The Geometry of Abstract Machines

Beniamino Accattoli Ugo Dal Lago Gabriele Vanoni

1 Introduction

Geometry of Interaction (GoI) [Gir89] is a semantic framework which forms a model of proof-nets [Gir87] and thus the λ-calculus, via the so-called Girard's translation. It has been initially formulated by Girard in terms of operator algebras. Other formulations were later introduced, e.g. domain-theoretic [AJ94] or categorical [AHS02]. The most concrete formulations of GoI, like context semantics [GAL92] or the interaction abstract machine [DR99], in particular, are concrete enough to induce compilation schemes towards concrete languages and architectures [Mac95, Ghi07].

We give in this paper an abstract machine inspired by the GoI context semantics directly at the level of the λ-calculus, avoiding the explicit use of proof-nets. This way, reasoning about the machine can be done in a purely inductive way, following the structure of terms. Moreover, we are interested in comparing from the complexity point of view this Interaction Abstract Machine (IAM_λ) and its possible optimizations with classical call-by-name abstract machines such as the Krivine Abstract Machine (KAM) [Kri07], following the pioneering work of Danos and Regnier [DR99]. Our presentation indeed turns GoI into a form closer to ordinary machines.

2 The Interaction Abstract Machine

This section is devoted to defining the IAM_λ for the untyped λ-calculus [Bar84], after giving some necessary preliminaries.

Definition 2.1. *Terms of the λ-calculus Λ are defined by the following grammar.*

$$t, u, r ::= x \mid \lambda x.t \mid tu.$$

Definition 2.2. Leveled contexts *are defined by*

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

Intuitively, a context C_n is one in which the hole lies inside exactly n boxes. A context in the form C_n is said to be n-ary and $\mathcal{C}_n$ is the set of all n-ary contexts, for every $n \in \mathbb{N}$. $\mathcal{C} = \bigcup_{n\in\mathbb{N}} \mathcal{C}_n$ is simply the set of all contexts. A context of level 0 is also called a *head context*. The level of a context will be omitted when not relevant to the discussion, however note that any ordinary context writes in a unique way as a levelled context, so that the omission does not really omit information.

A *position* (of level n) in a term t is a pair (u, C_n) s.t. $C_n\langle u\rangle = t$.

We need also an enriched notion of position, called *signature*, in which an n-ary context comes together with a list of n signatures, one for every box in which the hole is contained.

Definition 2.3. *The set of* signatures $\mathcal{E}$ *is the smallest set satisfying:*

$$(x, C\langle\lambda x.C_n\rangle, e_1 \cdot \ldots \cdot e_n) \in \mathcal{E} \text{ if } e_1, \ldots, e_n \in \mathcal{E}.$$

In order to define the states of the machine we also need a constant symbol p, whose is to keep track of the exploration of the term, and two polarities $\mathcal{P} = \{\uparrow, \downarrow\}$, encoding two different operating modalities of the machine. In fact, polarities will be mostly omitted and represented via colors.

Definition 2.4 (IAM$_\lambda$ State)**.** *A state of the* IAM$_\lambda$ *is a quintuple* (t, C, Ψ, Σ, p) *where*

1. t *is a* λ*-term: the* Code Term*;*
2. C *is a context: the* Code Context*;*
3. Ψ *is an element of* $(\{\mathsf{p}\} \cup \mathcal{E})^*$*: the* Input-Output Stack*;*
4. Σ *is an element of* $\mathcal{E}^*$*: the* Signature Stack*;*
5. p *is an element in* $\mathcal{P} = \{\uparrow, \downarrow\}$*: the* Polarity*.*

We call $\mathcal{S}$ the set of all IAM$_\lambda$ states. The code term and code context components form a position, whose role is to keep track of where, in the term under evaluation, the machine is currently executing. Both stacks contain information needed to carry on the computation deterministically. Polarities establish the operating mode of the machine: in mode $\downarrow$ the IAM$_\lambda$ walks down through the structure of the code term, in mode $\uparrow$ the IAM$_\lambda$ walks up through the structure of the code context. We will omit the polarity by colouring the term code in red for $\downarrow$ states and the context code in blue for $\uparrow$ states. We write $\mathbf{e}_n$ for a list of n signatures $e_1 \cdot \ldots \cdot e_n$.

