# On Jumps, Interactions, and Intersection Types

**Stefano Catozi**
Université Sorbonne Paris Nord, Villetaneuse, France

**Ugo Dal Lago**
Università di Bologna, Italia
Inria, Sophia Antipolis, France

**Gabriele Vanoni**
IRIF, Université Paris Cité, France

## Abstract

The Jumping Abstract Machine (JAM), an evaluation mechanism for the $\lambda$-calculus, was introduced by Danos and Regnier as an optimization of the Interaction Abstract Machine (IAM), itself an operational counterpart to Girard's Geometry of Interaction and Abramsky *et al.* game semantics. Moreover, the JAM is isomorphic to the Pointer Abstract Machine (PAM), the syntactical counterpart of Hyland and Ong's game semantics. We study a generalization of the JAM, that we call the Parametric Jumping Abstract Machine (PaJAM) and show that there is a tight correspondence between the PaJAM and non-idempotent intersection types: given a normalizing term $t$, the number of steps taken by the PaJAM when evaluating $t$ can be extracted from its non-idempotent intersection type derivation. Remarkably, fixing the backtracking depth of the PaJAM, one can easily recover both the JAM/PAM, when the depth is constrained to be zero, and the IAM, when it is instead unconstrained. Exploiting type-theoretic machinery, we analyze the complexity of the PaJAM, showing that it is *polynomial* in the number of weak head $\beta$ steps, giving rise to a *reasonable* cost model, for each *finite* bound on the backtracking depth.

**2012 ACM Subject Classification** Theory of computation → Linear logic; Theory of computation → Lambda calculus; Theory of computation → Abstract machines; Theory of computation → Operational semantics

**Keywords and phrases** lambda-calculus, geometry of interaction, intersection types, abstract machines

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2026.64

**Related Version** *Full Version*: `https://arxiv.org/abs/2606.27062` [16]

**Funding** *Gabriele Vanoni*: Supported by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 101034255.

## 1 Introduction

This paper deals with two central concepts in the theory of the $\lambda$-calculus, namely *abstract machines* [29, 31, 4] on the one hand and *intersection types* [18, 15, 13] on the other. We are particularly interested in the relationship between these two concepts and our main contribution is a new generalized tight correspondence result. This will allow us to analyze the efficiency of a family of machines via type-theoretic tools. We give a bit of context before delving into the technical details.

**Abstract Machines.** An abstract machine is an automaton designed to evaluate a term of the $\lambda$-calculus (or of similar calculi), *i.e.* to compute a representation of its normal form. In doing so, the abstract machine usually conforms to a notion of reduction, e.g., *call-by-value* or *call-by-name* reduction [31]. Abstract machines, unlike mere rewriting, are thought of as (relatively) low-level, first-order, descriptions of the evaluation process, such that each

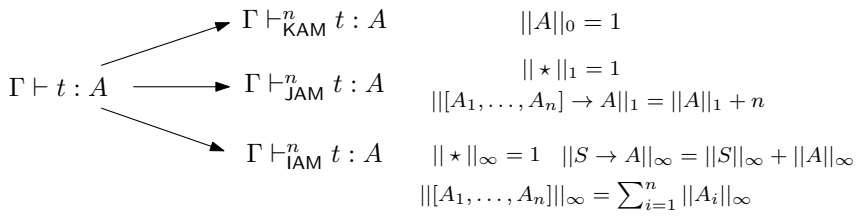
individual step is in some sense *elementary*. As it is well known, there are a variety of abstract machines, each with its own correctness and efficiency properties. Among the best-studied machines, there is the so-called Krivine Abstract Machine (KAM) [28], which implements call-by-name (a.k.a. weak-head) reduction. Machines which are similar in spirit to the Krivine machine but implementing call-by-value or call-by-need reduction have also been introduced [4, 5]. It is worth noticing that abstract machines have been studied in relation to their *complexity*. Indeed, one would like to execute $\lambda$-terms with a *polynomial* overhead with respect to the number of $\beta$-steps. This is important because abstract machines can then be implemented in Turing machines/RAMs again with a polynomial overhead, then providing a *reasonable* time cost model for the $\lambda$-calculus [2].

**Game Machines.** Since the pioneering work of Danos, Herbelin and Regnier [20], it is well known that there are machines implementing weak-head reduction like the KAM, but which have an even lower-level behavior, closely related to game semantics and Girard's geometry of interaction (GoI) [25]. In particular, the so-called Interaction Abstract Machine (IAM) [30, 21, 6] can be seen as an automata-theoretic counterpart to the GoI or to AJM games [1], and evaluates terms through a purely *local* and *reversible* process. There is also the so-called Pointer Abstract Machine (PAM) [20, 22], which is instead inspired, as its name says, by the mechanism of justification pointers from Hyland and Ong's games [26]. As discovered by Danos and Regnier [21], the PAM is isomorphic to the Jumping Abstract Machine (JAM), an optimization of the IAM obtained by allowing the latter to have, in certain circumstances, the possibility to *jump* from one subterm to another, thus losing locality and reversibility. These machines have been studied with respect to their correctness and relative performance. In particular, Accattoli, Dal Lago and Vanoni, have recently shown that the KAM can be seen as an improvement to the JAM (and thus of the PAM), and that the latter is an improvement to the IAM [7, 19]. Noticeably, while switching from the KAM to the JAM involves a *polynomial* overhead, the IAM can be *exponentially* less efficient than the JAM.

**Intersection Types and the KAM.** What role do intersection types play in all this? In their non-idempotent variant [24, 27], intersection types are known to be a syntactic counterpart of the *relational model* of linear logic [14]. Starting from de Carvalho's results [23], the size of the type derivation for a term $t$ has been put in relation to the number of reduction steps needed to evaluate the term $t$ using the KAM. The correspondence can be made tight by considering so-called *tight typings* [12] or by introducing a special type $\star$ corresponding to normal forms. In such correspondences, each KAM step is put in relation to the application of a typing rule, and this relation is bijective: subterms that are used many times appear many times in the derivation, while terms that are *not* used simply are not subject to typing.

**Intersection Types and Game Machines.** Accattoli *et al.* recently showed that the correspondence just mentioned, surprisingly, scales to the IAM [7]. Remarkably, this does not require any modification to the type system itself, but only amounts to altering the way each typing rule is weighted. While in the KAM each rule occurrence counts as *one*, the IAM requires assigning a weight equal to *the size of the conclusion type*. In other words, the number of IAM steps is precisely reflected by the number of occurrences of the type $\star$ to the right of $\vdash$ in the corresponding type derivation. Since types can be exponentially bigger than terms, this explains the aforementioned efficiency gap. But what about the JAM or the PAM? Would it be possible to precisely characterize the dynamic behavior of the JAM

by means of non-idempotent intersection types? More fundamentally: given that there is a phase transition between the JAM and the IAM, what is it that *drives* this transition? And why, by contrast, do the KAM and the JAM turn out to have both a polynomial overhead? These are precisely the questions that this work addresses. And the results we obtain confirm that non-idempotent intersection types are indeed a very flexible tool. Once again, and surprisingly, it is only a matter of modifying the weight given to each rule. If in the IAM we count *all* occurrences of $\star$ in the conclusion of each rule, the JAM requires to count only those occurrences of $\star$ that are at *a depth equal to* 0 *or* 1. As a result, the blowup can no longer be exponential and becomes polynomial instead. In other words, the JAM/PAM is much closer to the KAM than to the IAM, even if it was designed as an optimization of the latter. The situation, a bit more formally, is similar to the one in the figure below, where $n$ is the total size of the types occurring in the derivation.




As we explain below, however, we do much more, showing that the IAM and the JAM are both instances of a parametric version of the latter. Indeed, looking at the figure above, it is natural to wonder whether there is anything *in between* the JAM and the IAM in which some (but not all) occurrences of $\star$ at depth higher than 1 are counted. Moreover, are these intermediate machines efficient?

