The Machinery of Interaction

Beniamino Accattoli
INRIA & École Polytechnique
beniamino.accattoli@inria.fr

Ugo Dal Lago
Università di Bologna & INRIA
ugo.dallago@unibo.it

Gabriele Vanoni
Università di Bologna & INRIA
gabriele.vanoni2@unibo.it

ABSTRACT

This paper revisits the Interaction Abstract Machine (IAM), a machine based on Girard's Geometry of Interaction, introduced by Mackie and Danos & Regnier. It is an unusual machine, not relying on environments, presented on linear logic proof nets, and whose soundness proof is convoluted and passes through various other formalisms. Here we provide a new direct proof of its correctness, based on a variant of Sands's improvements, a natural notion of bisimulation. Moreover, our proof is carried out on a new presentation of the IAM, defined as a machine acting directly on λ-terms, rather than on linear logic proof nets.

CCS CONCEPTS

• **Theory of computation** → **Lambda calculus**; **Abstract machines**.

KEYWORDS

geometry of interaction, lambda-calculus, abstract machines

ACM Reference Format:
Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. 2020. The Machinery of Interaction. In *22nd International Symposium on Principles and Practice of Declarative Programming (PPDP '20), September 8–10, 2020, Bologna, Italy.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3414080.3414108

ACKNOWLEDGDEMENTS

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).

1 INTRODUCTION

The advantage, and at the same time the drawback, of the λ-calculus is its distance from low-level implementation details. It comes with just one rule, β-reduction, and with no indication about how to implement it on low-level machines. It is an advantage when *reasoning* about programs expressed as λ-terms. It is a drawback, instead, when one wants to *implement* the λ-calculus, or to do complexity analyses, because β-steps are far from being atomic operations. In particular, terms can grow exponentially with the number of β-steps, a degeneracy known as *size explosion*, which is why β-reduction cannot be reasonably implemented, at least if one sticks to an explicit representation of λ-terms.

Environment Machines. Implementations solve this issue by evaluating the λ-calculus up to *sharing of sub-terms*, where sharing is realized through a data structure called *environment*, collecting the sharing annotations generated by the machine, one for each encountered β-redex. For common *weak* evaluation strategies (*i.e.* that do not inspect in scope of λ-abstractions) such as call-by-name/value/need, the number of β-steps is a reasonable time cost model [16, 22, 50]. Environment machines—whose most famous examples are Landin's SECD [39], Felleisen and Friedman's CEK [29], or Krivine's KAM [38]—can be optimized to run within a linear overhead with respect to the number of β-steps [10, 14]. Said differently, they *respect* the time cost model (see [3] for an overview). For space, the situation is different. Only recently the problem has been tackled [31] and some preliminary and limited results have appeared. Then, environment machines store information for every β-step, therefore using space linear in time, which is *the worst possible use of space*[1].

Beyond Environments. Frameworks based on the λ-calculus are invariably implemented using environments. Nonetheless, the lack of a fixed execution schema for the λ-calculus leaves open, in theory, the possibility of alternative implementation schemes. The theory of linear logic indeed provides a completely different style of abstract machines, rooted in Girard's Geometry of Interaction [33] (shortened to GoI in the following). These GoI machines were pioneered by Mackie and Danos & Regnier in the nineties [27, 41]. The basic idea is that the machine does not use environments, and instead uses a data structure called *token*, saving information about the history of the computation. The key point is that the token does not store information about every single β-redex, thus disentangling space-consumption from time-consumption. In other words, GoI machines are good candidates for space-efficient implementation schemes, as first shown by Schöpp and coauthors [23, 51]. The price to pay is that the machine wastes a lot of time to retrieve β-redexes, so that time is sacrificed for space. The same, however, happens with space-sensitive Turing machines.

The Interaction Abstract Machine. The original GoI machine is the *Interaction Abstract Machine* (IAM). It was developed at the same time by Mackie and Danos & Regnier, and its first appearance is in a paper by Mackie in 1995 [41], dealing with implementations. Danos and Regnier study it in two papers, one in 1996 together with Herbelin [25], where it is dealt with quickly, and its implementation theorem (or correctness[2]) is proved via game semantics, and one by themselves [27], published only in 1999 but reporting work dating back to a few years before, dedicated to the IAM and to an

[1]On sequential models space cannot exceed time, as one needs a unit of time to use a unit of space.
[2]The result that an abstract machine implements a strategy is sometimes called *correctness* of the machine. We prefer to avoid such a terminology, because it suggests the existence of a dual *completeness* result, that is never given because already contained in the statement of correctness. We then simply talk of an *implementation theorem.*

optimization based on a fine analysis of IAM runs. These papers differ on many details but they all formulate the IAM on linear logic proof-nets as a reversible, bideterministic automaton.

In [25], Danos, Herbelin, and Regnier prove that the IAM implements *linear head evaluation* $\rightarrow_{\mathsf{lh}}$ (shortened to LHE), a refinement of head evaluation, arising from the linear logic decomposition of the λ-calculus. Their proof of the implementation theorem for the IAM—the only one in the literature—is indirect and rooted in game semantics, as it follows from a sequence of results relating the IAM to AJM games, AJM games to HO games, HO games to another abstract machine, the PAM, and finally the PAM to LHE. Moreover, the proof is technical, the main ingredients unclear, and not as neat as for environment machines.

New Proof of the Implementation Theorem. The main contribution of this paper is an alternative proof of the implementation theorem for the IAM, which is independent of game semantics and other abstract machines. Our proof is direct and based on a natural notion of bisimulation, namely a variation on Sands' *improvements* [49].

The implementation theorem of GoI machines amounts to showing that their result is an adequate and sound semantics for LHE, that is, it is invariant by LHE (soundness) and it is non-empty if and only if LHE terminates (adequacy). The key point for soundness is that—in contrast to the study of environment machines—steps of the GoI machine are not mapped to LHE steps, because the GoI computes differently. What is shown is that if $t \rightarrow_{\mathsf{lh}} u$ then the run of the machine on t is "akin" to the run on u, and they produce akin results—see Sect. 2 for more details. In our proof, "akin" is interpreted as *bisimilar*. An *improvement* is a bisimulation asking that the run on u is no longer than the run on t. Building on such a quantitative refinement, we prove adequacy.

The proof of our implementation theorem is arguably conceptually simpler than Danos, Herbelin, and Regnier's. Of course, their deep connection with game semantics is an important contribution that is not present here. We believe, however, that having independent and simpler proof techniques is also valuable.

The Lambda Interaction Abstract Machine. The second contribution of the paper is a formulation of the IAM as a machine acting directly on λ-terms rather than on linear logic proof nets. Our proof might also have been carried out on proof nets, but we prefer switching to λ-terms for two reasons. First, manipulating terms rather than proof nets is easier and less error-prone for the technical development. Second, we aim at minimizing the background required for understanding the IAM, and so doing we remove any explicit reference to linear logic and graphical syntaxes.

The starting point of our *Lambda* Interaction Abstract Machine (λ-IAM) is seeing a position in the code t (what is usually the position of the token on the proof net representation of t) as a pair (u, C) of a sub-term u and a context C such that $C\langle u\rangle = t$. These positions are simply a readable presentation of pointers[3].

The main novelty of the new presentation is that some of the exponential transitions on proof nets are packed together in macro transitions. The shape of our transitions makes a sort of backtracking mechanism more evident. *Careful*: that the IAM rests on backtracking is the key point of Danos and Regnier in [27], and therefore it is not a novelty in itself. What is new is that such a mechanism is already visible at the level of transitions, while on proof nets it requires sophisticated analyses of runs.

It may be argued that linear logic provides a useful conceptual framework for the GoI. While this is undeniable, we are trying to show that linear logic is however not needed, and that an alternative presentation provides other useful intuitions—the two presentations give different insights, and thus complement each other. The easy correspondence between the two is stated in Sect. 10.

More About the λ-IAM. The original papers on the IAM [25, 27, 41] differ on many points. Here we follow [25], modeling the λ-IAM on the call-by-name translation of the λ-calculus in linear logic and considering only the path/runs starting on the distinguished conclusion corresponding to the output of the net/term. This is natural for terms, and also along the lines of how AJM games interpret terms. Similarly to AJM games, then, our GoI semantics is sound also for open terms with respect to erasing steps.

An original point of our work is the identification of a new invariant of the λ-IAM—probably of independent interest—based on what we call *exhaustible states*. Informally, a state of the λ-IAM is exhaustible if its token can be emptied in a certain way, somehow mimicking the computation which leads to the state itself.

This Paper in Perspective. This paper is part of a long-time endeavor by the authors directed at understanding complexity measures and implementation schemas for the λ-calculus. We provide a new proof technique for GoI implementation theorems not relying on game semantics, together with an new presentation of the original machine by Mackie and Danos & Regnier not relying on linear logic. The aim is to set the ground for a formal, robust, and systematic study of GoI machines and their complexity, while at the same time shrinking to the minimum the required background. A further motivation behind our work is the desire to make the study of GoI machines easier to formalize in proof assistants, as proof nets are particularly cumbersome in that respect.

Related Work on GoI. This is certainly *not* the first paper on the GoI and the λ-calculus. Indeed, the literature on the topic and its applications is huge, and goes from Girard's original papers [33], to Abramsky *et al*'s reformulation using the INT-construction [1], Danos and Regnier's using path algebras [26], Ghica's applications to circuit synthesis [32], together with extensions by Hoshino, Muroya, and Hasuo to languages with various kinds of effects [35], and Laurent's extension to the additive connectives of linear logic [40]. In all these cases, the GoI interpretation, even when given on λ-terms, goes *through* linear logic (or symmetric monoidal categories) in an essential way. The only notable exceptions are perhaps the recent contributions by Schöpp on the relations between GoI, CPS, and defunctionalization [52, 53] in which, indeed, some deep relations are shown to exist between GoI and classic tools in the theory of λ-calculus. Even there, however, GoI is seen as obtained through the INT-construction [1, 36], although applied to a syntactic category of terms.

The GoI has been studied in relationship with implementations of functional languages, by Gonthier, Abadi and Levy, who studied

[3]For the acquainted reader, they play a role akin to the initial labels in Lévy's labeled λ-calculus, itself having deep connections with the IAM [15].

Lévy's optimal evaluation [34], and by Mackie with his GoI machine for PCF [41], Gödel System T [42], and—with Fernandez—for call-by-value [30]. The space-efficiency studied by Dal Lago and Schöpp [23] is exploited by Mazza in [44] and, together with Terui, in [45]. Dal Lago and coauthors have also introduced variants of the IAM acting on proof nets for a number of extensions of the λ-calculus [19–21, 24]. Curien and Herbelin study abstract machines related to game semantics and the IAM in [17, 18]. Muroya and Ghica have recently studied the GoI in combination with rewriting and abstract machines in [47]. The already cited works by Schöpp [52, 53] highlight how GoI can be seen as an optimized form of CPS transformation, followed by defunctionalization.

Related Work on Environment Machines. The *time* efficiency of environment machines has been recently closely scrutinized. Before 2014, the topic had been mostly neglected—the only two counterexamples being Blelloch and Greiner in 1995 [16] and Sands, Gustavsson, and Moran in 2002 [50]. Since 2014—motivated by advances by Accattoli and Dal Lago on time cost models for the λ-calculus [8]—Accattoli and co-authors have explored time analyses of environment machines from different angles [5–7, 10].

Proofs. Omitted proofs can be found in [13].

2 IMPLEMENTATIONS AND IMPROVEMENTS

In this section we explain in which sense the λ-IAM implements head evaluation. The λ-IAM itself shall be defined in the next sections, here we rather provide a high-level perspective on the differences between the λ-IAM and environment machines. We shall also define *improvements*, the notion of bisimulation that we use to prove the implementation theorems.

(Restricted) Head Evaluation. The λ-IAM implements *head evaluation*, the simple reduction relation defined as:

$$\lambda x_1.\dots\lambda x_i.(\lambda y.t)ur_1\dots r_j \quad \rightarrow_{\mathsf{h}} \quad \lambda x_1.\dots\lambda x_i.t\{y{\leftarrow}u\}r_1\dots r_j.$$

where $i, j \geq 0$. Note that $\rightarrow_{\mathsf{h}}$ reduces under abstractions, but only head abstractions, and that terms may be open. For a fixed $i \in \mathbb{N}$, the restriction $\rightarrow_{\mathsf{h}_i}$ of $\rightarrow_{\mathsf{h}}$ is obtained by forbidding $\rightarrow_{\mathsf{h}}$ to reduce under more than i head abstractions.

Machines. We need some basic terminology about machines. A *machine* $\mathrm{M} = (s, \rightarrow)$ is a transition system $\rightarrow$ over a set of states, noted s. A potentially empty sequence of transitions is a *run*. A state is usually given by a code, that is a λ-term, and a set of data structures. Among the data structures of environment machines there are always *environments*, that record the previously encountered β-redexes. The λ-IAM instead does not have any environment, because it uses a different mechanism to compute. Indeed, there are (at least) three key differences between environment machines and the λ-IAM, that we now overview.

Difference 1: Initial States and Depth. In the initial state s_t of code t for an environment machine, all data structures are empty. Environment machines, additionally, are either weak (that is, *never* enter abstractions) or strong (they enter into *all* abstractions). In contrast, the λ-IAM is *incrementally strong*, *i.e.*, it has a finer mechanism for entering into *some* abstractions. The number of head abstractions that a run of the λ-IAM can cross, called the *depth* of the run, is specified by initializing the token with a natural number i. Note the difference with environment machines: once the code t is fixed, such machines have only one initial state s_t, while the λ-IAM has a *family* of initial states $\{s_{t,i}\}_{i\in\mathbb{N}}$, one for each depth i.

Therefore, the λ-IAM *approximates* head reduction: a run from $s_{t,i}$ essentially correspond to evaluation with respect to the restricted i-head evaluation $\rightarrow_{\mathsf{h}_i}$, and one approximates head reduction by executing the λ-IAM on increasing values of i.

Difference 2: Evaluating Without Simulating the Rewriting. The way in which the λ-IAM implements $\rightarrow_{\mathsf{h}}$ is fundamentally different from the way in which environment machines implement strategies. Roughly, environment machines simulate the strategy, while the λ-IAM computes/approximates a semantics for it, as we now explain.

Implementation Theorem for Environment Machines. An environment abstract machine M implements a strategy $\rightarrow$ if from the initial state s_t of code t it computes a representation of the normal form $\mathsf{nf}_{\rightarrow}(t)$. In particular, the machine somehow maintains the representation of how the strategy $\rightarrow$ modifies the term t they both evaluate. The implementation theorem states a weak bisimulation between the transitions $s \rightarrow_{\mathrm{M}} s'$ of the machine and the steps $t \rightarrow u$ of the strategy. In particular, a run ρ_t of the machine on t passes through some states representing u.

Implementation Theorem(s) for the λ-IAM.. The λ-IAM, and more generally GoI machines, do implement strategies, but in a different way. The key point is that these machines do not trace how the strategy modifies the term. If $t \rightarrow_{\mathsf{h}} u$, a λ-IAM run of code t never passes through a representation of u, as *implementing* $\rightarrow_{\mathsf{h}}$ here denotes something else. There are actually *two* implementation theorems, called *soundness* and *adequacy*. Soundness is the fact that, at a fixed depth, the run of code t is bisimilar to the run of code u. Note the difference with environment machines: there, the bisimulation is between *steps* on terms and transitions on states. For the λ-IAM, it is between *transitions* on states (of code t) and transitions on states (of code u). The important consequence of soundness is that the runs from t and u produce bisimilar final states (if they terminate). The idea is that these final states are different but encode the same description of the normal form of t and u, which is the semantics $[\![t]\!]$ induced by the λ-IAM.

