# On jumps, interactions, and intersection types

Stefano Catozi[1], Ugo Dal Lago[2] and Gabriele Vanoni[3]

[1]*Université Sorbonne Paris Nord, Villetaneuse, France*
[2]*University of Bologna, Bologna, Italy*
[3]*IRIF, Université Paris Cité, Paris, 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 operational 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 $\beta$ steps, giving rise to a *reasonable* cost model, for each *finite* bound on the backtracking depth.

**Keywords**

lambda-calculus, geometry of interaction, intersection types, abstract machines

This extended abstract deals with two central concepts in the theory of the $\lambda$-calculus, namely *abstract machines* [1, 2, 3] on the one hand and *intersection types* [4, 5, 6] on the other. We are particularly interested in the relationship between these two concepts and our main contribution is a new tight correspondence result. This will allow us to analyze the efficiency of a family of machines via type-theoretic tools. In the following, we give some hints at the results we obtained, while the details can be found in [7].

**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 [2]. 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) [8], 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 [9, 10].

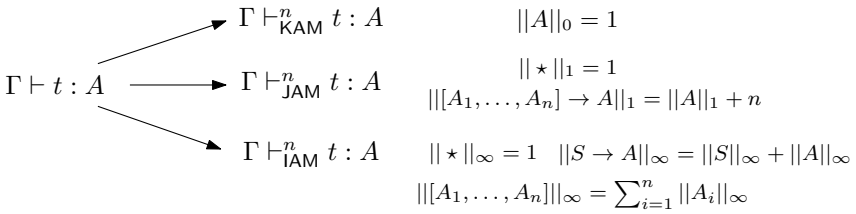
**Game machines.** Since the pioneering work of Danos, Herbelin and Regnier [11], it is well known that there are machines implementing weak-head reduction like the KAM, but which have an even lower-level behavior, closely related to game semantics and Girard's geometry of interaction (GoI) [12]. In particular, the so-called Interaction Abstract Machine (IAM) [13, 14, 15, 16] can be seen as an automata-theoretic counterpart of the GoI or of AJM games [17], and evaluates terms through a purely *local* and *reversible* process. There is also the so-called Pointer Abstract Machine (PAM) [11, 18], which is instead inspired, as its name says, by the mechanism of justification pointers from Hyland and Ong's games [19].

*ICTCS 2026: Italian Conference on Theoretical Computer Science, September 7–9, 2026, Udine, Italy*
catozi@lipn.univ-paris13.fr (S. Catozi); ugo.dallago@unibo.it (U. Dal Lago); gabriele.vanoni@irif.fr (G. Vanoni)
https://udallago.github.io (U. Dal Lago); https://vanoni.me (G. Vanoni)
0009-0007-0829-5789 (S. Catozi); 0000-0001-9200-070X (U. Dal Lago); 0000-0001-8762-8674 (G. Vanoni)

As discovered by Danos and Regnier [14], 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 [20]. 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 [21, 22], intersection types are known to be a syntactic counterpart of the *relational model* of linear logic [23]. Starting from de Carvalho's results [24], 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* [25, 26] 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 [20, 27]. 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 type rule 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. The key idea is to make the optimization mechanism underlying jumps depend on a parameter $k$, representing the maximum nesting depth of backtracking that the machine is allowed to perform before switching to a jump. In this way, the machine behaves like the IAM while the available backtracking budget has not been exhausted, and switches to the behavior of the JAM when the budget is over. This simple parameterization provides a uniform framework encompassing the two classical machines. Beyond unifying the two machines, this construction allows us to investigate the entire spectrum of intermediate evaluation strategies and to understand how the amount of allowed backtracking influences both the operational behavior of the machine and its computational complexity.
- We show that the number of steps performed by the PaJAM when evaluating a term $t$ can be measured, following the spirit of the figure above, by letting the size of a type depend on $k$ and counting occurrences of the base type $\star$ up to depth $k$. To establish this result, we first introduce a type-theoretic counterpart of the machine, called the Sequence PaJAM, and prove that it is strongly bisimilar to automata-theoretic machine. This correspondence allows machine executions to be interpreted as traversals of type derivations, where each reachable machine state corresponds to an occurrence of the ground type $\star$. The parameter $k$ naturally induces a bound on the depth of the reachable occurrences, so that changing the backtracking depth amounts to changing which parts of the derivation contribute to the execution. We then define a family of weighted type systems in which the weight of a derivation is precisely obtained by counting the occurrences of $\star$ that lie within the admissible depth. The resulting measure is shown to coincide exactly with the runtime of the PaJAM, thereby extending the quantitative correspondences previously known for the KAM, the JAM and the IAM to the whole parametric family.

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

- These results are then used to analyze the computational complexity of the parametric machine. Exploiting the exact correspondence between executions and weighted type derivations, we derive polynomial bounds on the size of the relevant portions of types by combining structural properties of non-idempotent intersection types with known bounds on the size of type derivations. This ultimately shows that every instance of the PaJAM with finite parameter $k < \infty$ simulates $\beta$-reduction with only a polynomial overhead, thus providing a reasonable cost model and proving time invariance. Interestingly, the analysis also highlights a sharp phase transition: all finite values of $k$ yield polynomial overheads, whereas the limiting case $k = \infty$, corresponding to the IAM, falls outside this regime and may exhibit exponentially longer executions. The framework therefore not only provides a unified analysis of the JAM and the IAM, but also explains precisely where their asymptotic difference originates.