**Contributions.** This work introduces an abstract machine, called the Parametric Jumping Abstract Machine (PaJAM), obtained by generalizing Danos and Regnier's jumps so that they *do not* occur until a certain backtracking depth $k$. Indeed, the intuition is that the IAM computes using a possibly nested backtracking mechanism and jumping allows one to skip this backtracking phase. However, because of the nesting, one could be inside $k$ backtracking phases and decide at some point not to go one level deeper, and instead jump, thus remaining at depth $k$. More specifically, we develop the following results:

- We define a generalization of the JAM, the PaJAM, which is a parametric machine, in Sect. 3. The JAM can be recovered by setting the backtracking depth $k$ as 0, while the IAM can be seen as a degenerate version of the PaJAM where $k = \infty$.
- As shown in Sect. 5, the number of steps performed by the PaJAM when evaluating a term $t$ can be measured, following the spirit of figure above, by letting the size of a type depend on $k$ and counting occurrences of the base type $\star$ up to depth $2k + 1$:

$$\Gamma \vdash t : A \longrightarrow \Gamma \vdash^n_{\mathsf{PaJAM}(k)} t : A \qquad || \cdot ||_{2k+1} = \cdots$$

- These results are then used in Sect. 6 to show that every instance of the PaJAM with $k < \infty$ simulates $\beta$ reduction with a polynomial overhead, thus being time invariant.

There are two take-home messages: not only non-idempotent intersection types allow a fine-grained and direct analysis of the dynamic properties of reduction, even when the latter is performed by abstract machines, but the performances of the JAM (and thus of the PAM) and the IAM can be studied within a *single* framework, which also gives a very simple account of their relative performance.

## 2 The Closed Call-by-Name $\lambda$-Calculus

Let $\mathcal{V}$ be a countable set of *variables*. The *terms* of the $\lambda$-calculus are defined as follows:

$$t, u, r, w \quad ::= \quad x \in \mathcal{V} \;\mid\; \lambda x.t \;\mid\; tu$$

*Free* and *bound variables* are defined as usual: $\lambda x.t$ binds $x$ in $t$. Terms are considered modulo $\alpha$-equivalence, and $t\{x \leftarrow u\}$ denotes the capture-avoiding (meta-level) *substitution* of all free occurrences of $x$ in $t$ for $u$. *Contexts* are $\lambda$-terms containing exactly one occurrence of a special symbol, the hole $\langle\cdot\rangle$, intuitively standing for a removed subterm. Here we adopt *levelled 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:

$$C_0 \quad ::= \quad \langle\cdot\rangle \;\mid\; \lambda x.C_0 \;\mid\; C_0 t\ ; \qquad C_{n+1} \quad ::= \quad C_{n+1}t \;\mid\; \lambda x.C_{n+1} \;\mid\; tC_n.$$

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*. This operation can potentially capture free variables. We consider a *call-by-name* (CbN) operational semantics which we restrict to *closed* terms, *i.e.* without any free variable. This is defined as follows:

$$(\lambda x.t)ur_1 \dots r_h \quad \to_{wh} \quad t\{x \leftarrow u\}r_1 \dots r_h \qquad\qquad h \geq 0.$$

This way, $\lambda$-abstractions are the only normal forms.

For any notion of reduction $\to$, we denote by $\to^*$ the reflexive and transitive closure of $\to$, and we define $\to^n$ by induction as $\to^0 := \mathsf{id}$ (the identity relation), and $\to^{n+1} := \to \to^n$.

## 3 The Parametric JAM

The Jumping Abstract Machine (JAM) is introduced in [21] as an optimization of the interaction abstract machine (IAM) [30, 21, 6]. The difference between them lies in the fact that a complex, and possibly nested, *backtracking* mechanism, which is the peculiarity of the IAM, is short-circuited in the JAM, making it more efficient. In this section, we introduce a new machine, the Parametric JAM (or PaJAM), which is a machine that behaves like the IAM until a fixed backtracking nesting depth (specified by the *initial parameter* of the machine), and then proceeds, avoiding backtracking, as the JAM.

Intuitively, the behaviour of the PaJAM, generalizing that of the IAM and the JAM, can be seen as that of a token that travels around the syntax tree of the program under evaluation, possibly making some *jumps*. The evaluation is done in two phases. In one, noted $\downarrow$, the machine looks for the head variable, while in the other one, noted $\uparrow$, the machine looks for the argument which should replace it. We redirect the reader to [6, 7] for detailed explanations on the behavior of this family of machines.

**PaJAM States.** The transitions of the PaJAM and all the data structures are defined in Fig. 1. As we can see from the definition, the machine relies on two mutually inductive data structures, namely *logs* and *logged positions*. We note with $L$ a log of arbitrary length. In order to implement the optimization mechanism introduced earlier, each machine state carries an integer parameter $k \in \mathbb{N} \cup \{\infty\}$, the *backtracking (nesting) depth*, indicating how many backtracking levels the machine may still enter. Initial states have the form $q_t^k := (t, \langle\cdot\rangle, \epsilon, \epsilon, k)$, with $k$ being the initial parameter of the machine. 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.

| | | | |
|---|---|---|---|
| Logged Positions | $l ::= (t, C_n, L_n)$ | Tapes | $T ::= \epsilon \mid \bullet \cdot T \mid l \cdot T$ |
| Logs | $L_0 ::= \epsilon \quad L_{n+1} ::= l \cdot L_n$ | Dir. | $d ::= \downarrow \mid \uparrow$ |
| States | $q ::= (t, C, L, T, d, k)$ | | $k \in \mathbb{N} \cup \{\infty\}$ |

| Sub-term | Context | Log | Tape | Dep. | | Sub-term | Context | Log | Tape | Dep. |
|---|---|---|---|---|---|---|---|---|---|---|
| $\underline{tu}$ | $C$ | $L$ | $T$ | $k$ | $\to_{\bullet 1}$ | $\underline{t}$ | $C\langle\langle\cdot\rangle u\rangle$ | $L$ | $\bullet \cdot T$ | $k$ |
| $\underline{\lambda x.t}$ | $C$ | $L$ | $\bullet \cdot T$ | $k$ | $\to_{\bullet 2}$ | $\underline{t}$ | $C\langle\lambda x.\langle\cdot\rangle\rangle$ | $L$ | $T$ | $k$ |
| $\underline{x}$ | $C\langle\lambda x.D_n\rangle$ | $L_n \cdot L$ | $T$ | $k$ | $\to_{\mathsf{var}}$ | $\lambda x.D_n\langle x\rangle$ | $\underline{C}$ | $L$ | $l' \cdot T$ | $k$ |
| $\underline{\lambda x.D_n\langle x\rangle}$ | $C$ | $L$ | $l' \cdot T$ | $k$ | $\to_{\mathsf{bt2}}$ | $x$ | $\underline{C\langle\lambda x.D_n\rangle}$ | $L_n \cdot L$ | $T$ | $k{+}1$ |
| $t$ | $\underline{C\langle\langle\cdot\rangle u\rangle}$ | $L$ | $\bullet \cdot T$ | $k$ | $\to_{\bullet 3}$ | $tu$ | $\underline{C}$ | $L$ | $T$ | $k$ |
| $t$ | $\underline{C\langle\lambda x.\langle\cdot\rangle\rangle}$ | $L$ | $T$ | $k$ | $\to_{\bullet 4}$ | $\lambda x.t$ | $\underline{C}$ | $L$ | $\bullet \cdot T$ | $k$ |
| $t$ | $\underline{C\langle\langle\cdot\rangle u\rangle}$ | $L$ | $l \cdot T$ | $k$ | $\to_{\mathsf{arg}}$ | $\underline{u}$ | $C\langle t\langle\cdot\rangle\rangle$ | $l \cdot L$ | $T$ | $k$ |
| $t$ | $\underline{C\langle u\langle\cdot\rangle\rangle}$ | $(x, D, L') \cdot L$ | $T$ | $0$ | $\to_{\mathsf{jmp}}$ | $x$ | $\underline{D}$ | $L'$ | $T$ | $0$ |
| $t$ | $\underline{C\langle u\langle\cdot\rangle\rangle}$ | $l \cdot L$ | $T$ | $k{+}1$ | $\to_{\mathsf{bt1}}$ | $\underline{u}$ | $C\langle\langle\cdot\rangle t\rangle$ | $L$ | $l \cdot T$ | $k$ |

where $l' := (x, C\langle\lambda x.D_n\rangle, L_n \cdot L)$.