Definition 2.5 (IAM$_\lambda$ Transitions)**.** *The transition rules for* $\downarrow$*-states, whose behavior depends on the code term (first column):*

ut	C	Ψ	Σ	$\longrightarrow_{\downarrow @\mathsf{p}^*}$	u	$C\langle\langle\cdot\rangle t\rangle$	$\mathsf{p}\cdot\Psi$	Σ
$\lambda x.t$	C	$\mathsf{p}\cdot\Psi$	Σ	$\longrightarrow_{\downarrow \overline{\lambda}\mathsf{p}}$	t	$C\langle\lambda x.\langle\cdot\rangle\rangle$	Ψ	Σ
x	$C\langle\lambda x.D_n\rangle$	Ψ	$\mathbf{e}_n\cdot\Sigma$	$\longrightarrow_{\downarrow de^*}$	$\lambda x.D_n\langle x\rangle$	C	$(x, C\langle\lambda x.D_n\rangle, \mathbf{e}_n)\cdot\Psi$	Σ
$\lambda x.t$	C	$(x, C\langle\lambda x.D_n\rangle, \Theta)\cdot\Psi$	Σ	$\longrightarrow_{\downarrow \overline{d}\mathsf{e}_2}$	x	$C\langle\lambda x.D_n\rangle$	Ψ	$\Theta\cdot\Sigma$

The transitions for $\uparrow$*-states, whose behavior depends on the context (second column):*

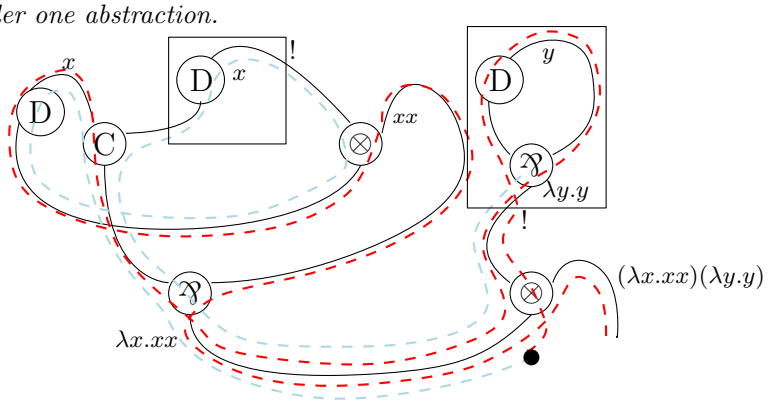
u	$C\langle\langle\cdot\rangle t\rangle$	$e\cdot\Psi$	Σ	$\longrightarrow_{\uparrow \overline{!}\mathsf{e}}$	t	$C\langle u\langle\cdot\rangle\rangle$	Ψ	$e\cdot\Sigma$
u	$C\langle\langle\cdot\rangle t\rangle$	$\mathsf{p}\cdot\Psi$	Σ	$\longrightarrow_{\uparrow \overline{@}\mathsf{p}_2}$	ut	C	Ψ	Σ
t	$C\langle u\langle\cdot\rangle\rangle$	Ψ	$e\cdot\Sigma$	$\longrightarrow_{\uparrow !\mathsf{e}_2^*}$	u	$C\langle\langle\cdot\rangle t\rangle$	$e\cdot\Psi$	Σ
t	$C\langle\lambda x.\langle\cdot\rangle\rangle$	Ψ	Σ	$\longrightarrow_{\uparrow \lambda\mathsf{p}_2^*}$	$\lambda x.t$	C	$\mathsf{p}\cdot\Psi$	Σ

The idea behind these transitions is to mimic the behaviour of token machines built on top of proof-nets via context semantics. In each state $s = (t, C_n, \Psi, \Sigma, p)$, the position (t, C) represents the position of the token on the net, and indeed the term $C\langle t\rangle$ obtained by plugging the Code Term into the Code Context is invariant during a run. The two stacks Ψ and Σ correspond to those used in [DR99]. The Input-Output Stack Ψ stores the information needed to traverse the term deterministically as long as the token does not interact (enters or exits) with a box, which here is nothing more than an argument of an application. The Signature Stack Σ instead is needed to store information when the token enters a box and to restore it when the token exits, as witnessed by the fact that $|\Sigma| = n$, which is the level of box nesting of t, always holds.