Adequacy guarantees that the interpretation $[\![t]\!]$ reflects some observable aspects of t. For a head evaluation, one usually observes termination, and, if it holds, the identity of the head variable. This is exactly what $[\![t]\!]$ reflects, or it is adequate for.

To be precise, the interpretation $[\![t]\!]$ is parametric in the depth i, thus $[\![t]\!]_i$ is a semantics for $\rightarrow_{\mathsf{h}_i}$, amounting to specifying the 4 possible outcomes of evaluating t with $\rightarrow_{\mathsf{h}_i}$: divergence, termination on a head normal form whose head variable is free or bound (these are two different outcomes), or termination but not on a head normal form (that is, on a term of the form $\lambda x_1.\dots\lambda x_i.\lambda y.t$, indicating that one has to consider the more informative approximation $[\![t]\!]_{i+1}$, inspecting under λy). Soundness then is the fact that if $t \rightarrow_{\mathsf{h}} u$ then $[\![t]\!]_i = [\![u]\!]_i$ for all i, while adequacy is the fact that there is i such that $[\![t]\!]_i$ finds a head variable if and only if $\rightarrow_{\mathsf{h}}$ terminates.

Difference 3: On Not Computing Results. Environment machines implementing a strategy $\rightarrow$ stop on final states s_f decoding to $\rightarrow$-normal forms. Another difference is that the λ-IAM does not compute a code representation of the result $\mathsf{hnf}(t)$. It only recovers the micro information $[\![t]\!]_i$ about $\mathsf{hnf}(t)$ described above, by exploring the immutable code t. This is in accordance with other models: space-sensitive Turing machines do not compute the whole output but only single bits of it. The λ-IAM computes the spine of $\mathsf{hnf}(t)$, that is, its sequence of head constructors, via potentially many runs. Indeed, if $\mathsf{hnf}(t) = \lambda x_1 \ldots . \lambda x_i .(y u_1 \ldots u_j)$ then one has to execute the λ-IAM $i+1$ times, computing $[\![t]\!]_k$ for $0 \le k \le i$. For $k \in \{0, \ldots, i-1\}$, $[\![t]\!]_k$ says that the λ-IAM terminates but not on a head normal form, thus providing the information that $\mathsf{hnf}(t)$ has i spine abstractions. The additional run provides the identity of the head variable (together with the information of whether it is bound or not) plus the number j of arguments.

Terminating without Ever Succeeding. The way in which the λ-IAM reflects $\to_{\mathsf{h}}$ divergence is subtle. It is possible that $\to_{\mathsf{h}}$ diverges on t and all the runs of the λ-IAM terminate on t without ever finding a head variable. The idea is that the λ-IAM *approximates* the head evaluation of t, computing $\to_{\mathsf{h}_i}$, which corresponds to incrementally build the spine branch of the Lévy-Longo tree of t. For instance, on a non-terminating term such as $\Lambda = (\lambda x.\lambda y.xx)(\lambda x.\lambda y.xx)$ the λ-IAM terminates at all depths. Note in fact that $\Lambda \to_{\mathsf{h}} \lambda x.\Lambda$, *i.e.* Λ has an infinite number of head abstractions in its limit normal form. So $\to_{\mathsf{h}_i}$ terminates on Λ for every k, and, accordingly, so does the λ-IAM run at depth i (the value of $[\![\Lambda]\!]_i$ is termination but not on a head normal form). On the contrary, Ω has no head abstractions in its limit normal form and thus both $\to_{\mathsf{h}_i}$ and the λ-IAM diverge on Ω at every depth i.

2.1 Improvements, Abstractly

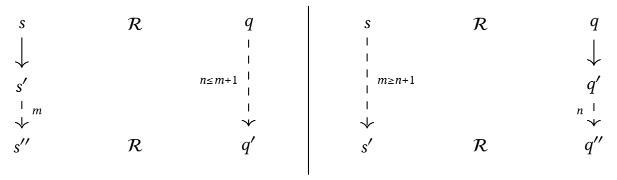
To prove the soundness and adequacy theorems we shall use *improvements*, a refinement of the classical notion of bisimulation inspired by Sands [49], and explained here. The concrete improvements at work in the proofs of the theorems are defined in Sect. 8.

An improvement is a weak bisimulation between two transition systems preserving termination and guaranteeing that, whenever s and q are related and terminating, then q terminates in no more steps than s—the *no-more-steps* part implies that the definition is asymmetric in the way it treats the two transition systems, and it shall play a key role in the proof of adequacy.

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

Definition 2.1 (Improvements). Given two DTS $\mathcal{S}$ and $\mathcal{Q}$, a relation $\mathcal{R} \subseteq S \times Q$ is an *improvement* if given $(s, q) \in \mathcal{R}$ the following conditions hold (for 2 and 3 see Fig. 1).

(1) *Final state right*: if $q \in \mathcal{F}_{\mathcal{Q}}$, then $s \to^n s'$, for some $s' \in \mathcal{F}_{\mathcal{S}}$ and $n \ge 0$.
(2) *Transition left*: if $s \to s'$, then there exists s'', q', n, m such that $s' \to^m s''$, $q \to^n q'$, $s'' \mathcal{R} q'$ and $n \le m+1$.



$$\begin{tikzcd}
s \arrow[d] & \mathcal{R} & q \arrow[dd, dashed, "n\le m+1"'] \\
s' \arrow[d, dashed, "m"] & & \\
s'' & \mathcal{R} & q'
\end{tikzcd}
\qquad
\begin{tikzcd}
s \arrow[dd, dashed, "m\ge n+1"] & \mathcal{R} & q \arrow[d] \\
& & q' \arrow[d, dashed, "n"'] \\
s' & \mathcal{R} & q''
\end{tikzcd}$$

Figure 1: Diagrammatic definition of improvements.

(3) *Transition right*: if $q \to q'$, then there exists s', q'', n, m such that $s \to^m s'$, $q' \to^n q''$, $s' \mathcal{R} q''$ and $m \ge n+1$.

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

Proposition 2.2. *Let $\mathcal{R}$ be an improvement on two DTS $\mathcal{S}$ and $\mathcal{Q}$, and $s\mathcal{R}q$.*

(1) Termination equivalence*: $s \in \mathcal{S}_\downarrow$ if and only if $q \in \mathcal{Q}_\downarrow$.*
(2) Improvement*: $|s| \ge |q|$.*

3 A GENTLE INTRODUCTION TO THE λ-IAM

This section introduces the λ-IAM gradually. There are various mechanisms at work in the λ-IAM. Most of them are also part of the simpler machine that evaluates linear λ-terms (where each variable occurs exactly once). Then we first see the Linear λ-IAM, which is easier to grasp, and later refine it with non-linearity.

Defining the Linear λ-IAM. An essential point is that the initial code t of the machine never changes. The λ-IAM only moves over it, in a local way, with no rewriting of the code and without ever substituting terms for variables. The current position in the code t is represented as a pair (u, C) where C is a context (that is, a term with a hole) and $C\langle u\rangle = t$.

A state s of the Linear λ-IAM has the shape (t, C, T, d) where (t, C) is a position while T and d are the linear token (which is a stack) and the direction, defined by:

$$\begin{array}{rrcl}
\textsc{(Linear) token} & T & ::= & \epsilon \mid @\cdot T \mid \lambda\cdot T \mid x\cdot T \mid \mathsf{b}\cdot T \\
\textsc{Direction} & d & ::= & \color{red}{\downarrow} \mid \color{blue}{\uparrow}
\end{array}$$

The token T can contain occurrences of @ and λ, variables, and occurrences of b. The use of these symbols is explained below via examples. Directions $\color{red}{\downarrow}$ and $\color{blue}{\uparrow}$, pronounced *down* and *up*, shall be represented mostly via colors and underlining: the code term in red and underlined, for $\color{red}{\downarrow}$, and the code context in blue and underlined, for $\color{blue}{\uparrow}$. This way, the fourth component of states is often omitted.

The transitions of the Linear λ-IAM are in Fig. 2. Roughly, when the direction is $\color{red}{\downarrow}$, the machine looks for the head variable of t. The direction changes to $\color{blue}{\uparrow}$ when the head variable x is found, moving also the machine to the position $(\lambda x.u, D)$ of the binder of x. In direction $\color{blue}{\uparrow}$, the λ-IAM explores D looking for the argument that head evaluation would substitute on x. *Initial states* have the form $s_{t,k} := (\underline{t}, \langle\cdot\rangle, @^k)$, where t is a term and $k \ge 0$ is the *depth.*

Mechanism 1: Search Up to β-Redexes. A basic mechanism of the Linear λ-IAM is the search of the head variable in direction $\color{red}{\downarrow}$ without recording β-redexes, via transitions $\to_{@1}$ and $\to_{@2}$. Consider the following run at depth 0, with $\mathsf{I}_z := \lambda z.z$ and $\mathsf{I}_w := \lambda w.w$.

Sub-tm	Context	Tok.		Sub-tm	Context	Tok.
$\underline{tu}$	C	T	$\to_{@1}$	$\underline{t}$	$C\langle\langle\cdot\rangle u\rangle$	$@\cdot T$
$\underline{\lambda x.t}$	C	$@\cdot T$	$\to_{@2}$	$\underline{t}$	$C\langle\lambda x.\langle\cdot\rangle\rangle$	T
$\underline{x}$	$C\langle\lambda x.D\rangle$	T	$\to_{\mathsf{var}}$	$\lambda x.D\langle x\rangle$	$\underline{C}$	$x\cdot T$
$\underline{\lambda x.D\langle x\rangle}$	C	$\mathsf{b}\cdot T$	$\to_{\mathsf{bt2}}$	x	$\underline{C\langle\lambda x.D\rangle}$	T

Sub-tm	Context	Tok.		Sub-tm	Context	Tok.
t	$\underline{C\langle\langle\cdot\rangle u\rangle}$	$\lambda\cdot T$	$\to_{\lambda 2}$	tu	$\underline{C}$	T
t	$\underline{C\langle\lambda x.\langle\cdot\rangle\rangle}$	T	$\to_{\lambda 1}$	$\lambda x.t$	$\underline{C}$	$\lambda\cdot T$
t	$\underline{C\langle\langle\cdot\rangle u\rangle}$	$x\cdot T$	$\to_{\mathsf{arg}}$	$\underline{u}$	$C\langle t\langle\cdot\rangle\rangle$	T
t	$\underline{C\langle u\langle\cdot\rangle\rangle}$	T	$\to_{\mathsf{bt1}}$	$\underline{u}$	$C\langle\langle\cdot\rangle t\rangle$	$\mathsf{b}\cdot T$

Figure 2: Linear λ-IAM transitions.

Sub-tm	Context	Tok.	
$\underline{((\lambda y.\lambda x.xy)\mathsf{I}_z)\mathsf{I}_w}$	$\langle\cdot\rangle$	ϵ	$\to_{@1}$
$\underline{(\lambda y.\lambda x.xy)\mathsf{I}_z}$	$\langle\cdot\rangle\mathsf{I}_w$	$@$	$\to_{@1}$
$\underline{\lambda y.\lambda x.xy}$	$(\langle\cdot\rangle\mathsf{I}_z)\mathsf{I}_w$	$@\cdot @$	$\to_{@2}$
$\underline{\lambda x.xy}$	$((\lambda y.\langle\cdot\rangle)\mathsf{I}_z)\mathsf{I}_w$	$@$	$\to_{@2}$
$\underline{xy}$	$((\lambda y.\lambda x.\langle\cdot\rangle)\mathsf{I}_z)\mathsf{I}_w$	ϵ	$\to_{@1}$
$\underline{x}$	$((\lambda y.\lambda x.\langle\cdot\rangle y)\mathsf{I}_z)\mathsf{I}_w$	$@$	

When the machine faces an application, transition $\to_{@1}$ records on the token the presence of an argument by pushing the symbol @, but not the argument itself (as it would instead do an environment machine), and moves to the left sub-term. Dually, on an abstraction, if the top of the token is @ then the machine pops it and moves to the body of the abstraction. This way, β-redexes are simply skipped. Note indeed that after the first 4 transitions the machine has crossed 2 β-redexes and the token is empty, as at the beginning.

Mechanism 2: Finding Variables and Arguments. In the example, the Linear λ-IAM finds the head variable x. Then transition $\to_{\mathsf{var}}$ applies, changing the direction to $\uparrow$ and moving the position to the binder λx. The machine now looks for the argument of the binder, exploring the context (now underlined and in blue) rather than the sub-term of the current position.

$\underline{x}$	$((\lambda y.\lambda x.\langle\cdot\rangle y)\mathsf{I}_z)\mathsf{I}_w$	$@$	$\to_{\mathsf{var}}$
$\lambda x.xy$	$\underline{((\lambda y.\langle\cdot\rangle)\mathsf{I}_z)\mathsf{I}_w}$	$x\cdot @$	$\to_{\lambda 1}$
$\lambda y.\lambda x.xy$	$\underline{(\langle\cdot\rangle\mathsf{I}_z)\mathsf{I}_w}$	$\lambda\cdot x\cdot @$	$\to_{\lambda 2}$
$(\lambda y.\lambda x.xy)\mathsf{I}_z$	$\underline{\langle\cdot\rangle\mathsf{I}_w}$	$x\cdot @$	$\to_{\mathsf{arg}}$
$\underline{\mathsf{I}_w}$	$((\lambda y.\lambda x.xy)\mathsf{I}_z)\langle\cdot\rangle$	$@$	

Note that $\to_{\mathsf{var}}$ adds x to the token, recording that the variable x has been found. The search for the argument is, again, *up to β-redexes*, via transitions $\to_{\lambda 1}$ and $\to_{\lambda 2}$, adding a symbol λ to the token on abstractions and removing it on applications. The $\uparrow$ phase ends with transition $\to_{\mathsf{arg}}$, that fires when the hole of the context is facing an argument and the token contains a variable, here x—such an argument matches the binder λx of the previously found variable. The transition removes x from the token, moves to the found argument, and switches direction.

Mechanism 3: Backtracking. On the example, the Linear λ-IAM continues by looking for the head variable of I_w, as expected.