**Figure 1** The data structures and transitions of the Parametric JAM (PaJAM).

**Transitions.** The transitions of the PaJAM are in Fig. 1 and their union is noted $\to_{\text{PaJAM}}$. Note that the machine is deterministic, but not reversible. A *run* $\rho : q \to^* q'$ is a possibly empty sequence of transitions, whose length is noted $|\rho|$. If it starts from an initial state it is called *initial*, and if moreover it ends on a final state it is called *complete*. Consider an execution of the PaJAM starting from the initial state $q_t^k$, only two cases can happen: either the execution diverges or it terminates in a state $q$ of the form $q = (\underline{\lambda x.u}, C, L, \epsilon, k)$. We call such states *final*. The fact that no other shapes are possible can proved as in [7]. Notice how the parameter $k$ is decremented by 1 after entering a backtracking phase via a $\to_{\mathsf{bt1}}$ transition and is incremented by 1 after exiting a backtracking with a $\to_{\mathsf{bt2}}$ transition. When $k$ reaches 0, the $\to_{\mathsf{bt1}}$ transition is replaced by a $\to_{\mathsf{jmp}}$, allowing the machine to avoid the backtracking runs after a certain depth.

▶ Remark 1. It is clear that if we start from an initial state $q_t^0$, the PaJAM run never performs backtracking and behaves exactly like the JAM [7]. On the other hand, when the initial parameter is set to $\infty$, no jump is performed, the PaJAM thus behaving like the IAM of [6].

The IAM is well-known to be correct for Closed CbN. This property is extended to the JAM in [7], using a simulation argument. The proof smoothly adapts to the PaJAM, and this is why we do not detail it here.

▶ **Proposition 2** (Correctness). *$q_t^k$ terminates if and only if $t$ has normal form.*

The direction of reachable states is directly related to the backtracking nesting depth, as the next invariant shows. Let $|T|_l$ be the number of logged position in a tape $T$.

▶ **Proposition 3** (Phase Invariant). *Let $s = (t, C, L, T, d, k')$ be a* PaJAM *state reachable from $q_u^{k<\infty}$. If $d = \uparrow$ then $|T|_l = 2(k - k')$ and if $d = \downarrow$ then $|T|_l = 2(k - k') + 1$.*

**An Example.** We give an example of execution of the PaJAM with parameter $k = 1$. The first part of the execution looks for the head variable of the term $(\lambda x.xx)(\lambda y.y)$, namely $x$.

| | Sub-term | Context | Log | Tape | Dir | Depth |
|---|---|---|---|---|---|---|
| | $\underline{(\lambda x.xx)(\lambda y.y)}$ | $\langle\cdot\rangle$ | $\epsilon$ | $\epsilon$ | $\downarrow$ | 1 |
| $\to_{\bullet 1}$ | $\underline{\lambda x.xx}$ | $\langle\cdot\rangle(\lambda y.y)$ | $\epsilon$ | $\bullet$ | $\downarrow$ | 1 |
| $\to_{\bullet 2}$ | $\underline{xx}$ | $(\lambda x.\langle\cdot\rangle)(\lambda y.y)$ | $\epsilon$ | $\epsilon$ | $\downarrow$ | 1 |
| $\to_{\bullet 1}$ | $\underline{x}$ | $(\lambda x.\langle\cdot\rangle x)(\lambda y.y)$ | $\epsilon$ | $\bullet$ | $\downarrow$ | 1 |
| $\to_{\mathsf{var}}$ | $\lambda x.xx$ | $\underline{\langle\cdot\rangle(\lambda y.y)}$ | $\epsilon$ | $(x, (\lambda x.\langle\cdot\rangle x)(\lambda y.y), \epsilon) \cdot \bullet$ | $\uparrow$ | 1 |

Once the variable head $x$ is found, the machine switches to $\uparrow$ and starts to search for its argument, while saving the logged position of $x$ on the tape. We set $l_x := (x, (\lambda x.\langle\cdot\rangle x)(\lambda y.y), \epsilon)$.

| | Sub-term | Context | Log | Tape | Dir | Depth |
|---|---|---|---|---|---|---|
| | $\lambda x.xx$ | $\underline{\langle\cdot\rangle(\lambda y.y)}$ | $\epsilon$ | $l_x \cdot \bullet$ | $\uparrow$ | 1 |
| $\to_{\mathsf{arg}}$ | $\underline{\lambda y.y}$ | $(\lambda x.xx)\langle\cdot\rangle$ | $l_x$ | $\bullet$ | $\downarrow$ | 1 |
| $\to_{\bullet 2}$ | $\underline{y}$ | $(\lambda x.xx)(\lambda y.\langle\cdot\rangle)$ | $l_x$ | $\epsilon$ | $\downarrow$ | 1 |
| $\to_{\mathsf{var}}$ | $\lambda y.y$ | $\underline{(\lambda x.xx)\langle\cdot\rangle}$ | $l_x$ | $(y, (\lambda x.xx)(\lambda y.\langle\cdot\rangle), l_x)$ | $\uparrow$ | 1 |

The $\to_{\mathsf{arg}}$ transition saves the position $l_x$ on the log and the machine starts looking for the head variable of the argument. Having found it, it saves its position on the tape. Now the machine starts looking for the argument of $\lambda y.y$ in $\uparrow$ mode. However, $\lambda y.y$ was not the left side of an application forming a $\beta$-redex. Indeed, it was virtually substituted for the first occurrence of $x$, thus creating the "virtual" redex $(\lambda y.y)x$. Its argument is thus the second occurrence of $x$. Since the backtracking parameter is 1, the machine is allowed to perform one level of backtracking. It therefore starts a backtracking run with a $\to_{\mathsf{bt1}}$ transition, in order to find the variable associated with the logged position $l_x$ stored in the log. We set $l_y := (y, (\lambda x.x)(\lambda y.\langle\cdot\rangle), l_x)$.

| | Sub-term | Context | Log | Tape | Dir | Depth |
|---|---|---|---|---|---|---|
| | $\lambda y.y$ | $\underline{(\lambda x.xx)\langle\cdot\rangle}$ | $(x, (\lambda x.\langle\cdot\rangle x)(\lambda y.y), \epsilon)$ | $l_y$ | $\uparrow$ | 1 |
| $\to_{\mathsf{bt1}}$ | $\underline{\lambda x.xx}$ | $\langle\cdot\rangle(\lambda y.y)$ | $\epsilon$ | $(x, (\lambda x.\langle\cdot\rangle x)(\lambda y.y), \epsilon) \cdot l_y$ | $\downarrow$ | 0 |
| $\to_{\mathsf{bt2}}$ | $x$ | $\underline{(\lambda x.\langle\cdot\rangle x)(\lambda y.y)}$ | $\epsilon$ | $l_y$ | $\uparrow$ | 1 |

If instead we had started with an initial parameter equal to 0, the machine would have skipped the backtracking run between $\to_{\mathsf{bt1}}$ and $\to_{\mathsf{bt2}}$, and would have jumped directly to the position stored in $l_x$, as shown below.

| | Sub-term | Context | Log | Tape | Dir | Depth |
|---|---|---|---|---|---|---|
| | $\lambda y.y$ | $\underline{(\lambda x.xx)\langle\cdot\rangle}$ | $(x, (\lambda x.\langle\cdot\rangle x)(\lambda y.y), \epsilon)$ | $l_y$ | $\uparrow$ | 0 |
| $\to_{\mathsf{jmp}}$ | $x$ | $\underline{(\lambda x.\langle\cdot\rangle x)(\lambda y.y)}$ | $\epsilon$ | $l_y$ | $\uparrow$ | 0 |

Notice that the states reached after $\to_{\mathsf{bt2}}$ and $\to_{\mathsf{jmp}}$ are the same, except for the value of the parameter. Continuing the original run, the machine can now find immediately the argument, being the second occurrence of $x$. Then, it starts looking for the argument of this variable, finding of course a new copy of $\lambda y.y$.

| | Sub-term | Context | Log | Tape | Dir | Depth |
|---|---|---|---|---|---|---|
| | $x$ | $\underline{(\lambda x.\langle\cdot\rangle x)(\lambda y.y)}$ | $\epsilon$ | $l_y$ | $\uparrow$ | 1 |
| $\to_{\mathsf{arg}}$ | $\underline{x}$ | $(\lambda x.x\langle\cdot\rangle)(\lambda y.y)$ | $l_y$ | $\epsilon$ | $\downarrow$ | 1 |
| $\to_{\mathsf{var}}$ | $\lambda x.xx$ | $\underline{\langle\cdot\rangle(\lambda y.y)}$ | $\epsilon$ | $(x, (\lambda x.x\langle\cdot\rangle)(\lambda y.y), l_y)$ | $\uparrow$ | 1 |
| $\to_{\mathsf{arg}}$ | $\underline{\lambda y.y}$ | $(\lambda x.xx)\langle\cdot\rangle$ | $(x, (\lambda x.x\langle\cdot\rangle)(\lambda y.y), l_y)$ | $\epsilon$ | $\downarrow$ | 1 |

The computation then stops, signaling that the term has a normal form.