Example 2.6. *We give here an example of a* IAM$_\lambda$ *run for the term* $t = (\lambda x.xx)(\lambda y.y)$*. The initial state is* $(t, \langle\cdot\rangle, \epsilon, \epsilon)$*, equivalent to pushing an empty token from the (unique) conclusion of the corresponding proof-net. Each line in the following table corresponds to a state, and each state evolves to the one below via the rules described above.*

$(\lambda x.xx)(\lambda y.y)$	$\langle\cdot\rangle$	ϵ	ϵ
$\lambda x.xx$	$\langle\cdot\rangle(\lambda y.y)$	p	ϵ
xx	$(\lambda x.\langle\cdot\rangle)(\lambda y.y)$	ϵ	ϵ
x	$(\lambda x.\langle\cdot\rangle x)(\lambda y.y)$	p	ϵ
$\lambda x.xx$	$\langle\cdot\rangle(\lambda y.y)$	$(x, (\lambda x.\langle\cdot\rangle x)(\lambda y.y), \epsilon)\cdot\mathsf{p}$	ϵ
$\lambda y.y$	$(\lambda x.xx)\langle\cdot\rangle$	p	$(x, (\lambda x.\langle\cdot\rangle x)(\lambda y.y), \epsilon)$
y	$(\lambda x.xx)(\lambda y.\langle\cdot\rangle)$	ϵ	$(x, (\lambda x.\langle\cdot\rangle x)(\lambda y.y), \epsilon)$
$\lambda y.y$	$(\lambda x.xx)\langle\cdot\rangle$	$(y, (\lambda x.xx)(\lambda y.\langle\cdot\rangle), \epsilon)$	$(x, (\lambda x.\langle\cdot\rangle x)(\lambda y.y), \epsilon)$
$\lambda x.xx$	$\langle\cdot\rangle(\lambda y.y)$	$(x, (\lambda x.\langle\cdot\rangle x)(\lambda y.y), \epsilon)\cdot(y, (\lambda x.xx)(\lambda y.\langle\cdot\rangle), \epsilon)$	ϵ
x	$(\lambda x.\langle\cdot\rangle x)(\lambda y.y)$	$(y, (\lambda x.xx)(\lambda y.\langle\cdot\rangle), \epsilon)$	ϵ
x	$(\lambda x.x\langle\cdot\rangle)(\lambda y.y)$	ϵ	$(y, (\lambda x.xx)(\lambda y.\langle\cdot\rangle), \epsilon)$
$\lambda x.xx$	$\langle\cdot\rangle(\lambda y.y)$	$(x, (\lambda x.x\langle\cdot\rangle)(\lambda y.y), (y, (\lambda x.xx)(\lambda y.\langle\cdot\rangle), \epsilon))$	ϵ
$\lambda y.y$	$(\lambda x.xx)\langle\cdot\rangle$	ϵ	$(x, (\lambda x.x\langle\cdot\rangle)(\lambda y.y), (y, (\lambda x.xx)(\lambda y.\langle\cdot\rangle), \epsilon))$

Below we have drawn the corresponding proof-net and the path followed by the token if one interprets the graph as an automaton via context semantics. One can see that there is a perfect match between positions in IAM_λ *states and graph nodes traversed by the token in the IAM [DR99]. Both machines are stuck in front of a* λ*-abstraction. Indeed, to evaluate under abstractions, one should feed the machine with a non-empty stack* Ψ*. In particular every* p *in the initial state allows to evaluate under one abstraction.*



A state $s \in \mathcal{S}$ is *terminating* (in s') if $s \to^*_{\mathrm{IAM}_\lambda} s'$ and s' is a final state, i.e. there is no possible transitions from it. What does the IAM_λ compute if the execution terminates? We make it precise with the following Definition.