$\underline{\lambda w.w}$	$((\lambda y.\lambda x.xy)\mathsf{I}_z)\langle\cdot\rangle$	$@$	$\to_{@1}$
$\underline{w}$	$((\lambda y.\lambda x.xy)\mathsf{I}_z)(\lambda w.\langle\cdot\rangle)$	ϵ	$\to_{\mathsf{var}}$
$\lambda w.w$	$\underline{((\lambda y.\lambda x.xy)\mathsf{I}_z)\langle\cdot\rangle}$	w	

Now something different happens. The machine is looking for the argument of I_w. Such an argument is not readily available, as the hole has no arguments in the current context, because $\langle\cdot\rangle$ is the right sub-term of an application. Since I_w is a replacement for x, its argument is actually the argument y of x. Then the machine *backtracks* to x. Backtracking is started by transition $\to_{\mathsf{bt1}}$, which adds b to the token, moves to the left sub-term of the application, and changes direction. The next 3 steps look for λx up to β-redexes:

I_w	$\underline{((\lambda y.\lambda x.xy)\mathsf{I}_z)\langle\cdot\rangle}$	w	$\to_{\mathsf{bt1}}$
$\underline{(\lambda y.\lambda x.xy)\mathsf{I}_z}$	$\langle\cdot\rangle\mathsf{I}_w$	$\mathsf{b}\cdot w$	$\to_{@1}$
$\underline{(\lambda y.\lambda x.xy)}$	$(\langle\cdot\rangle\mathsf{I}_z)\mathsf{I}_w$	$@\cdot\mathsf{b}\cdot w$	$\to_{@2}$
$\underline{\lambda x.xy}$	$((\lambda y.\langle\cdot\rangle)\mathsf{I}_z)\mathsf{I}_w$	$\mathsf{b}\cdot w$	$\to_{\mathsf{bt2}}$
x	$\underline{((\lambda y.\lambda x.\langle\cdot\rangle y)\mathsf{I}_z)\mathsf{I}_w}$	w	$\to_{\mathsf{arg}}$
$\underline{y}$	$((\lambda y.\lambda x.x\langle\cdot\rangle)\mathsf{I}_z)\mathsf{I}_w$	ϵ	

When λx is found, transition $\to_{\mathsf{bt2}}$ ends the backtracking, going back to the unique occurrence of x in the body of the abstraction (here linearity is crucial), and restores the search for arguments, changing direction and removing b. The next step is given by transition $\to_{\mathsf{arg}}$, that finally finds the argument y of I_w, removing w from the token.

Mechanism 4: Depth and Result. In two more transitions, the Linear λ-IAM reaches a final state.

$\underline{y}$	$((\lambda y.\lambda x.x\langle\cdot\rangle)\mathsf{I}_z)\mathsf{I}_w$	ϵ	$\to_{\mathsf{var}}$
$(\lambda y.\lambda x.xy)$	$\underline{(\langle\cdot\rangle\mathsf{I}_z)\mathsf{I}_w}$	y	$\to_{\mathsf{arg}}$
$\underline{\mathsf{I}_z}$	$((\lambda y.\lambda x.xy)\langle\cdot\rangle)\mathsf{I}_w$	ϵ	

Note that the machine terminates without finding the head variable z of the head normal form I_z of $t := ((\lambda y.\lambda x.xy)\mathsf{I}_z)\mathsf{I}_w$. This is because we executed the Linear λ-IAM at depth 0, while the head variable is at depth 1 in I_z. If we start over adding @ to the initial token—corresponding to execute the Linear λ-IAM at depth 1—the run would go exactly as before but for the fact that the token of every state has an additional @ at the end. The last state of the example would then no longer be final (since it has @ on the token). The run at depth 1 has the following shape.

$\underline{((\lambda y.\lambda x.xy)\mathsf{I}_z)\mathsf{I}_w}$	$\langle\cdot\rangle$	$@$	$\to_{@1}$
$\dots$	$\dots$	$\dots$	$\to^{*}\to_{\mathsf{arg}}$
$\underline{\lambda z.z}$	$((\lambda y.\lambda x.xy)\langle\cdot\rangle)\mathsf{I}_w$	$@$	$\to_{@2}$
$\underline{z}$	$((\lambda y.\lambda x.xy)(\lambda z.\langle\cdot\rangle))\mathsf{I}_w$	ϵ	$\to_{\mathsf{var}}$
$\dots$	$\dots$	$\dots$	$\to^{*}$
$((\lambda y.\lambda x.xy)\mathsf{I}_z)\mathsf{I}_w$	$\underline{\langle\cdot\rangle}$	z	

After finding z, the machine does a sequence of transitions ending on the initial position with direction $\uparrow$ and z on the token. The result of the run–that is, the identity of the head variable of the head normal form—is then available, on the token.

Two Observations: @ and λ Can Be Merged, and Locality. Note that the symbols @ and λ of the token play dual roles and never interact. In the following sections, we then harmlessly merge them into a single symbol $\bullet$, pronounced *dot*.

Transitions $\to_{\mathsf{var}}$ and $\to_{\mathsf{bt2}}$ move from a variable to its binder and vice versa. These moves are local if one assumes that λ-terms are represented by implementing variable occurrences as pointers to their binders, as in the proof net representation of λ-terms, see Sect. 10 for a precise comparison.

Two Issues with Non-Linearity. Generalizing the Linear λ-IAM to arbitrary, potentially non-linear λ-terms needs to address two difficulties. The first one is that, when the machine faces a sub-term $\lambda x.t$ and needs to end backtracking via transition $\rightarrow_{\mathrm{bt2}}$, it is no longer clear to which occurrence of x in t one should move, if there are many, or what to do, if x does not occur in t. The second issue is related to duplications. Consider the term $t := (\lambda x.x(xu))(\lambda y.r)$. The head evaluation of t makes two copies of $\lambda y.r$ that are used differently, because one is applied to u and the other one to $(\lambda y.r)u$. The machine does not duplicate sub-terms but still it has to somehow distinguish different uses of a sub-term.

Towards the λ-IAM. The two issues are solved via three correlated modifications of the machine. First, the symbols x and b for the token are generalized to variable positions $(x, \lambda x.C)$ inside the scope of their binder. Replacing b with $(x, \lambda x.C)$ in particular removes the non-determinism on $\rightarrow_{\mathrm{bt2}}$, when x has many occurrences[4]. Second, the token is split in two components, called *log* and *tape*. Roughly, the tape is the token of the Linear λ-IAM (generalized to variable positions) while the log stores, after transition $\rightarrow_{\mathrm{arg}}$, the variable position for which the machine found the argument, so as to be able to know to which occurrence to backtrack to in transition $\rightarrow_{\mathrm{bt1}}$. Third, there is a mechanism for distinguishing different uses of sub-terms. The log actually stores more than one variable position, and every position comes with its own log, acting as a sort of identifier for the use of that position. The next section formally develops this subtle mechanism. Logs and the way they distinguish uses of sub-terms without duplicating are far from being intuitive: they are both the mystery and the magic of the geometry of interaction.

4 THE λ-IAM

In this section we introduce the data structures used by the λ-IAM and its transition rules.

Terms and Leveled Contexts. Let $\mathcal{V}$ be a countable set of variables. Terms of the *λ-calculus* are defined as follows.

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

Free and *bound variables* are defined as usual: $\lambda x.t$ binds x in t.

Terms are considered modulo α-equivalence, and $t\{x{\leftarrow}u\}$ denotes capture-avoiding (meta-level) substitution of all the free occurrences of x for u in t.

The study of the λ-IAM requires *contexts*, that are terms with a single occurrence of a special constant $\langle\cdot\rangle$, called *the hole*, that is a place-holder for a removed sub-term. In fact, we need a notion of context more informative than the usual one, introduced next.

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

The index n in C_n counts the number of arguments into which the hole $\langle\cdot\rangle$ is contained in C_n[5]. Contexts of level 0 are also called *head contexts* and are denoted by H, K, G. The level of a context shall be omitted when not relevant to the discussion—note that any ordinary context can be written *in a unique way* as a leveled context, so that the omission is anyway harmless.

The *plugging* $C_n\langle t\rangle$ of a term t in C_n is defined by replacing the hole $\langle\cdot\rangle$ with t, potentially capturing free variables of t. Plugging $C_n\langle C_m\rangle$ of a context for a context is defined similarly. A *position* (of level n) in a term u is a pair (t, C_n) such that $C_n\langle t\rangle = u$.

Logs and Logged Positions. The λ-IAM relies on two mutually recursive notions, namely *logged positions* and *logs*: a logged position is a position (t, C_n) together with a log[6] L_n, that is a list of logged positions, having length n.

LOGGED POSITIONS	LOGS
$l ::= (t, C_n, L_n)$	$L_0 ::= \epsilon \qquad L_{n+1} ::= l \cdot L_n$

We use $\cdot$ also to concatenate logs, writing, *e.g.*, $L_n \cdot L$, using L for a log of unspecified length. Intuitively, logs contain some minimal information for backtracking to the associated position.

Tape, Token, Direction, State. The *tape* T is a finite sequence of elements of two kinds, namely logged positions, and occurrences of the special symbol $\bullet$, needed to cross abstractions and applications.

$$\text{TAPES} \qquad T \quad ::= \quad \epsilon \;\mid\; \bullet \cdot T \;\mid\; l \cdot T.$$

A *token* is a log plus a tape.

Definition 4.1 (λ-IAM State). A state s of the λ-IAM is a quintuple (t, C, L, T, d) where t is a λ-term, called the *code term*, C is a context, called the *code context*, L is a log, T is a tape, and d is an element of $\{\uparrow, \downarrow\}$, called the *direction*.

As for the Linear λ-IAM, directions shall be represented mostly via colors and underlinings, omitting the fifth component.

Initial States. The λ-IAM starts on *initial states* of the form $s_{t,k} := (t, \langle\cdot\rangle, \epsilon, \bullet^k, \downarrow)$, where t is a λ-term[7], $k \geq 0$ is the *depth* of the state, and ϵ is the empty log.

Transitions. The transitions of the λ-IAM are in Fig. 3. Their union is noted $\rightarrow_{\lambda\mathrm{IAM}}$. A state s is *reachable* if $s_{t,k} \rightarrow^*_{\lambda\mathrm{IAM}} s$ for an initial state $s_{t,k}$ and it is *final* if there exists no s' such that $s \rightarrow_{\lambda\mathrm{IAM}} s'$. The shape of final states is characterized in Sect. 5.

As for the Linear λ-IAM, $\downarrow$-states $(\underline{t}, C, L, T)$ are queries about the head variable of (the head normal form of) t and $\uparrow$-states $(t, \underline{C}, L, T)$ are queries about the argument of an abstraction.

With respect to the Linear λ-IAM, transitions $\rightarrow_{@1}, \rightarrow_{@2}, \rightarrow_{\lambda 1}$, and $\rightarrow_{\lambda 2}$ are respectively renamed $\rightarrow_{\bullet 1}, \rightarrow_{\bullet 2}, \rightarrow_{\bullet 4}$, and $\rightarrow_{\bullet 3}$, because $\bullet$ subsumes both token symbols @ and λ. The role of both symbols x and b is instead played by logged positions. Note that transition $\rightarrow_{\mathrm{arg}}$ moves the logged position from the tape to the log, and that transition $\rightarrow_{\mathrm{bt1}}$ moves it back to the tape, as it shall specify, at the end of the backtracking, to which variable occurrence $\rightarrow_{\mathrm{bt2}}$ has to move. The less intuitive aspect of the λ-IAM is the splitting of the log in transition $\rightarrow_{\mathrm{var}}$ and the dual concatenation of logs in

[4]An invariant shall guarantee that one never backtracks inside an abstraction whose variable has no occurrences.
[5]Such an index has a natural interpretation in linear logic terms. According to the standard (call-by-name) translation of the λ-calculus into linear logic proof nets, in a context C_n the hole lies inside exactly n !-boxes.

[6]In computer science logs are traces that can only grow, while here they also shrink. The terminology suggests a tracing mechanism—*trace* is avoided because related to categorical formulations of the GoI.
[7]To be precise, one needs a *well-named* term, that is, one in which all bound variables have distinct names, also distinct wrt free names. Since the code is immutable, this detail is only needed to assure that the relationship between variables and binders is unambiguous, for the definition of the transitions.

Sub-term	Context	Log	Tape		Sub-term	Context	Log	Tape
$\underline{tu}$	C	L	T	$\rightarrow_{\bullet 1}$	$\underline{t}$	$C\langle\langle\cdot\rangle u\rangle$	L	$\bullet \cdot T$
$\underline{\lambda x.t}$	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	$\rightarrow_{\mathsf{var}}$	$\lambda x.D_n\langle x\rangle$	$\underline{C}$	L	$(x, \lambda x.D_n, L_n) \cdot T$
$\underline{\lambda x.D_n\langle x\rangle}$	C	L	$(x, \lambda x.D_n, L_n) \cdot T$	$\rightarrow_{\mathsf{bt2}}$	x	$\underline{C\langle\lambda x.D_n\rangle}$	$L_n \cdot L$	T
t	$\underline{C\langle\langle\cdot\rangle u\rangle}$	L	$\bullet \cdot T$	$\rightarrow_{\bullet 3}$	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	$l \cdot T$	$\rightarrow_{\mathsf{arg}}$	$\underline{u}$	$C\langle t\langle\cdot\rangle\rangle$	$l \cdot L$	T
t	$\underline{C\langle u\langle\cdot\rangle\rangle}$	$l \cdot L$	T	$\rightarrow_{\mathsf{bt1}}$	$\underline{u}$	$C\langle\langle\cdot\rangle t\rangle$	L	$l \cdot T$

Figure 3: λ-IAM transitions.

$\rightarrow_{\mathsf{bt2}}$. To justify it, note first that the log is extended every time the machine enters into an argument via $\rightarrow_{\mathsf{arg}}$, which is also when the level of the position increases of 1. Note also that transitions $\rightarrow_{\mathsf{bt1}}$ moves out of an argument (decreasing the level of 1) and removes an entry from the log. To preserve the invariant that the log length is exactly the depth of the context, transition $\rightarrow_{\mathsf{var}}$ splits the log between the two positions $(\lambda x.D_n\langle x\rangle, C)$ and $(x, \lambda x.D_n)$, according to the depth of their contexts, and $\rightarrow_{\mathsf{bt2}}$ merges them back.

5 PROPERTIES OF THE λ-IAM

Here we first discuss a few invariants of the data structures of the machine, and then we analyze final states and the semantic interpretation defined by the λ-IAM.

The Code Invariant. An inspection of the rules shows that, along a computation, the machine travels on a λ-term without altering it.

Proposition 5.1 (Code Invariant). *If* $(t, C, L, T, d) \rightarrow_{\lambda IAM} (u, D, L', T', d')$*, then* $C\langle t\rangle = D\langle u\rangle$.

The Balance Invariant. 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), form a logged position, *i.e.* the length of L is exactly the level of the code context C. This fact guarantees that the λ-IAM never gets stuck because the log is not long enough for transitions $\rightarrow_{\mathsf{var}}$ and $\rightarrow_{\mathsf{bt1}}$ 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 balance 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 5.2 (Balance Invariant). *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*:* $d = \downarrow^{|T|_l}$.

Note that, because of the invariant, the tape T of a reachable $\uparrow$-state always contains at least one logged position, which is why it can be seen as the answer to a query about the head variable. More generally, the parity of a logged position l on the tape determines the role of l. If l is the n-th position on T (from the right) and n is odd, then l was added by $\rightarrow_{\mathsf{var}}$ and denotes a found variable waiting for an argument, while if n is even then l was added by $\rightarrow_{\mathsf{bt1}}$, its argument was already found, and the machine is backtracking to l.

The Exhaustible State Invariant. The study of the λ-IAM requires to prove that some bad configurations never arise. On states such as $(\underline{\lambda x.D\langle x\rangle}, C, L, l{\cdot}T)$, transition $\rightarrow_{\mathsf{bt2}}$ requires the logged position l to have shape $(x, \lambda x.D, L')$, that is, to contain a position isolating an occurrence of x in $\lambda x.D\langle x\rangle$, otherwise the machine is stuck. The *exhaustible state invariant* guarantees that the machine never gets stuck for this reason. The invariant being technical, it is developed in the Section 7. Here we only mention a key consequence.