## 4 The Sequence PaJAM

In this section, we present the definition of the Sequence Parametric JAM (SPaJAM), which is the type theoretic version of the PaJAM and which is strongly bisimilar to it. Instead of being defined in an automata-theoretic way, the SPaJAM is defined as a machine *acting on* the intersection type derivation for the term under evaluation. Indeed, abstracting from the concrete data structures, as done in [7, 8], allows us to better reason about the complexity of the machine. We do not have here the space to explain in detail how the type system works, and thus we refer to [15, 13].

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

$$\frac{\Gamma \vdash^{w} t : [A'_1, \ldots, A'_n] \to A \quad [\Delta_i \vdash^{v_i} u : A'_i]_{i\in[1,\ldots,n]}}{\Gamma \uplus_i \Delta_i \vdash^{w+\sum_i v_i+\|A\|_k} tu : A}\ \text{T-@}$$

**Figure 2** The weighted sequence type system.

**Sequence Types.** We define sequence types, or non-idempotent and not commutative intersection types, as follows:

$$\begin{array}{rrcl} \textsc{Linear types} & A, A' & ::= & \star \mid S \to A \\ \textsc{Sequence types} & S, S' & ::= & [A_1, \ldots, A_n] \quad n \geq 0 \end{array}$$

Please note that there is only one ground type $\star$. This type will be used to type normal forms, and these are precisely abstractions in Closed CbN. The empty sequence is denoted by $[\,]$ and is used to type subterms which will be discarded along the evaluation. The concatenation of two sequences $S$ and $S'$ is denoted by $S \uplus S'$. Please note that the type system is *linear*. This means that weakening and contraction are not allowed. Type judgments have the form $\Gamma \vdash t : A$, where $\Gamma$ is a type environment, *i.e.* a total function from variables to sequence types that maps all but finitely many variables to $[\,]$. We write $\Gamma = x_1 : S_1, \ldots, x_n : S_n$ if $\mathsf{dom}(\Gamma) = \{x_1, \ldots, x_n\}$, where $\mathsf{dom}(\Gamma)$ is defined as the (finite) set of variables in $\Gamma$ that are mapped to non-empty sequence types. The union of two type environments $\Gamma, \Delta$ is denoted by $\Gamma \uplus \Delta$ and stands for the type environment mapping every variable $x \in \mathsf{dom}(\Gamma) \cup \mathsf{dom}(\Delta)$ to the sequence $\Gamma(x) \uplus \Delta(x)$. The typing rules are in Figure 2, please ignore the weights for now. Type derivations are denoted by $\pi$, and we write $\pi \triangleright \Gamma \vdash t : A$ to denote a type derivation $\pi$ ending in the judgment $\Gamma \vdash t : A$.

**The SPaJAM.** The idea behind the SPaJAM is simple: the machine moves around a fixed type derivation $\pi \triangleright \vdash t : \star$, to be thought of as the code. The *focus* of the machine is expressed as an occurrence of a type judgment $J$ of $\pi$. Like the PaJAM, the SPaJAM has two possible directions, noted $\downarrow$ and $\uparrow$.[1] In direction $\uparrow$ the machine looks at the rule above the focused judgment, in direction $\downarrow$ at the rule below. Each state contains a type context $\mathbb{A}$ isolating an occurrence of $\star$ in the type $A$ of the focused judgment (occurrence) $\Gamma \vdash u : A$, defined as follows:

$$\begin{array}{rrcl} \textsc{Type ctxs} & \mathbb{A} & ::= & \langle\cdot\rangle \mid S \to \mathbb{A} \mid \mathbb{S} \to A \\ \textsc{Sequence ctxs} & \mathbb{S} & ::= & [A_1, .., \mathbb{A}, .., A_n] \quad n \geq 0 \end{array}$$

Moreover, as in the PaJAM, each state carries a parameter $k \in \mathbb{N} \cup \{\infty\}$. Summing up, a state $q$ is a quintuple $(\pi, J, \mathbb{A}, d, k)$. If $J$ is in the form $\Gamma \vdash u : A$, we often write $q$ as $\vdash u : \mathbb{A}\langle\star^k_d\rangle$, where $\mathbb{A}\langle\star\rangle = A$, in fact type environments play no role here. Given a derivation $\pi \triangleright \vdash t : \star$, the initial state of the machine will be $(\pi, J, \langle\cdot\rangle, \uparrow, k)$, with $J$ the final judgment of $\pi$ and $k \in \mathbb{N} \cup \{\infty\}$ the initial parameter of the machine. Then, it moves from judgment to judgment, following occurrences of $\star$ around $\pi$. The transitions are in Fig. 3, their union noted $\to_{\text{SPaJAM}}$. They have the same labels as PaJAM transitions, because they correspond to each other, as we shall show. As in the case of the PaJAM, the SPaJAM is deterministic.

[1] Type derivations are upside-down w.r.t. to the term structure, then direction $\downarrow$ of the PaJAM becomes here $\uparrow$, and $\uparrow$ is $\downarrow$.

$$\frac{\vdash t : S \to A \quad [\vdash]}{\vdash tu : \mathbb{A}\langle\star^k_\uparrow\rangle(= A)} \to_{\bullet 1} \frac{\vdash t : S \to \mathbb{A}\langle\star^k_\uparrow\rangle \quad [\vdash]}{\vdash tu : A} \qquad \frac{\vdash t : A(= \mathbb{A}\langle\star\rangle)}{\vdash \lambda x.t : S \to \mathbb{A}\langle\star^k_\uparrow\rangle} \to_{\bullet 2} \frac{\vdash t : \mathbb{A}\langle\star^k_\uparrow\rangle}{\vdash \lambda x.t : S \to A}$$

$$\frac{\vdash t : S \to \mathbb{A}\langle\star^k_\downarrow\rangle \quad [\vdash]}{\vdash tu : A(= \mathbb{A}\langle\star\rangle)} \to_{\bullet 3} \frac{\vdash t : S \to A \quad [\vdash]}{\vdash tu : \mathbb{A}\langle\star^k_\downarrow\rangle} \qquad \frac{\vdash t : \mathbb{A}\langle\star^k_\downarrow\rangle(= A)}{\vdash \lambda x.t : S \to A} \to_{\bullet 4} \frac{\vdash t : A}{\vdash \lambda x.t : S \to \mathbb{A}\langle\star^k_\downarrow\rangle}$$

$$\frac{\dfrac{}{\vdash x : \mathbb{A}\langle\star^k_\uparrow\rangle_i(= A_i)}\, i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : [...A_i...] \to A'}{\to_{\mathsf{var}}} \frac{\dfrac{}{\vdash x : A_i}\, i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : [...\mathbb{A}\langle\star^k_\downarrow\rangle_i...] \to A'}$$

$$\frac{\dfrac{}{\vdash x : A_i(= \mathbb{A}\langle\star\rangle_i)}\, i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : [...\mathbb{A}\langle\star^k_\uparrow\rangle_i...] \to A'}{\to_{\mathsf{bt2}}} \frac{\dfrac{}{\vdash x : \mathbb{A}\langle\star^{k+1}_\downarrow\rangle_i}\, i \\ \vdots \\ \vdash \lambda x.C\langle x\rangle : [...A_i...] \to A'}$$

