y.y)$	ϵ	\bullet	\downarrow
$\rightarrow_{\bullet 2}$	\underline{xx}	$(\lambda x.(\cdot))(\lambda y.y)$	ϵ	ϵ	\downarrow
$\rightarrow_{\bullet 1}$	\underline{x}	$(\lambda x.(\cdot)x)(\lambda y.y)$	ϵ	\bullet	\downarrow
\rightarrow_{var}	$\lambda x.xx$	$(\cdot)(\lambda y.y)$	ϵ	$(x, (\lambda x.(\cdot)x)(\lambda y.y), \epsilon) \cdot \bullet$	\uparrow

The execution continues as in the λ IAM case. We put $l_x := (x, (\lambda x.(\cdot)x)(\lambda y.y), \epsilon)$.

	Sub-term	Context	Log	Tape	Dir
	$\lambda x.xx$	$(\cdot)(\lambda y.y)$	ϵ	$l_x \cdot \bullet$	\uparrow
\rightarrow_{arg}	$\underline{\lambda y.y}$	$(\lambda x.xx)(\cdot)$	l_x	\bullet	\downarrow
$\rightarrow_{\bullet 2}$	\underline{y}	$(\lambda x.xx)(\lambda y.(\cdot))$	l_x	ϵ	\downarrow
\rightarrow_{var}	$\lambda y.y$	$(\lambda x.xx)(\cdot)$	l_x	$(y, (\lambda x.x)(\lambda y.(\cdot)), l_x)$	\uparrow

In this situation the λ IAM would backtrack. The λ JAM, instead, directly jumps to the previously saved logged position.

	Sub-term	Context	Log	Tape	Dir
	$\lambda y.y$	$(\lambda x.xx)(\cdot)$	l_x	$(y, (\lambda x.x)(\lambda y.(\cdot)), l_x)$	\uparrow
\rightarrow_{jmp}	x	$(\lambda x.(\cdot)x)(\lambda y.y)$	ϵ	$(y, (\lambda x.x)(\lambda y.(\cdot)), l_x)$	\uparrow

The computation then resumes as is the λ IAM case. We put $l_y := (y, (\lambda x.x)(\lambda y.(\cdot)), l_x)$.

	Sub-term	Context	Log	Tape	Dir
	x	$(\lambda x.(\cdot)x)(\lambda y.y)$	ϵ	l_y	\uparrow
\rightarrow_{arg}	\underline{x}	$(\lambda x.x(\cdot))(\lambda y.y)$	l_y	ϵ	\downarrow
\rightarrow_{var}	$\lambda x.xx$	$(\cdot)(\lambda y.y)$	ϵ	$(x, (\lambda x.x(\cdot))(\lambda y.y), l_y)$	\uparrow
\rightarrow_{arg}	$\underline{\lambda y.y}$	$(\lambda x.xx)(\cdot)$	$(x, (\lambda x.x(\cdot))(\lambda y.y), l_y)$	ϵ	\downarrow

F.3 The KAM

The KAM, as the other machines, looks for the head variable of the term, inspecting its spine. Every time an application is encountered, the argument is saved in the stack, together with its environment. When abstractions are encountered, an entry in the environment is created, linking

the abstracted variable with the closure on the top of the stack.

	Sub-term	Context	Env.	Stack
	$(\lambda x.xx)(\lambda y.y)$	$\langle \cdot \rangle$	ϵ	ϵ
\rightarrow_{app}	$\lambda x.xx$	$\langle \cdot \rangle (\lambda y.y)$	ϵ	$(\lambda y.y, (\lambda x.xx) \langle \cdot \rangle, \epsilon)$
\rightarrow_{abs}	x	$(\lambda x.\langle \cdot \rangle) (\lambda y.y)$	$[x \leftarrow (\lambda y.y, (\lambda x.xx) \langle \cdot \rangle, \epsilon)] =: E$	ϵ
\rightarrow_{app}	x	$(\lambda x.\langle \cdot \rangle x) (\lambda y.y)$	E	$(x, (\lambda x.x \langle \cdot \rangle) (\lambda y.y), E)$

This way, when a variable is encountered, the KAM can hop directly to the sub-term it would be substituted for, just by inspecting the environment. Moreover, the right environment for the sub-term can be restored from its closure.

	Sub-term	Context	Env.	Stack
	x	$(\lambda x.\langle \cdot \rangle x) (\lambda y.y)$	E	$(x, (\lambda x.x \langle \cdot \rangle) (\lambda y.y), E)$
\rightarrow_{var}	$\lambda y.y$	$(\lambda x.xx) \langle \cdot \rangle$	ϵ	$(x, (\lambda x.x \langle \cdot \rangle) (\lambda y.y), E)$
\rightarrow_{abs}	y	$(\lambda x.xx) (\lambda y.\langle \cdot \rangle)$	$[y \leftarrow (x, (\lambda x.x \langle \cdot \rangle) (\lambda y.y), E)]$	ϵ
\rightarrow_{var}	x	$(\lambda x.x \langle \cdot \rangle) (\lambda y.y)$	E	ϵ
\rightarrow_{var}	$\lambda y.y$	$(\lambda x.xx) \langle \cdot \rangle$	ϵ	ϵ

F.4 The λ PAM

The λ PAM starts the computation looking for the head variable, namely x , traversing the spine of t . When it has been found, the machine, now in \uparrow mode, turns to query the argument, saving the variable position on the tape.