Proposition 5.3 (Logged Positions Never Block the λ-IAM). *Let* $(\underline{\lambda x.D\langle x\rangle}, C, L, l{\cdot}T)$ *be a reachable state. Then* $l = (x, \lambda x.D, L')$.

Reversibility. The proof of Prop. 5.3 relies on a key property of the λ-IAM, that is, bi-determinism, or reversibility: the machine is deterministic, and moreover for each state s there is at most one state s' such that $s' \rightarrow_{\lambda\mathrm{IAM}} s$. The property follows by simply inspecting the rules. Moreover, a run can be reverted by just switching the direction.

Proposition 5.4 (Reversibility). *If* $(t, C, L, T, d) \rightarrow_{\lambda IAM} (u, D, L', T', d')$*, then* $(u, D, L', T', d'^1) \rightarrow_{\lambda IAM} (t, C, L, T, d^1)$.

Final States. A run of initial state $s_{t,k} = (\underline{t}, \langle\cdot\rangle, \epsilon, \bullet^k)$ may either never stop or end in one of three possible final states, as explained below. We shall prove this fact in Section 7 (Lemma 7.7). In order to explain the idea, let $\lambda x_0. \ldots \lambda x_i.(y u_1 \ldots u_j)$ be the head normal form $\mathsf{hnf}(t)$ of t. The three cases are:

- *Mute termination* $(\underline{\lambda x.u}, C, L, \epsilon)$: this is the machine's way of saying that $i > k$, that is, the run from $s_{t,k}$ (which allows to go under k abstractions) terminates but $\mathsf{hnf}(t)$ has more than k head abstractions, and so finding the head variable requires to increase the depth.
- *Open success* $(\underline{y}, C, L, \bullet^j)$: the machine has found the head variable, and it is the *free* variable y, which has j arguments and is under less than k head abstractions. Note that if y is instead bound by a λ-abstraction, then the machine is not stuck, as the machine would do a $\rightarrow_{\mathsf{var}}$ transition (as guaranteed by the balance invariant).

- *Bound success* $(t, \langle\cdot\rangle, L, \bullet^m \cdot l \cdot \bullet^j)$: the head variable has been found and it is $y = x_m$, to which j arguments are applied. When the machine $\downarrow$-travels on the head variable y, and it is abstracted, the logged position l containing x_m is put on the tape and the direction switches—the answer has been found. The sequence $\bullet^m$ on top of tape in the final state comes from the $\uparrow$ backtracking along the spine of $\mathsf{hnf}(t)$ for the equivalent of m abstractions, each one adding one $\bullet$. At this point the λ-IAM stops. Thus the abstraction binding y is λx_m, *i.e.* $y = x_m$.

The Semantics. The characterization of final states induces a semantic interpretation of terms, that we shall show to be sound and adequate with respect to (linear) head evaluation.

Definition 5.5 (λ-IAM Semantics). We define the λ-IAM semantics of λ-terms by way of a family of functions $[\![\cdot]\!]_k : \Lambda \to (\mathbb{N}\times\mathbb{N}) \cup (\mathcal{V}\times\mathbb{N}) \cup \{\Downarrow, \bot\}$, where $k \in \mathbb{N}$, defined as follows.

$$[\![t]\!]_k := \begin{cases} \langle h, j\rangle & \text{if } s_{t,k} \to^*_{\lambda\mathrm{IAM}} (t, \langle\cdot\rangle, \epsilon, \bullet^h \cdot l \cdot \bullet^j),\\ \langle x, h\rangle & \text{if } s_{t,k} \to^*_{\lambda\mathrm{IAM}} (\underline{x}, C, L, \bullet^h),\\ \Downarrow & \text{if } s_{t,k} \to^*_{\lambda\mathrm{IAM}} (\lambda x.u, C, L, \epsilon),\\ \bot & \text{if the } \lambda\text{-IAM diverges on } s_{t,k}.\end{cases}$$

5.1 Further Properties

The following properties of the λ-IAM are required for the proofs but are not essential for a first understanding of its functioning, so we suggest to skip them at a first reading.

Lifting. The λ-IAM verifies a sort of context-freeness with respect to the tape T. Intuitively, *lifting* the tape preserves the shape of the run and of the final state (up to lifting).

Lemma 5.6 (Lifting). *If $(t, C, L, T, d) \to^n_{\lambda IAM} (u, D, L', T', d')$, then $(t, C, L, T{\cdot}T'', d) \to^n_{\lambda IAM} (u, D, L', T'{\cdot}T'', d')$.*

Monotonicity of Runs. The previous lemma states that lifting the input from $\bullet^k$ to $\bullet^{k+1}$ cannot decrease the length of the λ-IAM run. Next, we show that if the run of input $\bullet^k$ is successful then the run of input $\bullet^{k+1}$ is also successful, in the same way, and it has the same length. As a consequence, the length may increase only if the run on $\bullet^k$ mutely terminates.

We write $|t|_k$ for the length of the λ-IAM run of initial state $s_{t,k} := (\underline{t}, \langle\cdot\rangle, \epsilon, \bullet^k)$, that is for the length of the maximum sequence of transitions from $s_{t,k}$, if the λ-IAM terminates, and $|t|_k = \infty$ if the machine diverges. The next lemma compares run lengths, for which we consider that $i < \infty$ for every $i \in \mathbb{N}$ and $\infty \not< \infty$. We also write $s^n_{t,k}$ for the state such that $s_{t,k} \to^n_{\lambda\mathrm{IAM}} s^n_{t,k}$, if it exists.

Lemma 5.7 (Monotonicity of runs). *The length of runs cannot decrease if the input increases, that is, $|t|_k \le |t|_{k+1}$. Moreover, if $|t|_k = n \in \mathbb{N}$ and the final state $s^n_{t,k}$ is bound (resp. open) successful then $|t|_k = |t|_h$ for every $h > k$ and the final state $s^n_{t,h}$ is bound (resp. open) successful.*

6 MICRO-STEP REFINEMENT

The proof of soundness of the λ-IAM cannot be directly carried out with respect to head evaluation: this is specified using meta-level substitutions, here noted $t\{x{\leftarrow}u\}$, which is a macro operation, potentially making *many* copies of u and modifying t in *many* places, while the λ-IAM does a minimalistic evaluation that in general does not even pass through most of those many places. It is very hard—if possible at all—to define explicitly a bisimulation of λ-IAM runs (as required for soundness) that relates states whose code is modified by meta-level substitution.

We then switch to *linear* head evaluation (shortened to LHE), a refinement of head evaluation in which substitution is performed in *micro-steps*, replacing only the head variable occurrence, and keeping the substitution suspended for all the other occurrences. This is also the approach followed by Danos, Herbelin, and Regnier [25].

We depart from their approach, however, in the way we formally define LHE. We adopt a formulation where the suspension of the substitution is formalized via a sharing constructor $t[x{\leftarrow}u]$, which is nothing else but a compact notation for $\mathtt{let}\ x = u\ \mathtt{in}\ t$, and the rewriting is modified accordingly. They instead avoid sharing, by encoding $t[x{\leftarrow}u]$ as $(\lambda x.t)u$, which is more compact but conflates different concepts and makes the technical development less clean.

An important point is that head evaluation and its linear variant are observationally equivalent, that is, one terminates on t if and only if the other terminates on t, and they produce the same head variable, as we shall discuss below.

The Adopted Presentation. Linear head evaluation was introduced by Mascari & Pedicini and Danos & Regnier [28, 43] as a strategy on proof nets. It is to proof nets for the λ-calculus what head evaluation is to the λ-calculus. The presentation adopted here, noted $\to_{\mathsf{lh}}$, was introduced by Accattoli [2], formulated as a strategy in a λ-calculus with explicit sharing, the *linear substitution calculus*[8] (shortened to LSC). The LSC presentation of $\to_{\mathsf{lh}}$ is isomorphic to the one on proof nets [4], while the one used by Danos and Regnier—although closely related to proof nets—is not. It is isomorphic only up to Regnier's σ-equivalence [48].

LSC Terms and Leveled contexts. Let $\mathcal{V}$ be a countable set of variables. Terms of the *linear substitution calculus* (LSC) are defined by the following grammar.

$$\text{LSC terms} \qquad t, u, r \quad ::= \quad x \in \mathcal{V} \mid \lambda x.t \mid tu \mid t[x{\leftarrow}u].$$

The construct $t[x{\leftarrow}u]$ is called an *explicit substitution* or ES, not to be confused with meta-level substitution $t\{x{\leftarrow}u\}$. As is standard, $t[x{\leftarrow}u]$ binds x in t, but not in u—terms are still considered up to α-conversion. Leveled contexts naturally extend to the LSC.

$$\begin{array}{lcl} \multicolumn{3}{l}{\text{Leveled Contexts}}\\ C_0 & ::= & \langle\cdot\rangle \mid \lambda x.C_0 \mid C_0 t \mid C_0[x{\leftarrow}t];\\ C_{n+1} & ::= & \lambda x.C_{n+1} \mid C_{n+1}t \mid C_{n+1}[x{\leftarrow}t] \mid tC_n \mid t[x{\leftarrow}C_n]. \end{array}$$

Contexts and Plugging. The LSC makes a crucial use of contexts to define its operational semantics. First of all, we need *substitution contexts*, that simply pack together ES:

$$\text{Substitution contexts} \qquad S \quad ::= \quad \langle\cdot\rangle \mid S[x{\leftarrow}t].$$

When plugging is used for substitution contexts, we write it in a post-fixed manner, that is $\langle t\rangle S$, to stress that the ES actually appears on the right of t.

[8]The LSC is a subtle reformulation of Milner's calculus with explicit substitutions [37, 46], inspired by Accattoli and Kesner structural λ-calculus [11].

$$\begin{array}{ccc} \textsc{Rules at top level} & \textsc{Contextual closure} & \textsc{Notation} \\ \begin{array}{rcl} \langle \lambda x.t\rangle Su & \mapsto_{\mathsf{dB}} & \langle t[x{\leftarrow}u]\rangle S \\ H\langle x\rangle[x{\leftarrow}t] & \mapsto_{\mathsf{ls}} & H\langle t\rangle[x{\leftarrow}t] \\ t[x{\leftarrow}u] & \mapsto_{\mathsf{gc}} & t \quad \text{if } x \notin \mathsf{fv}(t) \end{array} & \dfrac{t \mapsto_{\mathsf{a}} u}{H\langle t\rangle \to_{\mathsf{a}} H\langle u\rangle}\ \mathsf{a} \in \{\mathsf{dB}, \mathsf{ls}, \mathsf{gc}\} & \to_{\mathsf{lh}} := \to_{\mathsf{dB}} \cup \to_{\mathsf{ls}} \cup \to_{\mathsf{gc}} \end{array}$$

Figure 4: Rewriting rules for linear head evaluation $\to_{\mathsf{lh}}$.

Sub-term	Context	Log	Tape		Sub-term	Context	Log	Tape
$\underline{t[x{\leftarrow}u]}$	C	L	T	$\to_{\mathsf{es}}$	$\underline{t}$	$C\langle\langle\cdot\rangle[x{\leftarrow}u]\rangle$	L	T
$\underline{x}$	$C\langle D_n[x{\leftarrow}u]\rangle$	$L_n \cdot L$	T	$\to_{\mathsf{var2}}$	$\underline{u}$	$C\langle D_n\langle x\rangle[x{\leftarrow}\langle\cdot\rangle]\rangle$	$(x, D_n[x{\leftarrow}u], L_n)\cdot L$	T
t	$\underline{C\langle\langle\cdot\rangle[x{\leftarrow}u]\rangle}$	L	T	$\to_{\mathsf{es2}}$	$t[x{\leftarrow}u]$	$\underline{C}$	L	T
u	$\underline{C\langle D_n\langle x\rangle[x{\leftarrow}\langle\cdot\rangle]\rangle}$	$(x, D_n[x{\leftarrow}u], L_n)\cdot L$	T	$\to_{\mathsf{var3}}$	x	$\underline{C\langle D_n[x{\leftarrow}u]\rangle}$	$L_n \cdot L$	T

Figure 5: Transitions for LSC-terms.

Linear Head Evaluation. The LSC comes with a notion of reduction that resembles the decomposed, micro-step process of cut-elimination in linear logic proof-nets. Essentially, the meta-level substitution $t\{x{\leftarrow}u\}$ is decomposed into a sequence replacements from $t[x{\leftarrow}u]$ of one occurrence of x in t with u at the time. Linear head evaluation, moreover, is the reduction that replaces only the head variable occurrence y, if it is bound by an ES $[y{\leftarrow}r]$ and leaves the other occurrences of y, if any, bound by $[y{\leftarrow}r]$.

The rewriting rules, in Figure 4, are first defined at top level and then closed by head contexts. A feature of the LSC is that contexts are also used to define the linear substitution rule at top level $\mapsto_{\mathsf{ls}}$. In plugging t in H, rule $\to_{\mathsf{ls}}$ may perform on-the-fly renaming of bound variables in H, to avoid capture of free variables of t. Often, the literature does not include rule $\to_{\mathsf{gc}}$, responsible for erasing steps, in the definition of $\to_{\mathsf{lh}}$. The reason is that $\to_{\mathsf{gc}}$ is strongly normalizing and it can be postponed. Note that our definition of $\to_{\mathsf{lh}}$ is non-deterministic, for instance $t := (\lambda x.(y[y{\leftarrow}u]))r \to_{\mathsf{ls}} (\lambda x.(u[y{\leftarrow}u]))r$ and $t \to_{\mathsf{dB}} x[y{\leftarrow}u][x{\leftarrow}r]$. It is not a problem, as $\to_{\mathsf{lh}}$ has the diamond property—this is standard, see [2].

Example 6.1. We provide here an example of LHE sequence. Consider the following 3 steps:

$$\begin{array}{rcl} (\lambda x.xx)(\lambda y.y) & \to_{\mathsf{dB}} & (xx)[x{\leftarrow}\lambda y.y] \to_{\mathsf{ls}} ((\lambda y.y)x)[x{\leftarrow}\lambda y.y] \\ & \to_{\mathsf{dB}} & y[y{\leftarrow}x][x{\leftarrow}\lambda y.y] \end{array}$$

that turn a β/multiplicative redex into a ES, substitute on the head variable occurrence, and continue with another multiplicative step. Two linear substitution steps on the head, followed by two steps of garbage collection complete the evaluation:

$$\begin{array}{rcl} y[y{\leftarrow}x][x{\leftarrow}\lambda y.y] & \to_{\mathsf{ls}} & x[y{\leftarrow}x][x{\leftarrow}\lambda y.y] \\ & \to_{\mathsf{ls}} & (\lambda y.y)[y{\leftarrow}x][x{\leftarrow}\lambda y.y] \to^2_{\mathsf{gc}} \lambda z.z \end{array}$$

Additional λ-IAM transitions. The λ-IAM presented in the previous sections is easily adapted to the LSC, by simply considering (logged) positions with respect to the extended syntax, and adding the 4 transitions for ES in Fig. 5.