$$\frac{\vdash t : [...\mathbb{A}\langle\star^k_\downarrow\rangle_i...] \to A' \quad \vdash_i u : A_i(= \mathbb{A}\langle\star\rangle_i)}{\vdash tu : A'} \to_{\mathsf{arg}} \frac{\vdash t : [...A_i...] \to A' \quad \vdash_i u : \mathbb{A}\langle\star^k_\uparrow\rangle_i}{\vdash tu : A'}$$

$$\frac{\vdash t : [...A_i...] \to A' \quad \vdash_i u : \mathbb{A}\langle\star^{k+1}_\downarrow\rangle_i(= A_i)}{\vdash tu : A'} \to_{\mathsf{bt1}} \frac{\vdash t : [...\mathbb{A}\langle\star^k_\uparrow\rangle_i...] \to A' \quad \vdash_i u : A_i}{\vdash tu : A'}$$

$$\frac{\vdash t : [...A_i...] \to A' \quad q := \vdash_i u : \mathbb{A}\langle\star^0_\downarrow\rangle_i(= A_i)}{\vdash tu : A'} \to_{\mathsf{jmp}} \mathsf{jump}(q) = \frac{}{\vdash x : \mathbb{A}\langle\star^0_\downarrow\rangle_i(= A_i)}$$

where $\mathsf{jump}(\vdash u : \mathbb{A}\langle\star^0_\downarrow\rangle)$ is the first encountered state $\vdash x : \mathbb{A}\langle\star^0_\downarrow\rangle$ such that $\vdash u : \mathbb{A}\langle\star^\infty_\downarrow\rangle \to^+_{\mathrm{SPaJAM}} \vdash x : \mathbb{A}\langle\star^\infty_\downarrow\rangle$.

**Figure 3** The transitions of the Sequence Parametric JAM (SPaJAM).

The SPaJAM is a parametric version of the SIAM of [7]. Since most of its transitions coincide with those of the original machine, we refer to that work for a detailed explanation. We discuss the behavior of the parameter $k$. As in the PaJAM, the parameter $k$ is decremented after a transition $\to_{\mathsf{bt1}}$ and incremented after a transition $\to_{\mathsf{bt2}}$. When $k$ reaches 0, the backtracking mechanism is disabled and the $\to_{\mathsf{jmp}}$ transition is enabled. This means that the transitions $\to_{\mathsf{bt1}}$ and $\to_{\mathsf{bt2}}$, are substituted by the non-local "macro" transition $\to_{\mathsf{jmp}}$ that simulates the jump of the PaJAM. In fact, on type derivations we do not have logged positions, and thus we cannot define the jump in a direct way. Please note that *a priori* $\mathsf{jump}(\cdot)$ as defined in Fig. 3 is a partial function. Indeed, it is not obvious that there always exists a run such as the one described in the definition of the $\mathsf{jump}(\cdot)$ function. We will prove in the following section that indeed $\mathsf{jump}(\cdot)$ is total if restricted to reachable states.

▶ **Example 4.** We present here the execution on the SPaJAM on the same term analyzed in Sect. 3, with parameter $k = 1$. We have reported its type derivation, with the occurrences of $\star$ on the right of $\vdash$ annotated with increasing integers, a direction and a backtracking parameter. The occurrence of $\star$ marked with 1 represents the first state, and so on.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : [[\star] \to \star] \vdash x : [\star^1_{\downarrow 10}] \to \star_{\uparrow 4}} \quad \dfrac{}{x : [\star] \vdash x : \star^1_{\uparrow 11}}}{x : [[\star] \to \star, \star] \vdash xx : \star^1_{\uparrow 3}}}{\vdash \lambda x.xx : [[\star^0_{\uparrow 9}] \to \star^1_{\downarrow 5}, \star^1_{\downarrow 12}] \to \star^1_{\uparrow 2}} \quad \dfrac{\dfrac{}{y : \star \vdash y : \star^1_{\uparrow 7}}}{\vdash \lambda y.y : [\star^1_{\downarrow 8}] \to \star^1_{\uparrow 6}} \quad \dfrac{}{\vdash \lambda y.y : \star^1_{\uparrow 13}}}{\vdash (\lambda x.xx)(\lambda y.y) : \star^1_{\uparrow 1}}$$

One can immediately notice that every occurrence of $\star$ is visited exactly once. Also, the sequence of the visited subterms and transitions is the same as the one obtained in the example of Sect. 3. As we did for the example on the PaJAM, we now provide an execution with initial parameter $k = 0$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : [[\star] \to \star] \vdash x : [\star^0_{\downarrow 9}] \to \star^0_{\uparrow 4}} \quad \dfrac{}{x : [\star] \vdash x : \star^0_{\uparrow 10}}}{x : [[\star] \to \star, \star] \vdash xx : \star^0_{\uparrow 3}}}{\vdash \lambda x.xx : [[\star] \to \star^0_{\downarrow 5}, \star^0_{\downarrow 11}] \to \star^0_{\uparrow 2}} \quad \dfrac{\dfrac{}{y : \star \vdash y : \star^0_{\uparrow 7}}}{\vdash \lambda y.y : [\star^0_{\downarrow 8}] \to \star^0_{\uparrow 6}} \quad \dfrac{}{\vdash \lambda y.y : \star^0_{\uparrow 12}}}{\vdash (\lambda x.xx)(\lambda y.y) : \star^0_{\uparrow 1}}$$

Notice that the two runs are identical up to the eighth state. In the latter example, the backtracking depth of this state is 0, so the machine jumps directly to the axiom corresponding to $x$. In the former on the other hand, the machine initiates a backtracking run, passing once again through the subterm $\lambda x.xx$ before reaching the axiom.

The PaJAM and the SPaJAM are strongly bisimilar. The core idea underlying this bisimulation is that each state of the SPaJAM encodes the log of the PaJAM in the judgment occurrence and the tape in the type context. Since the proof of this correspondence essentially follows the same argument as in [7, 8], we defer the technical details to the technical report [16] and present here only its main consequence.

▶ **Proposition 5** (PaJAM-SPaJAM Bisimulation). *The* PaJAM *and the* SPaJAM *are strongly bisimilar.*

In particular, inside the technical development one proves that, whenever a $\to_{\mathsf{jmp}}$ transition is performed from a reachable SPaJAM state, there always exists a run respecting the definition of the $\mathsf{jump}(\cdot)$ function given above.

▶ **Proposition 6** (Well-Definedness of Jump). *Let $q$ be a reachable* SPaJAM *state. Then* $\mathsf{jump}(q)$ *is well-defined.*

## 5 Measuring the PaJAM Runtime via Sequence Types

In this section, we introduce the weight assignment system $\mathbf{W}^k_{\mathrm{PaJAM}}(\pi)$ and show that the weight of a typed $\lambda$-term $t$ corresponds exactly to the length of the complete PaJAM run on $t$. The main idea we rely on is that some occurrences of $\star$ in types appearing on the right-hand side of judgments within a derivation $\pi \triangleright \vdash t : \star$ correspond to the states of the PaJAM run starting from $t$. Therefore, to determine the number of states reached by this run, we have to count the number of times such occurrences of $\star$ appear in $\pi$. As we have seen, the $\to_{\mathsf{jmp}}$ transition enables the machine to skip certain parts of the computation through a mechanism that depends on the initial parameter $k$. In Example 4, we see that when the initial parameter is $k = 0$, the occurrences of $\star$ that are visited are precisely those located within at most one sequence. In contrast, when $k = 1$, the machine also reaches an occurrence of $\star$ nested within two sequences. We must therefore study how the presence of this parameter affects the occurrences of $\star$ in $\pi$ that are reachable by the SPaJAM from an initial state $q_t^{k<\infty}$.

We first define formally the notion of depth of a type context:

$$\mathsf{depth}(\langle\cdot\rangle) := 0 \quad \mathsf{depth}(S \to \mathbb{A}) := \mathsf{depth}(\mathbb{A}) \quad \mathsf{depth}([...\mathbb{A}...] \to A) := 1 + \mathsf{depth}(\mathbb{A})$$

Then, we prove that all the SPaJAM states which are reachable from $q_t^{k<\infty}$ have a type context of depth at most $2k+1$. As explained in the previous section, the bisimulation relation connects the type context of a SPaJAM state with the tape of the corresponding PaJAM state. In particular, given a SPaJAM state $\vdash u : \mathbb{A}\langle\star_d^k\rangle$, the number of logged positions on the tape of the corresponding PaJAM state is exactly $\mathsf{depth}(\mathbb{A})$. In Example 4, we observe a similar phenomenon in the SPaJAM. When $k=0$, the states reached by the PaJAM have at most one logged position in the tape. In contrast, when $k=1$, a state with two logged positions on the tape is reached. The following result can be seen as the type-theoretic counterpart of Prop. 3.