Definition 2.7 (IAM_λ Semantics)**.** *We define* IAM_λ *semantics* $[\![\cdot]\!]_k : \Lambda \to \mathbb{N} \cup \{\bot\}$*, where* $k \in \mathbb{N}$*, in the following way.*

$$[\![t]\!]_k = \begin{cases} h & \text{if } (t, \langle\cdot\rangle, \mathsf{p}^k, \epsilon) \text{ is terminating in the state } (u, C, \mathsf{p}^h \cdot \Psi, \Sigma, p), \\ \bot & \text{otherwise.} \end{cases}$$

We need to show that this semantics is adequate w.r.t. some observables. First, we argue that the above defined semantics is invariant by head β-reduction $\to_h$, i.e. the closure by head contexts of the β-rule $(\lambda x.t)u \mapsto t\{u/x\}$. The idea to prove this claim is to devise a coinductively defined relation (a sort of bisimulation) between IAM_λ states that preserves both termination and the structure of the Input-Output Stack Ψ. Then, one should just show that two initial states, one fed with a term containing a head-redex and the other one with the corresponding reduct can be put in relation.

Theorem 2.8 (Soundness)**.** *If* $t \to_h u$*, then* $[\![t]\!]_k = [\![u]\!]_k$*.*

Adequacy is then shown w.r.t. a refinement of termination.

Theorem 2.9 (Adequacy)**.** *Let* t *be a* λ*-term. Then* t *has head normal form if and only if* $[\![t]\!]_n \neq \bot$ *for each* $n \geq 0$*. Moreover, if* t *has head normal form* $\lambda x_0 \dots \lambda x_n . x_m t_1 \cdots t_l$ *where* $0 \leq m \leq n$*, then* $[\![t]\!]_{n+1} = m$*.*

3 Future Directions

After having proved the IAM_λ correct we intend to analyze its time and space efficiency. Certainly it is not reasonable in time, but what about space? Moreover, a number of optimizations have already been suggested. Introducing jumps [DR99], for example, turns the IAM_λ into the KAM, known to be reasonable in time. It is also suggested that further optimizations, akin to memoization, could lead to optimal evaluators, in the sense of Lévy.

References

[AHS02] Samson Abramsky, Esfandiar Haghverdi, and Philip Scott. Geometry of Interaction and linear combinatory algebras. *Mathematical Structures in Computer Science*, 12(5):625–665, 2002.

[AJ94] S. Abramsky and R. Jagadeesan. New Foundations for the Geometry of Interaction. *Information and Computation*, 111(1):53–119, 1994.

[Bar84] Hendrik Pieter Barendregt. *The lambda calculus: its syntax and semantics.* North-Holland, 1984.

[DR99] Vincent Danos and Laurent Regnier. Reversible, irreversible and optimal \lambda-machines. *Theoretical Computer Science*, 227(1):79–97, 1999.

[GAL92] Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. The Geometry of Optimal Lambda Reduction. In *Proceedings of the 19th POPL*, pages 15–26, 1992.

[Ghi07] Dan R. Ghica. Geometry of Synthesis: A Structured Approach to VLSI Design. In *Proceedings of the 34th POPL*, pages 363–375, 2007.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

[Gir89] Jean-Yves Girard. Geometry of Interaction 1: Interpretation of System F. In R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo, editors, *Studies in Logic and the Foundations of Mathematics*, volume 127, pages 221–260. Elsevier, 1989.

[Kri07] Jean-Louis Krivine. A Call-by-name Lambda-calculus Machine. *Higher Order Symbol. Comput.*, 20(3):199–207, 2007.

[Mac95] Ian Mackie. The Geometry of Interaction Machine. In *Proceedings of the 22nd POPL*, pages 198–208, 1995.