Transitions $\to_{\mathsf{es}}$ and $\to_{\mathsf{es2}}$ simply skip ES during search—now search is *up to β-redexes and ES.* Transition $\to_{\mathsf{var2}}$ shortcuts the search of the term u to substitute for x, given that u is already available in $[x{\leftarrow}u]$. Therefore, the machine stays in the $\downarrow$ phase and moves to evaluate u. Note that the logged position for x is directly added to the log and not to the tape. This is because we have avoided the search of the argument. We have reached it directly: note that when a $\uparrow$-search ends with the $\to_{\mathsf{arg}}$ transition, the logged position indeed goes from the tape to the log. Transition $\to_{\mathsf{var3}}$ is dual to $\to_{\mathsf{var2}}$, and it is used to keep looking for arguments when the current subterm u has none left.

All results and considerations of Sect. 4 and 5 still hold in this more general setting, *mutatis mutandis.*

Relationship with Head Evaluation, and Normal Forms. We shall prove that the λ-IAM is sound and adequate with respect to linear head evaluation $\to_{\mathsf{lh}}$, by showing that it approximates the spine structure of the $\to_{\mathsf{lh}}$-normal form of t, when it exists. Let us explain why the same holds also for head evaluation. The relationship between linear head and head evaluation is studied in detail by Accattoli and Dal Lago [12], who prove that the two notions are termination equivalent and produce the same normal forms up to unfolding. The definition of unfolding $t{\downarrow}$ of a term with ES follows.

$$\begin{array}{rclcrcl} x\downarrow & := & x & \quad & (tu)\downarrow & := & t\downarrow u\downarrow \\ (\lambda x.t)\downarrow & := & \lambda x.t\downarrow & & (t[x{\leftarrow}u])\downarrow & := & t\downarrow\{x \leftarrow u\downarrow\} \end{array}$$

Proposition 6.2 ([12]). *Let t be a λ-term. There exists a head evaluation $t \to^*_{\mathsf{h}} hnf(t)$ to head normal form if and only if there exists a linear head evaluation $t \to^*_{\mathsf{lh}} lhnf(t)$ to linear head normal form. Moreover, $lhnf(t){\downarrow} = hnf(t)$.*

Additionally, $\mathsf{lhnf}(t)$ and $\mathsf{hnf}(t)$ have the same spine structure. A linear head normal form has the same shape $\lambda x_1. \ldots \lambda x_k.(yt_1 \ldots t_h)$ of a head normal form but for the fact that each spine sub-term may be surrounded by a substitution context S, that is, they have the cumbersome shape (where S_i surrounds $\lambda x_i. \ldots \lambda x_k.(yt_1 \ldots t_h)$ and S'_j surrounds $yt_1 \ldots t_j$):

$$\langle \lambda x_1.\langle \lambda x_2. \ldots \langle \lambda x_k.(\langle\langle\langle y\rangle t_1\rangle S'_1 \ldots t_h\rangle S'_h\rangle)S_k \ldots\rangle S_2\rangle S_1$$

where none of the ES in S_i and S'_j binds y (otherwise there would be a $\to_{\mathsf{ls}}$ redex)[9].

It is fairly evident that if t is $\to_{\mathsf{lh}}$ normal then t and $t{\downarrow}$ have the same spine structure (that is, the number of head abstractions, the identity of the head variable, and the number of its arguments): the only substitutions that may alter the spine are those acting on the

[9]An inductive characterization of linear head normal forms can be given along the lines of the one in Figure 11 at page 36 of [9]. The minor difference is that in that paper $\to_{\mathsf{lh}}$ does not include $\to_{\mathsf{gc}}$.

head variable, but $t{\downarrow}$ unfolds no such substitutions, because no ES of t can bound the head variable, since t is $\to_{\mathsf{ls}}$ normal. In the following, we shall then consider a $\to_{\mathsf{lh}}$-normal LSC term t and without loss of generality consider its shape to be $\lambda x_1 \ldots \lambda x_k.(yt_1 \ldots t_h)$, that is, as if it were a $\to_{\mathsf{h}}$-normal λ-term.

7 THE EXHAUSTIBLE STATE INVARIANT

The previous sections introduced all the ingredients for the formal study of the λ-IAM. From now on, we turn to the development of the proofs of soundness and adequacy. The first step, taken here, is to formalize the exhaustible state invariant mentioned in Sect. 5.

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.

Why It Is Needed. The exhaustible state invariant is meant to show that some undesirable configurations never arise, to characterize the final states of the λ-IAM. On states such as $(\underline{\lambda x.D\langle x\rangle}, C, L, l{\cdot}T)$ the λ-IAM requires the logged position l to have shape $(x, \lambda x.D, L')$, that is, to be associated to a position isolating an occurrence of x in $\lambda x.D\langle x\rangle$, otherwise the machine is stuck. Similarly, on states such as $(t, \underline{C\langle D\langle x\rangle[x{\leftarrow}\langle\cdot\rangle]\rangle}, l \cdot L, T)$ the position of l is expected to isolate an occurrence of x in $D\langle x\rangle$, or the machine is stuck. Luckily, the machine is never stuck for these reasons, and exhaustible states are the technical tool to prove it[10]. The invariant also allows to prove that the backtracking to a logged position l always ends on a state of position l, as expected—that is, backtracking always succeeds, and to characterize the structure of the tape in final states.

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 those in the log of s.

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

Definition 7.1 (Tape tests). Let $s = (t, C, L, T'{\cdot}l{\cdot}T'', d)$ be a state. Then the *tape test of s of focus l* is the state $s_l = (t, C, L, T'{\cdot}l, \uparrow^{|T'\cdot l|_l})$.

Note that the direction of tape tests is reversed with respect to what stated by the *tape and direction* invariant (Lemma 5.2), 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 proving that backtracking always succeeds (Lemma 7.6 below), we shall extend their tape via the tape lifting property (Lemma 5.6) as to satisfy the invariant and be reachable.

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 state position still have to form a logged position, which requires to change the state position—said differently, the *position and the log* invariant (Lemma 5.2) has to be preserved.

In the applications of the exhaustible invariant given below, we need only log tests of a very simple form. Namely, given a state $s = (t, \underline{C}, l \cdot L, T)$, we shall consider the log test $s_l := (t, \underline{C}, l \cdot L, \epsilon)$, obtained by emptying the tape and (in this case) wihtout changing the position. The more general form of log tests needing the position change is technical and defined at the end of the section—it is unavoidable for proving the invariant, but we fear that giving it here would obfuscate the use of the exhaustible invariant, whose idea is instead quite simple.

The Exhaustibility Invariant. Exhausting a logged position l means backtracking to it. We then decorate the backtracking transition $\to_{\mathtt{bt1}}$ and $\to_{\mathtt{bt2}}$ as $\to_{\mathtt{bt1},l}$ and $\to_{\mathtt{bt2},l}$ to specify the involved logged position l. We also need a notion of state positioned in l, which is the target state of $\to_{\mathtt{bt2},l}$.

Definition 7.2 (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 context C_n and log L_n.

After having introduced all the necessary preliminaries, we can define exhaustible states.

Definition 7.3 (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 $s_l \to^*_{\lambda\mathrm{IAM}} \to_{\mathtt{bt2},l} s'' \in \mathcal{E}$, where s'' surrounds l. 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 7.4 (Exhaustible invariant). *Let s be a λ-IAM reachable state. Then s is exhaustible.*

The proof of Prop. 7.4 is long, but logically quite simple, being structured around a simple induction on the length of the run from the initial state to s, and can be found in the Extended Version of this paper [13].

Consequences of the Exhaustible Invariant. First, the λ-IAM never gets stuck for a mismatch of logged positions.

Corollary 7.5 (Logged Positions Never Block the λ-IAM). *Let s be a reachable state.*

1. *If $s = (\underline{\lambda x.D\langle x\rangle}, C, L, l{\cdot}T)$ then s is not final.*
2. *If $s = (u, \underline{C\langle D\langle x\rangle[x \leftarrow \langle\cdot\rangle]\rangle}, l \cdot L, T)$ then s is not final.*

Proof. For point 1, by the exhaustible invariant (Prop. 7.4), s is exhaustible. Then, its tape test $(\underline{\lambda x.D\langle x\rangle}, C, L, l)$ does at least one transition towards a state $s' \neq s$ surrounding l. Point 2 is analogous, just consider the log test $s' = (u, \underline{C\langle D\langle x\rangle[x \leftarrow \langle\cdot\rangle]\rangle}, l \cdot L, \epsilon)$. □

The second consequence is that backtracking always succeeds.

Lemma 7.6 (Backtracking always succeeds). *Let s a reachable state. If $s \to_{\mathtt{bt1},l} s'$ then there is s'' such that $s' \to^*_{\lambda IAM} \to_{\mathtt{bt2},l} s''$.*

[10] One could redefine the transitions of the λ-IAM asking—for these states—to jump to whatever variable position is in the logged position l. Then the λ-IAM would not get stuck, and the invariant would not be needed for characterizing final states, but we would then need it for soundness—there is no easy way out.

Proof. Consider $s = (t, \underline{C\langle u\langle\cdot\rangle\rangle}, l \cdot L, T) \to_{\mathrm{bt1},l} (\underline{u}, C\langle\langle\cdot\rangle t\rangle, L, l \cdot T) = s'$. 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 run $\pi : s'_l \to^*_{\lambda\mathrm{IAM}} \to_{\mathrm{bt2},l} q$ for a state q surrounding l. Note that s'_l is s' with empty tape. Now, we lift π to a run $\pi^T : s' \to^*_{\lambda\mathrm{IAM}} \to_{\mathrm{bt2},l} s''$ by using the tape lifting lemma (Lemma 5.6). □

Finally, we characterize final states, as anticipated in Section 5.

Lemma 7.7 (Final states). *Let $s_t \to^*_{\lambda IAM} s$ be a run ending on a final state. Then s has one of the following three shapes: $(\underline{\lambda x.u}, C, L, \epsilon)$, $(\underline{y}, C, L, \bullet^j)$ with y free in $C\langle y\rangle$, or $(t, \underline{\langle\cdot\rangle}, \epsilon, \bullet^m \cdot l \cdot \bullet^j)$.*

Proof. The λ-IAM is never stuck on $\to_{\bullet 1}$, $\to_{\bullet 4}$, $\to_{\mathrm{es}}$, and $\to_{\mathrm{es2}}$. By the balance invariant (Lemma 5.2), it is also never stuck on $\to_{\mathrm{var}}$, $\to_{\mathrm{var2}}$, and $\to_{\mathrm{bt1}}$ because the log has not enough entries. Note also that if the position is $(t, C\langle\langle\cdot\rangle u\rangle)$ and the direction is $\uparrow$, one among $\to_{\bullet 3}$ and $\to_{\mathrm{arg}}$ always applies, as the tape cannot be empty, by the balance invariant. By Corollary 7.5, the λ-IAM cannot be stuck on $\to_{\mathrm{bt2}}$ and $\to_{\mathrm{var3}}$. Then, the machine is stuck in three cases only. First, on $\to_{\bullet 2}$ if the tape is empty—the stuck state is $(\underline{\lambda x.u}, C, L, \epsilon)$. Second, in $\to_{\mathrm{var}}$ or $\to_{\mathrm{var2}}$ if the variable x is free—the state is $(\underline{y}, C, L, T)$. Third, in direction $\uparrow$ if the context is empty—the state is $(t, \underline{\langle\cdot\rangle}, \epsilon, T)$.

Let's now characterize the shape of the tape T. For $(\underline{y}, C, L, T)$, the tape T cannot contain any logged position. Otherwise, its tape would be $T = \bullet^k \cdot l \cdot T'$ and by the exhaustible invariant, one could consider the tape test $s_l = (\underline{y}, C, L, \bullet^k \cdot l)$, such that $s_l \to^+_{\lambda\mathrm{IAM}} s'$, where s' surrounds l. This is impossible since s_l is final as well. Finally, the λ-IAM stuck on $\underline{\langle\cdot\rangle}$ needs to have exactly one logged position on the tape. Indeed, the number n of logged positions on the tape has to be odd. However, $n \le 1$ because otherwise one could consider the tape test relative to the second logged position that would run in the same contradiction as in the previous case. □

Log Tests and Position Changes. To define the log test focussing on the m-th logged position l_m in the log of a state $(t, C_n, l_n \cdots l_2 \cdot l_1, T, d)$, 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 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, and a term r satisfying exactly one of the two following conditions (levels can be incremented in two ways).

- Case $t = C_m\langle rC_k\langle u\rangle\rangle$. 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 $(C_k\langle u\rangle, O_{m+1})$.
- Case $t = C_m\langle r[x \leftarrow C_k\langle u\rangle]\rangle$. Then, the $m+1$-*outer context* of the position (u, C_{n+1}) is the context $O_{m+1} := C_m\langle r[x \leftarrow \langle\cdot\rangle]\rangle$ 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 $1 \le m \le n+1$, 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 7.8 (Log tests). Let $s = (t, C_n, l_n \cdots l_2 \cdot l_1, T, d)$ be a state with $1 \le m \le n$, and (u, O_m) be the m-outer position of (t, C_n). The *m-log test of s of focus l_m* is the state $s_{l_m} := (u, O_m, l_m \cdots l_2 \cdot l_1, \epsilon, \uparrow)$.

8 IMPROVEMENTS, CONCRETELY

In this section we define an improvement $\blacktriangleright$ relation for the λ-IAM, to be used in the sequel to prove soundness and adequacy.

Given a $\to_{\mathrm{lh}}$-step $t \to_{\mathrm{lh}} u$, the improvement $\blacktriangleright$ has to relate states of code t with states of code u. Since $\to_{\mathrm{lh}}$ is the union of the three rewriting rules $\to_{\mathrm{dB}}$, $\to_{\mathrm{ls}}$ and $\to_{\mathrm{gc}}$, we are going to define $\blacktriangleright$ as the union of three improvements $\blacktriangleright_{\mathrm{dB}}$, $\blacktriangleright_{\mathrm{ls}}$, and $\blacktriangleright_{\mathrm{gc}}$.

Improvement for $\to_{\mathrm{dB}}$. Lifting a step $t \to_{\mathrm{lh}} u$ to a relation between a λ-IAM state s of code t and a state q of code u requires changing all positions relative to t in s to positions relative to u in q. Note that all the positions in the token have to be changed, so that $\blacktriangleright$ has to relate positions, logged positions, tape, log, and states.

Explaining the Need of Context Rewriting. A second more technical aspect is that one needs to extend linear head evaluation to contexts. Consider a step $t \to_{\mathrm{dB}} u$ where—for simplicity—the redex is at top level and the associated state $(\underline{\langle\lambda x.r\rangle Sw}, \langle\cdot\rangle, \epsilon, \epsilon)$ has an empty token. This should be $\blacktriangleright_{\mathrm{dB}}$-related to a state $(\underline{\langle r[x{\leftarrow}w]\rangle S}, \langle\cdot\rangle, \epsilon, \epsilon)$. Let's have a look at how the two states evolve:

$$\begin{tikzcd}
(\underline{\langle\lambda x.r\rangle Sw}, \langle\cdot\rangle, \epsilon, \epsilon) \arrow[d] \arrow[rr, phantom, "\blacktriangleright_{\mathrm{dB}}"] & & (\underline{\langle r[x{\leftarrow}w]\rangle S}, \langle\cdot\rangle, \epsilon, \epsilon) \arrow[dd, "|S|"] \\
(\underline{\langle\lambda x.r\rangle S}, \langle\cdot\rangle w, \epsilon, \bullet) \arrow[d, "|S|"] & & \\
(\underline{\lambda x.r}, Sw, \epsilon, \bullet) \arrow[d] & & (\underline{r[x{\leftarrow}w]}, S, \epsilon, \epsilon) \arrow[d] \\
(\underline{r}, \langle\lambda x.\langle\cdot\rangle\rangle Sw, \epsilon, \epsilon) & & (\underline{r}, \langle\langle\cdot\rangle[x{\leftarrow}w]\rangle S, \epsilon, \epsilon)
\end{tikzcd}$$

To close the diagram, we need $\blacktriangleright_{\mathrm{dB}}$ to relate the two bottom states. Note that their relation can be seen as a $\to_{\mathrm{dB}}$ step involving the contexts of the two positions. Therefore we extend the definition of $\to_{\mathrm{dB}}$ to contexts adding the following top level clause (then included in $\to_{\mathrm{dB}}$ via a closure by head contexts): $\langle\lambda x.C\rangle St \mapsto_{\mathrm{dB}} \langle C[x{\leftarrow}t]\rangle S$. The new clause, in turn, requires a further extension of $\to_{\mathrm{dB}}$ (again closed by head contexts): $\langle\lambda x.t\rangle SC \mapsto_{\mathrm{dB}} \langle t[x{\leftarrow}C]\rangle S$.