▶ **Proposition 7** (SPaJAM Phase Invariant). *Let $q := \vdash t : \mathbb{A}\langle\star_d^p\rangle$ be a* SPaJAM *state reachable by the initial state $q_u^k$. If $d = \uparrow$, then $\mathsf{depth}(\mathbb{A}) = 2(k-p)$, otherwise if $d = \downarrow$, then $\mathsf{depth}(\mathbb{A}) = 2(k-p)+1$. In particular, $\mathsf{depth}(\mathbb{A}) \leq 2k+1$.*

We define a norm on linear and sequence types, which counts the number of occurrences of the ground type $\star$ inside at most $k \geq 1$ nested sequences:

$$\begin{array}{rcl} \|\star\|_k & := & 1 \\ \|S \to A\|_k & := & \|S\|_k + \|A\|_k \end{array} \qquad \big\|[A_1,\ldots,A_n]\big\|_k := \begin{cases} n & \text{if } k=1 \\ \sum_{1\leq i\leq n} \|A_i\|_{k-1} & \text{if } k>1 \end{cases}$$

This is extended to type derivations with the weight system $\mathbf{W}^k_{\mathrm{PaJAM}}(\cdot)$ of Fig. 2.

▶ **Proposition 8** (Weights Bound PaJAM Time). *Let $\pi \triangleright \vdash t : \star$ be a type derivation and $n$ the length of the complete* PaJAM *run from the initial state $q_t^k$. Then $n \leq \mathbf{W}^{2k+1}_{\mathrm{PaJAM}}(\pi)$.*

**Proof.** By the definition of the weight assignment system, $\mathbf{W}^{2k+1}_{\mathrm{PaJAM}}(\pi)$ is the number of occurrences of $\star$ in $\pi$ having depth less or equal than $2k+1$. Since by the proposition above all reachable SPaJAM states having depth less or equal than $2k+1$, the result follows by the bisimulation between the PaJAM and the SPaJAM (Prop. 5). ◀

Of course, a priori it could be the case that there are occurrences of $\star$ at depth less or equal than $2k+1$ and which are *not* reachable SPaJAM states. We should rule out this case in order to turn the upper bound of the proposition above into an exact measure. We prove this fact in several steps. First, we need to recall a result about the SIAM, which corresponds in our setting to the SPaJAM with initial parameter $\infty$.

▶ **Proposition 9** (All $\star$ Are Reachable, [7]). *Let $\pi \triangleright \vdash t : \star$ be a type derivation. Then every occurrence of $\star$ corresponds to a state reachable by the* PaJAM *from the initial state $q_t^\infty$.*

If we fix $k \neq \infty$, the occurrences of $\star$ reachable by the SPaJAM from the initial state $q_t^k$ are clearly a subset of those reachable from the initial state $q_t^\infty$. In particular, by the very definition of the SPaJAM, it is clear that those occurrences of $\star$ that are reachable from $q_t^\infty$ but not from $q_t^k$ correspond precisely to the states that are *jumped*. Moreover, whenever a $\to_{\mathsf{jmp}}$ transition is performed starting from $q_t^k$, there exists a corresponding run starting from $q_t^\infty$ that reaches those jumped states. Having observed this, it is enough to prove that all such jumped states have depth strictly greater than $2k+1$. We prove this result in the technical report [16], by strengthening an invariant needed in the bisimulation proof.

▶ **Proposition 10** (SPaJAM Jump Runs Depth). *Let $q'$ be a state reachable by the* SPaJAM *with initial parameter $k$ and $q' =: (\pi, J', \mathbb{A}, \downarrow, 0) \to_{\mathsf{jmp}} (\pi, J, \mathbb{A}, \downarrow, 0) := q$. Then there exists a run $\rho : (\pi, J', \mathbb{A}, \downarrow, \infty) \to_{\mathsf{bt1}} s \to^*_{\mathrm{SPaJAM}} s' \to_{\mathsf{bt2}} (\pi, J, \mathbb{A}, \downarrow, \infty)$ such that for each state $s'' = (\pi, J'', \mathbb{A}', d, \infty)$ in $\sigma : s \to^*_{\mathrm{SPaJAM}} s'$ we have $\mathsf{depth}(\mathbb{A}') > 2k+1$.*

▶ **Corollary 11** (PaJAM Time Bounds Weights). *Let $\pi \triangleright \vdash t : \star$ be a type derivation and $n$ the length of the complete* PaJAM *run from the initial state $q_t^k$. Then $\mathbf{W}^{2k+1}_{\text{PaJAM}}(\pi) \leq n$.*

**Proof.** By the proposition above, we have that all the SPaJAM unreachable occurrences of $\star$ in $\pi$ have depth $> 2k+1$. This means that all the occurrences of $\star$ which have depth $\leq 2k+1$ are reachable. The result then follows by the bisimulation between the PaJAM and the SPaJAM (Prop. 5). ◀

The weight system is then capable of precisely measuring the runtime of the PaJAM.

▶ **Theorem 12** (PaJAM Time via Weighted Derivations). *Let $t$ be a closed term. The following are equivalent:*

1. *There exists a complete* PaJAM *run $\rho$ from the initial state $q_t^k$ such that $|\rho| = n$.*
2. *There exists a derivation $\pi$ such that $\pi \triangleright \vdash t : \star$ and $\mathbf{W}^{2k+1}_{\text{PaJAM}}(\pi) = n$.*

**Proof.** (2) ⇒ (1). We argue by using the bisimilarity between the PaJAM and the SPaJAM together with Prop. 8 and Corollary 11. (1) ⇒ (2). Since the PaJAM is correct for Closed CbN, as stated in Prop. 2, the existence of a finite PaJAM run implies that $t$ is has a normal form. This, by Prop. 13, implies the existence of a derivation $\pi$ such that $\pi \triangleright \vdash t : \star$. We are therefore once again in the setting of both Prop. 8 and Corollary 11. ◀

## 6 Complexity of the Parametric JAM

In this last section, we exploit our type theoretic analysis of the PaJAM to study its complexity. To achieve this, we establish a bound on the number of occurrences of the base type $\star$ up to a fixed depth within each type in a given derivation. Let $t$ be a closed term and $\pi$ be a derivation such that $\pi \triangleright \vdash t : \star$, also let $n$ be the number of $\beta$ steps needed to normalize $t$. We show that the number of occurrences of $\star$ in $\pi$ that appear at depth at most $k$ is polynomial in $n$, for any $k \in \mathbb{N}$. This result, in conjunction with Thm. 12, yields a bound on the length of the execution of the machine, proving that the complexity of the Parametric JAM is polynomial with respect to $n$ for any given initial parameter $k < \infty$.

**Bounding Arrows and Sequences.** We begin by defining a size on types capturing the maximum number of arrows "at the same depth". Intuitively, viewing a type $A$ as a tree in which internal nodes correspond to $\to$ and in which leaves correspond to occurrences of $\star$, we define the size $|A|_\to$ as the maximal length of a path from the root to any leaf.

$$|\star|_\to := 0 \qquad |S \to A|_\to := \max(|S|_\to, 1 + |A|_\to) \qquad |[A_1, \ldots, A_n]|_\to := \max_{i \leq n}(|A_i|_\to)$$

We define $\#_\to(\pi)$ as the maximum $|A|_\to$ for any type $A$ occurring in $\pi$ (on the right-hand side of $\vdash$). Since by $\beta$ expanding a term at most one occurrence of $\to$ can be created, we have the following bound.

▶ **Proposition 13** (Arrow Size Bound). *Let $t$ be a closed $\lambda$-term such that $t \to^n_{wh} \lambda x.u$. Then, there exists a unique $\pi \triangleright \vdash t : \star$ (modulo permutations). Moreover, $\#_\to(\pi) \leq n$.*

▶ Remark 14. By *permutation equivalence* we intend to equate type derivations which differ just by the ordering of elements inside sequences, and thus of premises in the rule T-@. It is clear that $\#_\to(\cdot)$ is invariant by permutation equivalence.