	Sub-term	Context	Hist.	Index	Tape	Dir
	$(\lambda x.xx)(\lambda y.y)$	$\langle \cdot \rangle$	ϵ	0	ϵ	\downarrow
$\rightarrow_{\bullet 1}$	$\lambda x.xx$	$\langle \cdot \rangle (\lambda y.y)$	ϵ	0	\bullet	\downarrow
$\rightarrow_{\bullet 2}$	x	$(\lambda x.\langle \cdot \rangle) (\lambda y.y)$	ϵ	0	ϵ	\downarrow
$\rightarrow_{\bullet 1}$	x	$(\lambda x.\langle \cdot \rangle x) (\lambda y.y)$	ϵ	0	\bullet	\downarrow
\rightarrow_{var}	$\lambda x.xx$	$\langle \cdot \rangle (\lambda y.y)$	ϵ	0	$(x, (\lambda x.\langle \cdot \rangle x) (\lambda y.y)) \bullet$	\uparrow

When the argument, namely $\lambda y.y$ has been found, the variable gets saved in the history permanently, together with the index needed to reconstruct the chain of virtual substitutions.

	Sub-term	Context	Hist.	Index	Tape	Dir
	$\lambda x.xx$	$\langle \cdot \rangle (\lambda y.y)$	ϵ	0	$p_x \bullet$	\uparrow
\rightarrow_{arg}	$\lambda y.y$	$(\lambda x.xx) \langle \cdot \rangle$	$(p_x, 0)$	1	\bullet	\downarrow
$\rightarrow_{\bullet 2}$	y	$(\lambda x.xx) (\lambda y.\langle \cdot \rangle)$	$(p_x, 0)$	1	ϵ	\downarrow
\rightarrow_{var}	$\lambda y.y$	$(\lambda x.xx) \langle \cdot \rangle$	$(p_x, 0)$	1	$(y, (\lambda x.x) (\lambda y.\langle \cdot \rangle))$	\uparrow

Jumps are handled by retrieving the position in the history at the current index, that is then decreased by one in order to coherently update the information associated to substitutions.

	Sub-term	Context	Hist.	Index	Tape	Dir
	$\lambda y.y$	$(\lambda x.xx) \langle \cdot \rangle$	$(p_x, 0)$	1	$(y, (\lambda x.x) (\lambda y.\langle \cdot \rangle))$	\uparrow
\rightarrow_{jmp}	x	$(\lambda x.\langle \cdot \rangle x) (\lambda y.y)$	$(p_x, 0)$	0	$(y, (\lambda x.x) (\lambda y.\langle \cdot \rangle))$	\uparrow

In transition \rightarrow_{var} , if the level of the binding context is greater than zero, one has to follow the chain of pointers in order to set the correct information about substitutions.

	Sub-term	Context	Hist.	Index	Tape	Dir
	x	$(\lambda x.\langle \cdot \rangle x) (\lambda y.y)$	$(p_x, 0)$	0	p_y	\uparrow
\rightarrow_{arg}	x	$(\lambda x.x \langle \cdot \rangle) (\lambda y.y)$	$(p_y, 0) \cdot (p_x, 0)$	2	ϵ	\downarrow
\rightarrow_{var}	$\lambda x.xx$	$\langle \cdot \rangle (\lambda y.y)$	$(p_y, 0) \cdot (p_x, 0)$	0	$(x, (\lambda x.x \langle \cdot \rangle) (\lambda y.y))$	\uparrow
\rightarrow_{arg}	$\lambda y.y$	$(\lambda x.xx) \langle \cdot \rangle$	$(p_{x_2}, 0) \cdot (p_y, 0) \cdot (p_x, 0)$	3	ϵ	\downarrow

In general, the way in which the λ PAM works is not very intuitive. The simplest way to understand it, is to think about the correspondence with the λ JAM. Given a log $L_n := (x, C, L') \cdot l_2 \cdots l_n$, a history H and index i , if $i = |H|$, then the λ PAM is considering the log L_n . Otherwise diminishing i by one and considering the position in the history corresponding to the index i amounts to consider the log L' . Finally, following the chain of pointers k times, *i.e.* considering as index $\phi_H^k(i)$ amounts to consider the log $l_{k+1} \cdots l_n$.

F.5 The SIAM

As we have already proved, the SIAM is strongly bisimilar to the λ IAM. One can indeed observe that the sequence of states reached by the SIAM is the same sequence of the λ IAM. The SIAM does not need additional data structures, since every sub-term has been already duplicated in advanced as many times as needed. The argument $\lambda y.y$, for example, is typed twice, since it is substituted for *two* occurrences of x during the evaluation of t .

$$\frac{\frac{\frac{x : [[\star] \rightarrow \star] \vdash x : [\star]_{\downarrow 10} \rightarrow \star_{\uparrow 4} \quad x : [\star] \vdash x : \star_{\uparrow 11}}{x : [\star, [\star] \rightarrow \star] \vdash xx : \star_{\uparrow 3}}}{\vdash \lambda x.x : [\star]_{\downarrow 12}, [\star]_{\uparrow 9} \rightarrow \star_{\downarrow 5} \rightarrow \star_{\uparrow 2}} \quad \frac{\frac{y : \star \vdash y : \star_{\uparrow 7}}{\vdash \lambda y.y : [\star]_{\downarrow 8} \rightarrow \star_{\uparrow 6}} \quad \vdash \lambda y.y : \star_{\uparrow 13}}{\vdash (\lambda x.xx)(\lambda y.y) : \star_{\uparrow 1}}}{\vdash (\lambda x.xx)(\lambda y.y) : \star_{\uparrow 1}}$$