Note that in the shown local bisimulation diagram the right side is shorter. This is typical of when the machine travels through the redex. Outside of the redex, however, the two sides have the same length, as the next example shows—example that also motivates a further extension of $\to_{\mathrm{dB}}$ to contexts. Consider the case where $t \to_{\mathrm{dB}} u$ and the diagram is (the states do a $\to_{\mathrm{arg}}$ transition):

$$\begin{tikzcd}
(t, \underline{\langle\cdot\rangle r}, \epsilon, l) \arrow[d] \arrow[r, phantom, "\blacktriangleright_{\mathrm{dB}}"] & (u, \underline{\langle\cdot\rangle r}, \epsilon, l) \arrow[d] \\
(\underline{r}, t\langle\cdot\rangle, l, \epsilon) & (\underline{r}, u\langle\cdot\rangle, l, \epsilon)
\end{tikzcd}$$

We then need to extend $\to_{\mathrm{dB}}$ so that $t\langle\cdot\rangle \to_{\mathrm{dB}} u\langle\cdot\rangle$. A similar situation happens also when entering an ES with transition $\to_{\mathrm{var2}}$. To close these diagrams, we add two further cases of reduction on contexts. Note that this time they have to be expressed via steps on terms (then included in $\to_{\mathrm{dB}}$ via a closure by head contexts), as their direct definition would require contexts with two holes. Of course, the same situation arises with ls and gc steps.

$$\frac{t \to_{\mathsf{a}} u}{tC \to_{\mathsf{a}} uC} \qquad \frac{t \to_{\mathsf{a}} u}{t[x{\leftarrow}C] \to_{\mathsf{a}} u[x{\leftarrow}C]} \quad \mathsf{a} \in \{\mathsf{dB}, \mathsf{ls}, \mathsf{gc}\}.$$

Definition 8.1. The (overloaded) binary relation $\blacktriangleright_{\mathsf{dB}}$ between positions, stacks, and states is defined by the following rules[11].

$$\frac{t \to_{\mathsf{dB}} u}{(t, H) \blacktriangleright_{\mathsf{dB}} (u, H)}\ \mathsf{rdx_{dB}} \qquad \frac{C \to_{\mathsf{dB}} D}{(t, C) \blacktriangleright_{\mathsf{dB}} (t, D)}\ \mathsf{ctx_{dB}}$$

$$\frac{}{\epsilon \blacktriangleright_{\mathsf{dB}} \epsilon}\ \mathsf{tok1_{dB}} \qquad \frac{T \blacktriangleright_{\mathsf{dB}} T'}{\bullet \cdot T \blacktriangleright_{\mathsf{dB}} \bullet \cdot T'}\ \mathsf{tok2_{dB}}$$

$$\frac{(x, C) \blacktriangleright_{\mathsf{dB}} (x, D) \qquad L \blacktriangleright_{\mathsf{dB}} L'}{(x, C, L) \blacktriangleright_{\mathsf{dB}} (x, D, L')}\ \mathsf{pos_{dB}} \qquad \frac{l \blacktriangleright_{\mathsf{dB}} l' \qquad \Gamma \blacktriangleright_{\mathsf{dB}} \Gamma'}{l \cdot \Gamma \blacktriangleright_{\mathsf{dB}} l' \cdot \Gamma'}\ \mathsf{tok3_{dB}}$$

$$\frac{(t, C) \blacktriangleright_{\mathsf{dB}} (u, D) \qquad T \blacktriangleright_{\mathsf{dB}} T' \qquad L \blacktriangleright_{\mathsf{dB}} L' \qquad d = d'}{(t, C, L, T, d) \blacktriangleright_{\mathsf{dB}} (u, D, L', T', d')}\ \mathsf{state_{dB}}$$

Note that $\blacktriangleright_{\mathsf{dB}}$ contains all pairs $((\underline{t}, \langle\cdot\rangle, \epsilon, \bullet^k), (\underline{u}, \langle\cdot\rangle, \epsilon, \bullet^k))$, where $t \to_{\mathsf{dB}} u$, *i.e.* all the pairs of initial states involving a dB-redex.

Improvement for $\to_{\mathsf{ls}}$. As for $\to_{\mathsf{dB}}$, the improvement for $\to_{\mathsf{ls}}$ requires extending the rewriting relation to contexts. There are however some new subtleties. Given $t \to_{\mathsf{dB}} u$ and a position (r, C) for t, for $\blacktriangleright_{\mathsf{dB}}$ the redex in t falls always entirely either in t or C. If $t \to_{\mathsf{ls}} u$, instead, the redex can be split between the two. Consider the following diagram (where to simplify we assume the step to be at top level and the token to be empty).

$$\begin{tikzcd}
(\underline{H\langle x\rangle[x{\leftarrow}r]}, \langle\cdot\rangle, \epsilon, \epsilon) \arrow[r, phantom, "\blacktriangleright_{\mathsf{ls}}"] \arrow[d] & (\underline{H\langle r\rangle[x{\leftarrow}r]}, \langle\cdot\rangle, \epsilon, \epsilon) \arrow[d] \\
(\underline{H\langle x\rangle}, \langle\cdot\rangle[x{\leftarrow}r], \epsilon, \epsilon) & (\underline{H\langle r\rangle}, \langle\cdot\rangle[x{\leftarrow}r], \epsilon, \epsilon)
\end{tikzcd}$$

To close it, we have to $\blacktriangleright_{\mathsf{ls}}$-relate the two bottom states, where the pattern of the redex/reduct is split between the two parts of the position. This motivates clause rdx2 in the definition of $\blacktriangleright_{\mathsf{ls}}$ below.

The new rule comes with consequences. Consider the following diagram involving the new clause for $\blacktriangleright_{\mathsf{ls}}$:

$$\begin{tikzcd}
(\underline{x}, H[x{\leftarrow}t], \epsilon, \epsilon) \arrow[r, phantom, "\blacktriangleright_{\mathsf{ls}}"] \arrow[d] & (\underline{t}, H[x{\leftarrow}t], \epsilon, \epsilon) =: q \\
s := (\underline{t}, H\langle x\rangle[x{\leftarrow}\langle\cdot\rangle], (x, H[x{\leftarrow}t], \epsilon), \epsilon) &
\end{tikzcd}$$

To close the diagram, we have to $\blacktriangleright_{\mathsf{ls}}$-relate s and q. There are, however, two delicate points. First, we cannot see the context $H\langle x\rangle[x{\leftarrow}\langle\cdot\rangle]$ as making a $\to_{\mathsf{ls}}$ step towards $H[x{\leftarrow}t]$, because t does not occur in $H\langle x\rangle[x{\leftarrow}\langle\cdot\rangle]$. For that, we have to introduce a variant of $\to_{\mathsf{ls}}$ on contexts that is parametric in t (and more general than the one to deal with the showed simplified diagram):

$$H\langle x\rangle[x{\leftarrow}C] \quad \mapsto_{\mathsf{ls},t} \quad H\langle C\rangle[x{\leftarrow}C\langle t\rangle].$$

The second delicate point of diagram (1) is that the extension of $\blacktriangleright_{\mathsf{ls}}$ has to also $\blacktriangleright_{\mathsf{ls}}$-relate logs of different length, namely ϵ and $(x, H[x{\leftarrow}t], \epsilon)$. This happens because positions of the two states do isolate the same term, but at different depths, as one is in the ES. Then the definition of $\blacktriangleright_{\mathsf{ls}}$ has two clauses, one for logs ($\mathsf{pos2_{ls}}$) and one for states ($\mathsf{state2_{ls}}$), to handle such a case. The mismatch in logs lengths is at most 1.

Definition 8.2. The binary relation $\blacktriangleright_{\mathsf{ls}}$ is defined by:

$$\frac{t \to_{\mathsf{ls}} u}{(t, H) \blacktriangleright_{\mathsf{ls}} (u, H)}\ \mathsf{rdx_{ls}} \qquad \frac{C \to_{\mathsf{ls}} D}{(t, C) \blacktriangleright_{\mathsf{ls}} (t, D)}\ \mathsf{ctx_{ls}}$$

$$\frac{K = K'\langle G[x \leftarrow t]\rangle}{(H\langle x\rangle, K) \blacktriangleright_{\mathsf{ls}} (H\langle t\rangle, K)}\ \mathsf{rdx2} \qquad \frac{}{\epsilon \blacktriangleright_{\mathsf{ls}} \epsilon}\ \mathsf{tok1_{ls}}$$

$$\frac{T \blacktriangleright_{\mathsf{ls}} T'}{\bullet \cdot T \blacktriangleright_{\mathsf{ls}} \bullet \cdot T'}\ \mathsf{tok2_{ls}} \qquad \frac{l \blacktriangleright_{\mathsf{ls}} l' \qquad \Gamma \blacktriangleright_{\mathsf{ls}} \Gamma'}{l \cdot \Gamma \blacktriangleright_{\mathsf{ls}} l' \cdot \Gamma'}\ \mathsf{tok3_{ls}}$$

$$\frac{(x, C) \blacktriangleright_{\mathsf{ls}} (x, D) \qquad L \blacktriangleright_{\mathsf{ls}} L'}{(x, C, L \blacktriangleright_{\mathsf{ls}} (x, D, L')}\ \mathsf{pos_{ls}} \qquad \frac{C \to_{\mathsf{ls},x} D \qquad L \blacktriangleright_{\mathsf{ls}} L'}{(x, C, L \cdot l) \blacktriangleright_{\mathsf{ls}} (x, D, L')}\ \mathsf{pos2_{ls}}$$

$$\frac{(t, C) \blacktriangleright_{\mathsf{ls}} (u, D) \qquad T \blacktriangleright_{\mathsf{ls}} T' \qquad L \blacktriangleright_{\mathsf{ls}} L' \qquad d = d'}{(t, C, L, T, d) \blacktriangleright_{\mathsf{ls}} (u, D, L', T', d')}\ \mathsf{state_{ls}}$$

$$\frac{C \to_{\mathsf{ls},t} D \qquad T \blacktriangleright_{\mathsf{ls}} T' \qquad L \blacktriangleright_{\mathsf{ls}} L' \qquad d = d'}{(t, C, L \cdot l, T, d) \blacktriangleright_{\mathsf{ls}} (t, D, L', T', d')}\ \mathsf{state2_{ls}}$$

Note that $\blacktriangleright_{\mathsf{ls}}$ contains all pairs $((\underline{t}, \langle\cdot\rangle, \epsilon, \bullet^k), (\underline{u}, \langle\cdot\rangle, \epsilon, \bullet^k))$, where $t \to_{\mathsf{ls}} u$, *i.e.* all the initial states containing a ls-redex and its reduct.

Improvement for $\to_{\mathsf{gc}}$. The candidate improvement $\blacktriangleright_{\mathsf{gc}}$ induced by $\to_{\mathsf{gc}}$ requires an extension of $\to_{\mathsf{gc}}$ with a rule on contexts similar to the parametric one for $\to_{\mathsf{ls}}$. Let $t[x{\leftarrow}u] \to_{\mathsf{gc}} t$ and consider:

$$\begin{tikzcd}
(\underline{t[x{\leftarrow}u]}, \langle\cdot\rangle, \epsilon, \epsilon) \arrow[r, phantom, "\blacktriangleright_{\mathsf{gc}}"] \arrow[d] & (\underline{t}, \langle\cdot\rangle, \epsilon, \epsilon) \\
(\underline{t}, \langle\cdot\rangle[x{\leftarrow}u], \epsilon, \epsilon) &
\end{tikzcd}$$

To close the diagram, we extend the definition of $\to_{\mathsf{gc}}$ to context with the following parametric rule (closed by head contexts):

$$C[x{\leftarrow}u] \quad \mapsto_{\mathsf{gc},t} \quad C \qquad \text{if } x \notin \mathsf{fv}(t) \cup \mathsf{fv}(C).$$

Definition 8.3. The binary relation $\blacktriangleright_{\mathsf{gc}}$ is defined as follows.

$$\frac{t \to_{\mathsf{gc}} u}{(t, H) \blacktriangleright_{\mathsf{gc}} (u, H)}\ \mathsf{rdx_{gc}} \qquad \frac{C \to_{\mathsf{gc}} D}{(t, C) \blacktriangleright_{\mathsf{gc}} (t, D)}\ \mathsf{ctx_{gc}}$$

$$\frac{}{\epsilon \blacktriangleright_{\mathsf{gc}} \epsilon}\ \mathsf{tok1_{gc}} \qquad \frac{C \to_{\mathsf{gc},t} D}{(t, C) \blacktriangleright_{\mathsf{gc}} (t, D)}\ \mathsf{ctx2_{gc}}$$

$$\frac{T \blacktriangleright_{\mathsf{gc}} T'}{\bullet \cdot T \blacktriangleright_{\mathsf{gc}} \bullet \cdot T'}\ \mathsf{tok2_{gc}} \qquad \frac{l \blacktriangleright_{\mathsf{gc}} l' \qquad \Gamma \blacktriangleright_{\mathsf{gc}} \Gamma'}{l \cdot \Gamma \blacktriangleright_{\mathsf{gc}} l' \cdot \Gamma'}\ \mathsf{tok3_{gc}}$$

$$\frac{(x, C) \blacktriangleright_{\mathsf{gc}} (x, D) \qquad L \blacktriangleright_{\mathsf{gc}} L'}{(x, C, L) \blacktriangleright_{\mathsf{gc}} (x, D, L')}\ \mathsf{pos_{gc}}$$

$$\frac{(t, C) \blacktriangleright_{\mathsf{gc}} (u, D) \qquad T \blacktriangleright_{\mathsf{gc}} T' \qquad L \blacktriangleright_{\mathsf{gc}} L' \qquad d = d'}{(t, C, L, T, d) \blacktriangleright_{\mathsf{gc}} (u, D, L', T', d')}\ \mathsf{state_{gc}}$$

The proof of the next theorem is a tedious easy check of diagrams.