Then, we need to define a new measure on types, that computes the maximum size of sequences appearing in a type:

$$|\star|_{[]} := 0 \qquad |S \to A|_{[]} := \max(|S|_{[]}, |A|_{[]}) \qquad |[A_1, \dots, A_n]|_{[]} := \max(\max_{i \leq n} |A_i|_{[]}, n)$$

We define $\#_{[]}(\pi)$ as the maximum size $|A|_{[]}$ for any type $A$ occurring in $\pi$ (on the right-hand side of $\vdash$). We observe that that the maximum cardinality of sequences in a type derivation $\pi$ ending in $\star$ cannot be larger than the size of $\pi$ itself, noted $|\pi|$.

▶ **Proposition 15** (Sequence Size Bound). *Let $t$ be a closed term such that $t \to_{wh}^{*} \lambda x.u$ Then, there exists a unique $\pi \triangleright \vdash t : \star$ (modulo permutations). Moreover, $\#_{[]}(\pi) \leq |\pi|$.*

Given a term $t$ normalizing in $n$ steps, we are able to transfer the bound from being parametric on the size of its type derivation, to $n$. The overhead is indeed at most quadratic.

▶ **Proposition 16** (Derivation Size Bound [23, 5]). *Let $t$ be a closed term such that $t \to_{wh}^{n} \lambda x.u$. Then there exists a unique $\pi \triangleright \vdash t : \star$ (modulo permutations). Moreover, $|\pi| = \mathcal{O}(n^2)$.*

▶ Remark 17. Again, also here of course the size of a type derivation $|\cdot|$ is invariant by permutation equivalence.

**Final Bound.** We can now give a bound on the number of occurrences of $\star$ that appear inside at most $k$ nested sequences in a type $A$, i.e. the measure $\|A\|_k$. Each type $A$ has the shape $A = [A_1^1, \dots, A_{m_1}^1] \to \dots \to [A_1^n, \dots, A_{m_n}^n] \to \star$, for $n, m \geq 0$. By definition, we have that $n \leq |A|_{\to}$ and $m_i \leq |A|_{[]}$ for each $i \leq n$.[2] Hence, the number of types $A_j^i$ is at most $|A|_{\to} \cdot |A|_{[]}$. Notice that each $A_j^i$ has exactly one $\star$ at depth 0, which appears at depth 1 in $A$. Hence, we obtain the bound $\|A\|_1 \leq |A|_{\to} \cdot |A|_{[]} + 1$. By iterating this bound on each type $A_j^i$, we get the following lemma.

▶ **Lemma 18** (Type Size Bound). *For each type $A$ and each $k \geq 0$ we have that $\|A\|_{k+1} \leq (|A|_{\to} \cdot |A|_{[]})^{k+1} + \|A\|_k$.*

**Proof.** We start by noticing that for each type $B$ we have $\|B\|_0 = 1$. We now proceed by induction on $A$. If $A = \star$, we have $\|\star\|_1 = 1$. If $A = [A_1, \dots, A_n] \to B$, we have the following:

$$\begin{aligned}
\big\|[A_1, \dots, A_n] \to B\big\|_{k+1} &= \sum_{i \leq n} \|A_i\|_k + \|B\|_{k+1} \\
&\leq \sum_{i \leq n} ((|A_i|_{\to} \cdot |A_i|_{[]})^k + \|A_i\|_{k-1}) + (|B|_{\to} \cdot |B|_{[]})^{k+1} + \|B\|_k \\
&= \sum_{i \leq n} (|A_i|_{\to} \cdot |A_i|_{[]})^k + (|B|_{\to} \cdot |B|_{[]})^{k+1} + \sum_{i \leq n} \|A_i\|_{k-1} + \|B\|_k \\
&= \sum_{i \leq n} (|A_i|_{\to} \cdot |A_i|_{[]})^k + (|B|_{\to} \cdot |B|_{[]})^{k+1} + \big\|[A_1, \dots, A_n] \to B\big\|_k
\end{aligned}$$

Now, we just need to prove $\sum_{i \leq n} (|A_i|_{\to} \cdot |A_i|_{[]})^k + (|B|_{\to} \cdot |B|_{[]})^{k+1} \leq (|A|_{\to} \cdot |A|_{[]})^{k+1}$. By definition, we have that $|[A_1, \dots, A_n] \to B|_{\to} := \max_{i \leq n}(1 + |B|_{\to}, |A_i|_{\to})$, and thus $|B|_{\to} \leq |[A_1, \dots, A_n] \to B|_{\to} - 1 = |A|_{\to} - 1$. Then, we have:

[2] Notice that $n$ could in fact be bound by the maximal number of $\to$ occurring at the same sequence depth, whereas our size $|\cdot|_{\to}$ is larger. We use this definition because it leads to the same final bound and it is simpler to work with.

$$\sum_{i\leq n}(|A_i|_{\to}\cdot|A_i|_{[]})^k+(|B|_{\to}\cdot|B|_{[]})^{k+1}\leq\sum_{i\leq n}(|A|_{\to}\cdot|A|_{[]})^k+((|A|_{\to}-1)\cdot|A|_{[]})^{k+1}$$
$$\leq n(|A|_{\to}\cdot|A|_{[]})^k+((|A|_{\to}-1)\cdot|A|_{[]})^{k+1}\leq|A|_{[]}(|A|_{\to}\cdot|A|_{[]})^k+((|A|_{\to}-1)\cdot|A|_{[]})^{k+1}$$
$$=|A|_{[]}^{k+1}(|A|_{\to}^k+(|A|_{\to}-1)^{k+1})\leq|A|_{[]}^{k+1}\cdot|A|_{\to}^{k+1},$$

where the last step comes from the fact that $a^k+(a-1)^{k+1}\leq a^{k+1}$ when $k\geq 0$ and $a\geq 1$, as it is the case since $|A|_{\to}=|[A_1,\ldots,A_n]\to B|_{\to}\geq 1$. ◀

Then, we obtain a bound as a function of the number of normalization steps.

▶ **Proposition 19** (Concrete Type Bound). *Let $t$ be a closed term such that $t\to_{wh}^n\lambda x.u$ and $\pi\triangleright\vdash t:\star$. Then, for each type $A$ in $\pi$ and $k<\infty$ we have $\|A\|_k=\mathcal{O}(n^{3k})$.*

**Proof.** We proceed by induction on $k$. If $k=0$ then $\|A\|_0=1=n^0$. If $k>0$:

$$\|A\|_k\overset{\text{(L. 18)}}{\leq}(|A|_{\to}\cdot|A|_{[]})^k+\|A\|_{k-1}\overset{\text{(Prop. 13 + i.h.)}}{\leq}(n\cdot|A|_{[]})^k+\mathcal{O}(n^{3(k-1)})$$
$$\overset{\text{(Prop. 15 + Prop. 16)}}{\leq}(n\cdot\mathcal{O}(n^2))^k+\mathcal{O}(n^{3(k-1)})=\mathcal{O}(n^{3k}).$$ ◀

Then, we count the number of occurrences of $\star$ appearing at most at depth $k<\infty$ in the entire derivation $\pi$. This amounts to multiplying the bound established in the previous proposition by the size of $\pi$ (quadratic in the number of $\beta$-steps, see Prop. 16).

▶ **Theorem 20** (Bounding Weighted Derivations). *Let $t$ be a closed term such that $t\to_{wh}^n\lambda x.u$ and $\pi\triangleright\vdash t:\star$, then $\mathbf{W}^k_{\mathrm{PaJAM}}(\pi)=\mathcal{O}(n^{3k+2})$ for each $k<\infty$.*

**Proof.** The previous proposition gives us a bound for every type $A$ occurring in $\pi$. To obtain the total weight $\mathbf{W}^k_{\mathrm{PaJAM}}(\pi)$, we multiply this bound by the number of judgments in $\pi$, *i.e.* by $|\pi|=\mathcal{O}(n^2)$, by Prop. 16. Then $\mathbf{W}^k_{\mathrm{PaJAM}}(\pi)=\mathcal{O}(n^{3k})\cdot\mathcal{O}(n^2)=\mathcal{O}(n^{3k+2})$. ◀

Finally, we obtain a bound on the length of complete PaJAM runs.