**Wrapping up.** There are two take-home messages to this work:

1. 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;
2. the performances of the JAM (and thus of the PAM) and of the IAM can be studied within a *single* framework, which also gives a very simple account of their relative performance.

## Declaration on Generative AI

The authors have not employed any Generative AI tools.

## References


[1] P. J. Landin, The mechanical evaluation of expressions, The Computer Journal 6 (1964) 308–320.

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

[3] B. Accattoli, P. Barenbaum, D. Mazza, Distilling abstract machines, in: Proc. of ICFP 2014, ACM, 2014, pp. 363–376. doi:10.1145/2628136.2628154.

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

[5] A. Bucciarelli, D. Kesner, D. Ventura, Non-idempotent intersection types for the lambda-calculus, Log. J. IGPL 25 (2017) 431–464. doi:10.1093/JIGPAL/JZX018.

[6] V. Bono, M. Dezani-Ciancaglini, A tale of intersection types, in: Proc. of LICS 2020, ACM, 2020, pp. 7–20. doi:10.1145/3373718.3394733.

[7] S. Catozi, U. Dal Lago, G. Vanoni, On jumps, interactions, and intersection types, in: Proc. of MFCS 2026, 2026. doi:https://doi.org/10.4230/LIPIcs.MFCS.2026.59.

[8] J.-L. Krivine, A Call-by-name Lambda-calculus Machine, Higher Order Symbol. Comput. 20 (2007) 199–207. doi:10.1007/s10990-007-9018-9.

[9] B. Accattoli, P. Barenbaum, D. Mazza, Distilling abstract machines, in: Proc. of ICFP 2014, ACM, 2014, pp. 363–376. doi:10.1145/2628136.2628154.

[10] B. Accattoli, B. Barras, Environments and the complexity of abstract machines, in: Proc. of PPDP 2017, ACM, 2017, pp. 4–16. doi:10.1145/3131851.3131855.

[11] V. Danos, H. Herbelin, L. Regnier, Game semantics & abstract machines, in: Proc. of LICS 1996, IEEE Computer Society, 1996, pp. 394–405. doi:10.1109/LICS.1996.561456.

[12] J.-Y. Girard, Geometry of Interaction 1: Interpretation of System F, in: Studies in Logic and the Foundations of Mathematics, volume 127, 1989, pp. 221–260.

[13] I. Mackie, The Geometry of Interaction Machine, in: Proc. of POPL 1995, ACM Press, 1995, pp. 198–208. doi:10.1145/199448.199483.

[14] V. Danos, L. Regnier, Reversible, irreversible and optimal lambda-machines, Theoretical Computer Science 227 (1999) 79–97. doi:10.1016/S0304-3975(99)00049-3.

[15] B. Accattoli, U. Dal Lago, G. Vanoni, The machinery of interaction, in: Proc. of PPDP 2020, Association for Computing Machinery, 2020. doi:10.1145/3414080.3414108.

[16] U. Dal Lago, G. Vanoni, (Definitely not) boring interaction abstract machines, Theor. Comput. Sci. 1064 (2026) 115683. doi:10.1016/J.TCS.2025.115683.

[17] S. Abramsky, R. Jagadeesan, P. Malacaria, Full abstraction for PCF, Inf. Comput. 163 (2000) 409–470. doi:10.1006/inco.2000.2930.

[18] V. Danos, L. Regnier, Head Linear Reduction, Technical Report, 2004.

[19] J. M. E. Hyland, C. L. Ong, On full abstraction for PCF: i, ii, and III, Inf. Comput. 163 (2000) 285–408. doi:10.1006/inco.2000.2917.

[20] B. Accattoli, U. Dal Lago, G. Vanoni, The (in)efficiency of interaction, Proc. ACM Program. Lang. 5 (2021). doi:10.1145/3434332.

[21] P. Gardner, Discovering needed reductions using type theory, in: Proc. of TACS 1994, volume 789 of *LNCS*, 1994, pp. 555–574. doi:10.1007/3-540-57887-0_115.

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

[23] A. Bucciarelli, T. Ehrhard, On phase semantics and denotational semantics: the exponentials, Ann. Pure Appl. Log. 109 (2001) 205–241. doi:10.1016/S0168-0072(00)00056-7.

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

[25] B. Accattoli, S. Graham-Lengrand, D. Kesner, Tight typings and split bounds, fully developed, J. Funct. Program. 30 (2020) e14. doi:10.1017/S095679682000012X.

[26] B. Accattoli, G. Guerrieri, M. Leberle, Types by need, in: Proc. of ESOP 2019, volume 11423 of *LNCS*, 2019, pp. 410–439. doi:10.1007/978-3-030-17184-1_15.

[27] B. Accattoli, U. Dal Lago, G. Vanoni, The space of interaction, in: 36th Symposium on Logic in Computer Science (LICS), 2021, pp. 1–13. doi:10.1109/LICS52264.2021.9470726.