Theorem 8.4. *$\blacktriangleright_{\mathsf{ls}}$, $\blacktriangleright_{\mathsf{dB}}$ and $\blacktriangleright_{\mathsf{gc}}$ are improvements.*

9 SOUNDNESS AND ADEQUACY, PROVED

Here we use the improvements of the previous sections to prove soundness and adequacy. Consider $\blacktriangleright = \blacktriangleright_{\mathsf{dB}} \cup \blacktriangleright_{\mathsf{ls}} \cup \blacktriangleright_{\mathsf{gc}}$, that is an improvement because its components are. Consequently, if $t \to_{\mathsf{lh}} u$, then the λ-IAM run on u improves the one on t, that is, $s_{t,k} = (\underline{t}, \langle\cdot\rangle, \epsilon, \bullet^k) \ \blacktriangleright \ (\underline{u}, \langle\cdot\rangle, \epsilon, \bullet^k) = s_{u,k}$.

Improvements transfer more than termination/divergence along $\to_{\mathsf{lh}}$. They also give bisimilar, structurally equivalent tapes, proving the invariance of the semantics, that is, soundness.

Theorem 9.1 (Soundness). *If $t \to_{\mathsf{lh}} u$, then $[\![t]\!]_k = [\![u]\!]_k$ for each $k \geq 0$.*

[11] Γ is a meta-variable that stands either for a log L or for a tape T.

Proof. Since $t \to_{\mathsf{lh}} u$, then $s = (\underline{t}, \langle\cdot\rangle, \epsilon, \bullet^k) \blacktriangleright (\underline{u}, \langle\cdot\rangle, \epsilon, \bullet^k) = q$ by the results about improvements (Theorem 8.4). Since improvements transfer termination/divergence (Prop. 1), we have $[\![t]\!]_k = \bot$ iff $[\![u]\!]_k = \bot$. If $[\![t]\!]_k \neq \bot$ let s' be the final state of s. Since $\blacktriangleright$ is an improvement, there is a final state $q' = (r', C', L', T', d)$ such that $q \to^*_{\lambda\mathrm{IAM}} q'$ and $s' \blacktriangleright q'$. Cases of s':

- $s' = (\underline{\lambda x.w}, C, L, \epsilon)$. Since $s' \blacktriangleright q'$, either $\lambda x.w \to_{\mathsf{lh}} r'$, and thus $r' = \lambda x.w'$ or $C \to_{\mathsf{lh}} C'$ and thus $r' = \lambda x.w$. Then, since $\epsilon \blacktriangleright T'$, also $T' = \epsilon$ and thus $[\![t]\!]_k = [\![u]\!]_k = \Downarrow$.
- $s' = (t, \underline{\langle\cdot\rangle}, \epsilon, \bullet^m \cdot l \cdot \bullet^n)$. Then, since $s' \blacktriangleright q'$, $C' = \langle\cdot\rangle$, because the hole cannot $\to_{\mathsf{lh}}$-reduce. Moreover, the structure of the tape is preserved by $\blacktriangleright$, in particular by its definition also $T' = \bullet^m \cdot l' \cdot \bullet^n$ and thus $[\![t]\!]_k = [\![u]\!]_k = \langle m, n\rangle$.
- $s' = (\underline{x}, C, L, \bullet^m)$. Since a variable cannot $\to_{\mathsf{lh}}$-reduce, also $r' = x$. Then, since the structure of the tape is preserved by $\blacktriangleright$, also $T' = \bullet^m$ and thus $[\![t]\!]_k = [\![u]\!]_k = \langle x, m\rangle$. □

Adequacy. Adequacy is the fact that $[\![t]\!]$ is successful if and only if $\to_{\mathsf{lh}}$ terminates. We prove the two directions separately.

Direction λ-IAM to $\to_{\mathsf{lh}}$. The *only if* direction of the statement is easy to prove. Since $[\![t]\!]_k$ is invariant by $\to_{\mathsf{lh}}$ (soundness) and $\to_{\mathsf{lh}}$ terminates on t, we can as well assume that t is normal. The rest is given by the following proposition.

Proposition 9.2 (Reading the head variable on $\to_{\mathsf{lh}}$-normal forms). *Let $t = \lambda x_1 \ldots \lambda x_n.yu_1\ldots u_l$ be a head (linear) normal form, where $n, l \geq 0$. If $y = x_m$ where $1 \leq m \leq n$, then $[\![t]\!]_{n+k} = \langle m-1, l+k\rangle$, otherwise, if y is free, then $[\![t]\!]_{n+k} = \langle y, l+k\rangle$. Moreover, if $n \geq 1$ and $0 \leq k \leq n-1$, then $[\![t]\!]_k = \Downarrow$.*

Proof. We proceed computing $[\![t]\!]_n$ explicitly. We have:

$$\begin{array}{llll} (\underline{t}, \langle\cdot\rangle, \epsilon, \bullet^n) & \to^n_{\bullet 2} & (\underline{yu_1\ldots u_l}, \lambda x_1 \ldots \lambda x_n.\langle\cdot\rangle, \epsilon, \epsilon) \\ & \to^l_{\bullet 1} & (\underline{y}, \lambda x_1 \ldots \lambda x_n.\langle\cdot\rangle u_1\ldots u_l, \epsilon, \bullet^l) \end{array}$$

If y is free, the λ-IAM stops and $[\![t]\!]_n = \langle y, l\rangle$. By lifting (Lemma 5.6), one has that $[\![t]\!]_{n+k} = \langle y, l+k\rangle$. Otherwise, if y is bound by a λ-abstraction, *i.e.* $y = x_m$ for $1 \leq m \leq n$, the computation continues.

$$\begin{array}{ll} (\underline{t}, \langle\cdot\rangle, \epsilon, \bullet^n) & \to^{n+l}_{\lambda\mathrm{IAM}} \\ (\underline{x_m}, \lambda x_1 \ldots \lambda x_n.\langle\cdot\rangle u_1\ldots u_l, \epsilon, \bullet^l) & \to_{\mathrm{var}} \\ (\lambda x_m \ldots \lambda x_n.x_m u_1\ldots u_l, \underline{\lambda x_1 \ldots \lambda x_{m-1}.\langle\cdot\rangle}, l \cdot \bullet^l, \epsilon) & \to^{m-1}_{\bullet 4} \\ (u, \underline{\langle\cdot\rangle}, \epsilon, \bullet^{m-1} \cdot l \cdot \bullet^l). & \end{array}$$

where $l = (x_m, \lambda x_1 \ldots \lambda x_n.\langle\cdot\rangle u_1\ldots u_l, \epsilon)$. Then $[\![t]\!]_n = \langle m-1, l\rangle$. By lifting (Lemma 5.6), one has that $[\![t]\!]_{n+k} = \langle m-1, l+k\rangle$.

Moreover, please note that if $n \geq 1$ and $0 \leq k \leq n-1$, we have:

$$(\underline{t}, \langle\cdot\rangle, \epsilon, \bullet^k) \to^k_{\bullet 2} (\underline{\lambda x_k \ldots \lambda x_n.yu_1\ldots u_l}, \lambda x_1 \ldots \lambda x_{k-1}.\langle\cdot\rangle, \epsilon, \epsilon)$$

and thus $[\![t]\!]_k = \Downarrow$. □

Direction $\to_{\mathsf{lh}}$ to λ-IAM. The proof of the *if* direction of the adequacy theorem is by contra-position: if the $\to_{\mathsf{lh}}$ diverges on t then no run of the λ-IAM on t ends in a successful state.

The proof is obtained via a quantitative analysis of the improvements, showing that the length of runs *strictly* decreases along $\to_{\mathsf{lh}}$. Note indeed that improvements guarantee only that the length of runs does not increase. To prove that it actually decreases one needs an additional *global* analysis of runs—improvements only deal with *local* bisimulation diagrams. On proof nets, this decreasing property correspond to the standard fact that IAM paths passing through a cut have shorter residuals after that cut.

We recall that we write $|t|_k$ for the length of the run $(\underline{t}, \langle\cdot\rangle, \epsilon, \bullet^k)$, with the convention that $|t|_k = \infty$ if the machine diverges.

Lemma 9.3 (The length of terminating runs strictly decreases along $\to_{\mathsf{lh}}$). *Let $t \to_{\mathsf{lh}} u$ and $|t|_k \neq \infty$. There exists $k \geq 0$ such that $|t|_h > |u|_h$ for each $h \geq k$.*

Proof. We treat the case of $t \to_{\mathrm{dB}} u$, the others are obtained via similar diagrams. If t has a $\to_{\mathrm{dB}}$-redex then it has the shape $t = H\langle\langle\lambda x.r\rangle Sw\rangle$ and u is in the form $u = H\langle\langle r[x{\leftarrow}w]\rangle S\rangle$. By induction on the structure of H one can prove that there exist $k, n \geq 0$ such that $(\underline{t}, \langle\cdot\rangle, \epsilon, \bullet^k) \to^n_{\lambda\mathrm{IAM}} (\underline{\langle\lambda x.r\rangle Sw}, H, \epsilon, \epsilon)$ and $(\underline{u}, \langle\cdot\rangle, \epsilon, \bullet^k) \to^n_{\lambda\mathrm{IAM}} (\underline{\langle r[x{\leftarrow}w]\rangle S}, H, \epsilon, \epsilon)$. Given such n and k by the lifting lemma (Lemma 5.6) also the following holds: for any $j \geq 0$, $(\underline{t}, \langle\cdot\rangle, \epsilon, \bullet^j \cdot \bullet^k) \to^n_{\lambda\mathrm{IAM}} (\underline{\langle\lambda x.r\rangle Sw}, H, \epsilon, \bullet^j)$ and $(\underline{u}, \langle\cdot\rangle, \epsilon, \bullet^j \cdot \bullet^k) \to^n_{\lambda\mathrm{IAM}} (\underline{\langle r[x{\leftarrow}w]\rangle S}, H, \epsilon, \bullet^j)$. Moreover, by definition of the improvement $\blacktriangleright_{\mathrm{dB}}$ we have the following diagram.

$$\begin{tikzcd}
(\underline{H\langle\langle\lambda x.r\rangle Sw\rangle}, \langle\cdot\rangle, \epsilon, \bullet^j \cdot \bullet^k) \arrow[d, "n"] \arrow[r, phantom, "\blacktriangleright_{\mathrm{dB}}"] & (\underline{H\langle\langle r[x{\leftarrow}w]\rangle S\rangle}, \langle\cdot\rangle, \epsilon, \bullet^j \cdot \bullet^k) \arrow[d, "n"] \\
(\underline{\langle\lambda x.r\rangle Sw}, H, \epsilon, \bullet^j) \arrow[d] \arrow[r, phantom, "\blacktriangleright_{\mathrm{dB}}"] & (\underline{\langle r[x{\leftarrow}w]\rangle S}, H, \epsilon, \bullet^j) \arrow[dd, "|S|"] \\
(\underline{\langle\lambda x.r\rangle S}, H\langle\langle\cdot\rangle w\rangle, \epsilon, \bullet \cdot \bullet^j) \arrow[d, "|S|"] & \\
(\underline{\lambda x.r}, H\langle Sw\rangle, \epsilon, \bullet \cdot \bullet^j) \arrow[d] & (\underline{r[x{\leftarrow}w]}, H\langle\langle\cdot\rangle L\rangle, \epsilon, \bullet^j) \arrow[d] \\
s_1 = (\underline{r}, H\langle\langle\lambda x.\langle\cdot\rangle\rangle Sw\rangle, \epsilon, \bullet^j) \arrow[r, phantom, "\blacktriangleright_{\mathrm{dB}}"] & (\underline{r}, H\langle\langle\langle\cdot\rangle[x{\leftarrow}w]\rangle S\rangle, \epsilon, \bullet^j) = s_2
\end{tikzcd}$$

From $s_1 \blacktriangleright_{\mathrm{dB}} s_2$, the hypothesis $|t|_k \neq \infty$, and the properties of improvements (Lemma 2), we obtain $|s_1| \geq |s_2|$. Then, by setting $h := k+j$, we have $|t|_h = n+1+|S|+1+|s_1| > n+|S|+1+|s_2| = |u|_h$. □

Using the lemma, we prove the *if* direction of adequacy, that then follows.

Proposition 9.4 ($\to_{\mathsf{lh}}$-divergence implies that the λ-IAM never succeeds). *Let t be a $\to_{\mathsf{lh}}$-divergent LSC term. There is no $k \geq 0$ such that $[\![t]\!]_k$ is successful.*

Proof. By contradiction, suppose that there exists k such that $[\![t]\!]_k$ is successful. Then by soundness $|t|_k = n \in \mathbb{N}$ and it ends on a successful state. By monotonicity of runs (Lemma 5.7), $|t|_k = |t|_h = n$ for every $h > k$. Since $t \to_{\mathsf{lh}}$-divergent, then there exists an infinite reduction sequence $\rho : t = t_0 \to_{\mathsf{lh}} t_1 \to_{\mathsf{lh}} t_2 \to_{\mathsf{lh}} \cdots t_k \to_{\mathsf{lh}} \cdots$. Since the length of terminating runs strictly decreases along $\to_{\mathsf{lh}}$ for sufficiently long inputs (Lemma 9.3), for each $i \in \mathbb{N}$ if $t_i \to_{\mathsf{lh}} t_{i+1}$ then there exists k_i such that $|t_i|_{k_i} > |t_{i+1}|_{k_i}$. Now, consider $h = \max\{k_0, \ldots, k_n, k_{n+1}\}$. We have that $|t_j|_h > |t_{j+1}|_h$ for every $j \in \{0, 1, \ldots, n+1\}$. Then $|t_0|_h \geq |t_{n+1}|_h + n + 1$. Since the length of runs is non-negative, we obtain that $|t_0|_h \geq n+1$, which is absurd because $h \geq k_0$ and so $|t_0|_h = n$. □

Theorem 9.5 (Adequacy). *Let t be a LSC term. Then t has $\to_{\mathsf{lh}}$-normal form if and only if there exists $h \geq 0$ such that for all $k \geq h$ $[\![t]\!]_k$ is successful. Moreover, t has weak $\to_{\mathsf{lh}}$-normal form if and only if either $[\![t]\!]_0 = \Downarrow$ or $[\![t]\!]_0 = \langle x, n\rangle$ for some $x \in \mathcal{V}, n \geq 0$.*

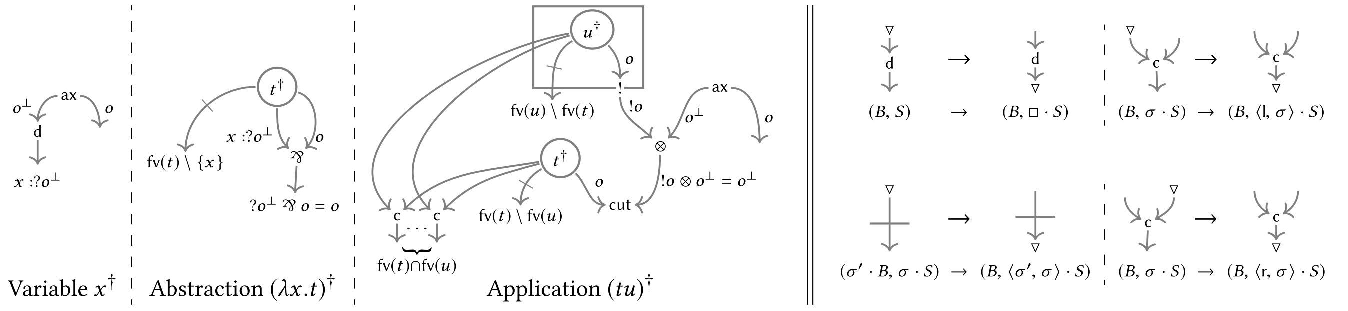

Variable $x^\dagger$
Abstraction $(\lambda x.t)^\dagger$
Application $(tu)^\dagger$
ax
$o^\perp$
o
d
$x :?o^\perp$
$t^\dagger$
$\mathsf{fv}(t) \setminus \{x\}$
$?o^\perp \parr o = o$
$u^\dagger$
$\mathsf{fv}(u) \setminus \mathsf{fv}(t)$
$!o$
$\otimes$
$!o \otimes o^\perp = o^\perp$
cut
$\mathsf{fv}(t) \setminus \mathsf{fv}(u)$
$\mathsf{fv}(t) \cap \mathsf{fv}(u)$
$(B, S) \to (B, \square \cdot S)$
$(B, \sigma \cdot S) \to (B, \langle \mathsf{l}, \sigma \rangle \cdot S)$
$(\sigma' \cdot B, \sigma \cdot S) \to (B, \langle \sigma', \sigma \rangle \cdot S)$
$(B, \sigma \cdot S) \to (B, \langle \mathsf{r}, \sigma \rangle \cdot S)$

Figure 6: On the left, the call-by-name translation $(\cdot)^\dagger$ of the λ-calculus into linear logic proof nets. On the right, transition rules of the proof nets presentation of the IAM related to exponential signatures.