▶ **Corollary 21** (The PaJAM Has Polynomial Overhead). *Let $t$ be a closed term such that $t\to_{wh}^n\lambda x.u$. Then the complete* PaJAM *run from $q_t^k$ has length $\mathcal{O}(n^{6k+5})$, for each $k<\infty$.*

**Proof.** Let $m$ be the length of the complete run. By Thm. 12, we have $\pi\triangleright\vdash t:\star$ such that $m\overset{\text{(Thm. 12)}}{=}\mathbf{W}^{2k+1}_{\mathrm{PaJAM}}(\pi)\overset{\text{(Thm. 20)}}{=}\mathcal{O}(n^{3(2k+1)+2})=\mathcal{O}(n^{6k+5})$. ◀

Now, the natural question is if this bound is tight. We believe it is not, as in the case of the JAM, *i.e.* when $k=0$, we already know from [7] that another bound is $\mathcal{O}(n^4\cdot|t|)$, likely not tight as well, instead of $\mathcal{O}(n^5)$. We leave for future work understanding the relationship between our complexity analysis and the one carried out in [7], which seem to be orthogonal. Intuitively, here we have analyzed type derivations "horizontally", judgment by judgment, while in [7] they have been analyzed "vertically", hence the dependence on the size of the term, which bounds derivation height.

## 7 Conclusion

In this paper, we introduced a new parametric machine, the PaJAM, which behaves like the IAM up to a fixed backtracking nesting depth, and then switches to behaving like the JAM. We showed that the runtime of this machine can be characterized using intersection type derivations, extending the results about the KAM and the IAM. Furthermore, thanks

to the aforementioned type theoretic framework, we were able to analyze its complexity. This allowed us to prove that the number of steps the PaJAM needs to evaluate a term $t$ is *polynomial* in the number of $\beta$-steps needed to normalize it, thus making the PaJAM a *reasonable* cost model [3].

Although, from a technical viewpoint, the analysis of the PaJAM just described is certainly of interest, the most significant conceptual contribution lies in the exploration of the space between the JAM and the IAM, the former being obtained as an optimization of the latter and thereby achieving, on certain terms, an exponential speedup. On the one hand, our work shows that the phase transition occurs "on the IAM side," only when jumps are never performed. On the other hand, it demonstrates that, from a quantitative perspective, the JAM is more similar to the KAM than to the machine of which it is an optimization.

In future work, we would like to investigate the relationship between the PaJAM and game semantics. Indeed, it would be interesting to analyze our results on non-idempotent intersection types from the perspective of game semantics, e.g. following the lines of [32, 17], and understand how the PaJAM relates with game-theoretical models, enriching the already well-known connections between the JAM and HO games (via the PAM-JAM isomorphism) and the IAM and AJM games.

Moreover, we are interested in the *space* performances of the PaJAM, in particular depending on the choice of the parameter. This analysis could be carried out either directly on the machine, as in [10, 11], or via intersection types, as in [8, 9]. The problem is non-trivial, as one should in this case take into account how data structures are represented at low-level, and how garbage collection is implemented.

## References


1 Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Inf. Comput.*, 163(2):409–470, 2000. doi:10.1006/inco.2000.2930.

2 Beniamino Accattoli. The complexity of abstract machines. In Horatiu Cirstea and Santiago Escobar, editors, *Proceedings Third International Workshop on Rewriting Techniques for Program Transformations and Evaluation, WPTE@FSCD 2016, Porto, Portugal, 23rd June 2016*, volume 235 of *EPTCS*, pages 1–15, 2016. doi:10.4204/EPTCS.235.1.

3 Beniamino Accattoli. (In)Efficiency and Reasonable Cost Models. In *12th Workshop on Logical and Semantic Frameworks, with Applications, LSFA 2017, Brasília, Brazil, September 23-24, 2017*, volume 338 of *Electronic Notes in Theoretical Computer Science*, pages 23–43. Elsevier, 2017. doi:10.1016/j.entcs.2018.10.003.

4 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. Distilling abstract machines. In Johan Jeuring and Manuel M. T. Chakravarty, editors, *Proceedings of the 19th ACM SIGPLAN international conference on Functional progranitarmming, Gothenburg, Sweden, September 1-3, 2014*, pages 363–376. ACM, 2014. doi:10.1145/2628136.2628154.

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

6 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. The machinery of interaction. In *Proceedings of the 22nd International Symposium on Principles and Practice of Declarative Programming*, PPDP '20, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3414080.3414108.

7 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. The (in)efficiency of interaction. *Proc. ACM Program. Lang.*, 5(POPL), January 2021. doi:10.1145/3434332.

8 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. The space of interaction. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2021. doi:10.1109/LICS52264.2021.9470726.

9 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. Multi types and reasonable space. *Proc. ACM Program. Lang.*, 6(ICFP):799–825, 2022. doi:10.1145/3547650.

10 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. Reasonable space for the $\lambda$-calculus, logarithmically. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 47:1–47:13. ACM, 2022. doi:10.1145/3531130.3533362.

11 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. Reasonable space for the $\lambda$-calculus, logarithmically. *Log. Methods Comput. Sci.*, 20(4), 2024. doi:10.46298/LMCS-20(4:15)2024.

12 Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds, fully developed. *J. Funct. Program.*, 30:e14, 2020. doi:10.1017/S095679682000012X.

13 Viviana Bono and Mariangiola Dezani-Ciancaglini. A tale of intersection types. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 7–20. ACM, 2020. doi:10.1145/3373718.3394733.

14 Antonio Bucciarelli and Thomas Ehrhard. On phase semantics and denotational semantics: the exponentials. *Ann. Pure Appl. Log.*, 109(3):205–241, 2001. doi:10.1016/S0168-0072(00)00056-7.

15 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Log. J. IGPL*, 25(4):431–464, 2017. doi:10.1093/JIGPAL/JZX018.

16 Stefano Catozi, Ugo Dal Lago, and Gabriele Vanoni. On jumps, interactions, and intersection types, 2026. arXiv:2606.27062.

17 Pierre Clairambault and Simon Forest. An analysis of symmetry in quantitative semantics. In Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza, editors, *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, pages 26:1–26:13. ACM, 2024. doi:10.1145/3661814.3662092.

18 Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the $\lambda$-calculus. *Notre Dame J. Formal Log.*, 21(4):685–693, 1980. doi:10.1305/NDJFL/1093883253.

19 Ugo Dal Lago and Gabriele Vanoni. (Definitely not) boring interaction abstract machines. *Theor. Comput. Sci.*, 1064:115683, 2026. doi:10.1016/J.TCS.2025.115683.

20 Vincent Danos, Hugo Herbelin, and Laurent Regnier. Game semantics & abstract machines. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 394–405. IEEE Computer Society, 1996. doi:10.1109/LICS.1996.561456.

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

22 Vincent Danos and Laurent Regnier. Head linear reduction, 2004.

23 Daniel de Carvalho. Execution time of $\lambda$-terms via denotational semantics and intersection types. *Math. Str. in Comput. Sci.*, 28(7):1169–1203, 2018. doi:10.1017/S0960129516000396.

24 Philippa Gardner. Discovering needed reductions using type theory. In *Theoretical Aspects of Computer Software (TACS '94)*, volume 789 of *Lecture Notes in Computer Science*, pages 555–574, 1994. doi:10.1007/3-540-57887-0_115.

25 Jean-Yves Girard. Geometry of Interaction 1: Interpretation of System F. In R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo, editors, *Studies in Logic and the Foundations of Mathematics*, volume 127, pages 221–260. Elsevier, 1989.

26 J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000. doi:10.1006/inco.2000.2917.

27 Assaf J. Kfoury. A linearization of the lambda-calculus and consequences. *Journal of Logic and Computation*, 10(3):411–436, 2000. doi:10.1093/logcom/10.3.411.

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

**29** P. J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, 1964. doi:10.1093/COMJNL/6.4.308.

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

**31** Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975. doi:10.1016/0304-3975(75)90017-1.

**32** Takeshi Tsukada, Kazuyuki Asada, and C.-H. Luke Ong. Generalised species of rigid resource terms. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005093.