10 COMPARISON WITH PROOF NETS

Here we sketch how the λ-IAM relates to the original presentation based on linear logic proof nets, due to Mackie and Danos & Regnier [25, 27, 41], the IAM. For lack of space, we avoid defining proof nets and related concepts, and focus only on the key points.

The λ-IAM corresponds to the IAM on proof nets representing λ-terms according to the call-by-name translation $(\cdot)^\dagger$ in Fig. 6[12], and considering only paths from the distinguished conclusion of the net, as in [25] (while [27, 41] use the call-by-value translation, and [27] considers paths starting on whatever conclusions).

There is a bisimulation between the λ-IAM and such a restricted IAM, which is not strong because two λ-IAM transitions rather are *macros*, packing together whole sequences of transitions in their presentation. Namely, transition $\to_{\mathsf{var}}$ short-circuits the path between a variable x and its abstraction $\lambda x.C_n\langle x\rangle$. In proof nets, this path traverses a dereliction, exactly n auxiliary doors, possibly a contraction tree, and ends on the $\parr$ representing the abstraction. The dual transition $\to_{\mathsf{bt2}}$ does the reverse job, corresponding to the reversed path. Aside the different notations and the *macrification*, our transitions correspond exactly to the actions attached to proof net edges presented in [27][13], as we explain next.

In the proof nets presentation the token is given by two stacks, called *boxes stack B* and *balancing stack S*, corresponding exactly to our log L and tape T, respectively. They are formed by sequences of multiplicative constants p (corresponding to our $\bullet$) and by *exponential signatures σ*. They are defined by the following grammar[14].

$$\begin{array}{rrcl}
\textsc{Balancing stacks} & S & ::= & \epsilon \mid \mathsf{p} \cdot S \mid \sigma \cdot S \\
\textsc{Boxes stacks} & B & ::= & \epsilon \mid \sigma \cdot B \\
\textsc{Exp. signatures} & \sigma, \sigma' & ::= & \square \mid \langle \sigma, \sigma' \rangle \mid \langle \mathsf{l}, \sigma \rangle \mid \langle \mathsf{r}, \sigma \rangle
\end{array}$$

Intuitively, exponential signatures are binary trees with $\square$, l or r as leaves, where l and r denote the left/right premise of a contraction. Fig. 6 shows the IAM transitions concerning exponential signatures that are the relevant difference with respect to the λ-IAM.

[12]The translation uses a recursive type $o = ?o^\perp \parr o$ in order to be able to represent untyped terms of the λ-calculus—this is standard. Every net has a unique conclusion labeled with o, which is the *output*, and all the other conclusions have type $?o^\perp$ and are labeled with a free variable of the term. In the abstraction case $\lambda x.t$, if $x \notin \mathsf{fv}(t)$ then a weakening is added to represent that variable.

[13]We refer to [27] rather than [25] because in [25] the definition is only sketched, while [27] is more accurate.

[14]With respect to [27]: for clarity, we use symbols l and r instead of p′ and q′, and we omit q, dual of p, as in the call-by-name translation it is always and only next to exponential signatures, which then subsume it.

To explain how $\to_{\mathsf{var}}$ is simulated by the IAM, let's recall it:

$$(\underline{x}, C\langle \lambda x.D_n\rangle, L_n \cdot L, T) \to_{\mathsf{var}} (\lambda x.D_n\langle x\rangle, \underline{C}, L, (x, \lambda x.D_n, L_n) \cdot T).$$

The IAM does the same, just in more steps and with another syntax. Consider a token $(B_n \cdot B, S)$ approaching a variable x that is n boxes deeper than its binder $\lambda x.D_n\langle x\rangle$. Variables are translated as dereliction links and thus we have: $(B_n \cdot B, S) \to (B_n \cdot B, \square \cdot S)$.

Then, the token travels until the binder of x is found (a $\parr$ in the proof net translation of the term), *i.e.* it traverses exactly n boxes always exiting from the auxiliary doors. Moreover, for every such box a contraction could be encountered. Let's first suppose that x is used linearly, so that no contractions are encountered. Then the token rewrites in the following way, traversing n auxiliary doors.

$$\begin{array}{ll}
(\sigma_1 \cdot B_{n-1} \cdot B, \square \cdot S) & \to (\sigma_2 \cdot B_{n-2} \cdot B, \langle \sigma_1, \square \rangle \cdot S) \\
& \to (\sigma_3 \cdot B_{n-3} \cdot B, \langle \sigma_2, \langle \sigma_1, \square \rangle\rangle \cdot S) \\
& \to \cdots \to (B, \langle \sigma_n, \langle \cdots \langle \sigma_1, \square \rangle \cdots \rangle\rangle \cdot S).
\end{array}$$

Note the perfect matching between the two formulations: in both cases the first n logged positions/signatures in the log/boxes stack are removed from it and, once wrapped in a single logged position/signature, then put on the tape/balancing stack. In presence of contractions the exponential signature $\langle \sigma_n, \langle \cdots \langle \sigma_1, \square \rangle \cdots \rangle\rangle$ is interleaved by l and r leaves. These symbols represent nothing more than a binary code used to traverse the contraction tree of x. In the λ-IAM, the same information is represented, more compactly, by specifying the variable occurrence via its position inside its binder.

11 CONCLUSIONS

This paper presents a direct proof of the implementation theorem for Mackie and Danos & Regnier's Interaction Abstract Machine, building over a natural notion of bisimulation and avoiding detours via game semantics. Additionally, it (re)formulates the machine directly on λ-terms, making it conceptually closer to traditional abstract machines, and more apt to formalizations in proof assistants.

Our work opens the way to a fine analysis of the complexity of the implementation of the λ-calculus, in particular regarding the space-time trade-off, by comparing the λ-IAM, that the literature suggests being tuned for space-efficiency, to traditional environment machines that are instead tuned for time-efficiency.

REFERENCES

[1] Samson Abramsky, Esfandiar Haghverdi, and Philip Scott. 2002. Geometry of Interaction and linear combinatory algebras. *Mathematical Structures in Computer Science* 12, 5 (2002), 625–665.

[2] Beniamino Accattoli. 2012. An Abstract Factorization Theorem for Explicit Substitutions. In *Proceedings of RTA'12 (LIPIcs)*, Vol. 15. 6–21.

[3] Beniamino Accattoli. 2018. (In)Efficiency and Reasonable Cost Models. *Electr. Notes Theor. Comput. Sci.* 338 (2018), 23–43.

[4] Beniamino Accattoli. 2018. Proof Nets and the Linear Substitution Calculus. In *Proceedings of the 15th ICTAC*. 37–61.

[5] Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. 2014. Distilling abstract machines. In *Proceedings of ICFP 2014*. 363–376.

[6] Beniamino Accattoli and Bruno Barras. 2017. Environments and the complexity of abstract machines. In *Proceedings of the 19th PPDP*. 4–16.

[7] Beniamino Accattoli, Andrea Condoluci, Giulio Guerrieri, and Claudio Sacerdoti Coen. 2019. Crumbling Abstract Machines. In *Proceedings of the 21st PPDP*. 4:1–4:15.

[8] Beniamino Accattoli and Ugo Dal Lago. 2016. (Leftmost-Outermost) Beta Reduction is Invariant, Indeed. *Logical Methods in Computer Science* 12, 1 (2016).

[9] Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. 2020. Tight typings and split bounds, fully developed. *J. Funct. Program.* 30 (2020), e14.

[10] Beniamino Accattoli and Giulio Guerrieri. 2017. Implementing Open Call-by-Value. In *FSEN 2017, Revised Selected Papers*. 1–19.

[11] Beniamino Accattoli and Delia Kesner. 2010. The Structural λ-Calculus. In *Proceedings of CSL'10*. 381–395.

[12] Beniamino Accattoli and Ugo Dal Lago. 2012. On the Invariance of the Unitary Cost Model for Head Reduction. In *Proceedings of the 23rd RTA*. 22–37.

[13] Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. 2020. The Machinery of Interaction (Long Version). https://arxiv.org/abs/2002.05649.

[14] Beniamino Accattoli and Claudio Sacerdoti Coen. 2015. On the Relative Usefulness of Fireballs. In *Proceedings of the 30th LICS*. 141–155.

[15] Andrea Asperti, Vincent Danos, Cosimo Laneve, and Laurent Regnier. 1994. Paths in the lambda-calculus. In *Proceedings of LICS '94*. 426–436.

[16] Guy E. Blelloch and John Greiner. 1995. Parallelism in Sequential Functional Languages. In *FPCA*. 226–237.

[17] Pierre-Louis Curien and Hugo Herbelin. 1998. Computing with Abstract Böhm Trees. In *Proceedings of the 3rd FLOPS*.

[18] Pierre-Louis Curien and Hugo Herbelin. 2007. Abstract machines for dialogue games. (2007). arXiv:0706.2544 http://arxiv.org/abs/0706.2544

[19] 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.

[20] 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.

[21] Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. 2017. The geometry of parallelism: classical, probabilistic, and quantum effects. In *Proceedings of the 44th POPL*. 833–845.

[22] Ugo Dal Lago and Simone Martini. 2008. The weak lambda calculus as a reasonable machine. *Theor. Comput. Sci.* 398, 1-3 (2008), 32–50.

[23] Ugo Dal Lago and Ulrich Schöpp. 2016. Computation by interaction for space-bounded functional programming. *Information and Computation* 248 (2016), 150–194.

[24] 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.

[25] Vincent Danos, Hugo Herbelin, and Laurent Regnier. 1996. Game Semantics & Abstract Machines. In *Proceedings of the 11th LICS*. 394–405.

[26] Vincent Danos and Laurent Regnier. 1993. Local and asynchronous beta-reduction (an analysis of Girard's execution formula). In *Proceedings of the 8th LICS*. 296–306.

[27] Vincent Danos and Laurent Regnier. 1999. Reversible, irreversible and optimal lambda-machines. *Theoretical Computer Science* 227, 1 (1999), 79–97.

[28] Vincent Danos and Laurent Regnier. 2004. *Head Linear Reduction.* Technical Report.

[29] Matthias Felleisen and Daniel P. Friedman. 1986. Control operators, the SECD-machine, and the lambda-calculus. In *3rd Working Conference on the Formal Description of Programming Concepts*.

[30] Maribel Fernández and Ian Mackie. 2002. Call-by-Value lambda-Graph Rewriting Without Rewriting. In *Proceedings of the 1st ICGT*. 75–89.

[31] Yannick Forster, Fabian Kunze, and Marc Roth. 2019. The Weak Call-by-Value λ-Calculus is Reasonable for Both Time and Space. *PACMPL* 4, POPL, Article 27 (2019), 23 pages.

[32] Dan R. Ghica. 2007. Geometry of Synthesis: A Structured Approach to VLSI Design. In *Proceedings of the 34th POPL*. 363–375.

[33] Jean-Yves Girard. 1989. Geometry of Interaction 1: 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.

[34] Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. 1992. The Geometry of Optimal Lambda Reduction. In *Proceedings of the 19th POPL*. 15–26.

[35] 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.

[36] André Joyal, Ross Street, and Dominic Verity. 1996. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society* 119, 3 (1996), 447–468.

[37] Delia Kesner and Shane Ó Conchúir. 2008. *Milner's Lambda Calculus with Partial Substitutions.* Technical Report. Paris 7 University. http://www.pps.univ-paris-diderot.fr/~kesner/papers/shortpartial.pdf.

[38] Jean-Louis Krivine. 2007. A Call-by-name Lambda-calculus Machine. *Higher Order Symbol. Comput.* 20, 3 (2007), 199–207.

[39] Peter John Landin. 1965. Correspondence Between ALGOL 60 and Church's Lambda-notation: Part I. *Commun. ACM* 8, 2 (1965), 89–101.

[40] Olivier Laurent. 2001. A Token Machine for Full Geometry of Interaction. In *Proceedings of the 5th TLCA*. 283–297.

[41] Ian Mackie. 1995. The Geometry of Interaction Machine. In *Proceedings of the 22nd POPL*. 198–208.

[42] Ian Mackie. 2017. A Geometry of Interaction Machine for Gödel's System T. In *Proceedings of the 24th WoLLIC*. 229–241.

[43] Gianfranco Mascari and Marco Pedicini. 1994. Head Linear Reduction and Pure Proof Net Extraction. *Theoretical Computer Science* 135, 1 (1994), 111–137.

[44] Damiano Mazza. 2015. Simple Parsimonious Types and Logarithmic Space. In *Proceedings of the 24th CSL*. 24–40.

[45] Damiano Mazza and Kazushige Terui. 2015. Parsimonious Types and Non-uniform Computation. In *Proceedings of the 42nd ICALP*. 350–361.

[46] Robin Milner. 2007. Local Bigraphs and Confluence: Two Conjectures. *Electronic Notes in Theoretical Computer Science* 175, 3 (2007), 65–73.

[47] Koko Muroya and Dan R. Ghica. 2017. The Dynamic Geometry of Interaction Machine: A Call-by-Need Graph Rewriter. In *Proceedings of the 26th CSL*. 32:1–32:15.

[48] Laurent Regnier. 1994. Une équivalence sur les lambda- termes. *Theoretical Computer Science* 126, 2 (1994), 281 – 292.

[49] David Sands. 1996. Total Correctness by Local Improvement in the Transformation of Functional Programs. *ACM Trans. Program. Lang. Syst.* 18, 2 (1996), 175–234.

[50] David Sands, Jörgen Gustavsson, and Andrew Moran. 2002. Lambda Calculi and Linear Speedups. In *The Essence of Computation*. 60–84.

[51] Ulrich Schopp. 2007. Stratified Bounded Affine Logic for Logarithmic Space. In *Proceedings of LICS 2007*. 411–420.

[52] Ulrich Schöpp. 2014. On the Relation of Interaction Semantics to Continuations and Defunctionalization. *Logical Methods in Computer Science* 10, 4 (2014).

[53] Ulrich Schöpp. 2015. From Call-by-Value to Interaction by Typed Closure Conversion. In *Proc. of PPDP 2015 (LNCS)*, Vol. 9458. Springer, 251–270.