On Higher-Order (In)Efficiency

Beniamino Accattoli[1] Ugo Dal Lago[2] Gabriele Vanoni[2]

[1]INRIA & LIX, École Polytechnique
[2]Università di Bologna & INRIA

LIPN, 8 October, 2020

The Simplest Setting: Closed Call-by-Name λ-calculus

We set our analysis in the simplest higher-order Turing-complete framework: pure untyped λ-calculus with weak head reduction. Abstractions are the *only* values.

Closed CbN λ-calculus

Terms	t, u	$::=$	$x \mid \lambda x.t \mid tu$
Context	C	$::=$	$\langle\cdot\rangle \mid Ct$
Reduction	$C\langle(\lambda x.t)u\rangle$	$\to$	$C\langle t\{x \leftarrow u\}\rangle$

Of course, λ-terms may **diverge**, *e.g.* $\Omega = (\lambda x.xx)(\lambda x.xx) \to \Omega$.

Types Can Guarantee Termination

Simple Types for the Closed CbN λ-Calculus

$$\text{Types} \quad A, A' \quad ::= \quad \star \mid A \to A'$$

Simple types are **sound** for termination: if $\Gamma \vdash t : A$ then $t \Downarrow$

But they are not **complete** : $\vdash \lambda x.xx : ?$

Types Can Characterize Termination

Intersection Types (IT) for the Closed CbN λ-Calculus

$$\begin{array}{lrcl} \text{Types} & A, A' & ::= & \star \mid S \to A \\ \text{Intersections} & S & ::= & \{A_1 \cdots A'_n\} \end{array}$$

where $\{\cdot\}$ is a set.

IT are **sound** and **complete** for termination: $\Gamma \vdash t : A$ iff $t \Downarrow$

A Qualitative Flavor

Intersection types satisfy subject reduction and expansion.

Thus, they cannot describe dynamic properties, *e.g.* time and space of reduction.

Type derivations could give more information, but idempotency is a problem.

Non-Idempotency in Rescue

Sequence Types (a.k.a. Non-Idempotent IT) for the CbN λ-Calculus

$$\begin{array}{lrcll} \text{Linear Types} & A & ::= & \star \mid S \to A & \\ \text{Sequence Types} & S & ::= & [A_1 \cdots A_n] & n \geq 0 \end{array}$$

Type judgments are in the form $x_1 : S \cdots x_n : S' \vdash t : A$.
We consider non-commutative sequences, this is harmless in our setting.

Again, sequence types are **sound** and **complete** for termination: $\vdash t : \star$ iff $t \Downarrow$.

A Bridge Between Syntax and Semantics

Sequence types are nothing but another way of formulating operational/denotational **semantics** of the λ-calculus in a Linear Logic fashion. There are many other examples.

Operational Semantics

- Linear Substitution Calculus (*i.e. proof net reduction*)
- Krivine's Abstract Machine
- Interaction Abstract Machine
- Polyadic approximations

Denotational Semantics

- **Relational model**
- Game semantics

A Quantitative Flavor

In the same way linear logic refines intuitionistic logic with **quantitative** information, sequence types refine IT.

They **ease** completeness proofs, since a simple induction can be used, instead of the reducibility technique.

Sequence type derivations can measure the **complexity** of λ-terms, according to different notions of evaluation.

Inference Rules: Abstraction as Value

$$\frac{}{\Gamma \vdash \lambda x.t : \star}$$

$$\frac{\Gamma, x : S \vdash t : A}{\Gamma \vdash \lambda x.t : S \to A}$$

Inference Rules: Variables

$$\frac{}{x : [A] \vdash x : A}$$

Inference Rules: Application

$$\frac{\Gamma \vdash t : [A'_1, \ldots, A'_n] \to A \quad [\Delta_i \vdash u : A'_i]_{i \in [1,\ldots,n]}}{\Gamma \uplus \sum_{i \in [1,\ldots,n]} \Delta_i \vdash tu : A}$$

A Machine Built on Type Derivations

Given a type derivation ending in $\vdash t : \star$, we can consider a **token machine** that starts from the root occurrence of $\star$.

One can move in the *natural* way inside the derivation, following that occurrence of $\star$.

The obtained machine is strongly bisimilar to the **Interaction Abstract Machine** (IAM).

It is just another IAM

Geometry of Interaction can be made **concrete** in many ways. The most known is the IAM, by Mackie and Danos & Regnier.

Its definition on type derivation is even easier: there is **no** need for any **stack**.

The **multiplicative** part is handled by **types**.

The **exponential** part is **no more needed**: in type derivations each subterm has already been copied. This way, each copy is used only once, in a **linear** way.

The Machine in Action

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : [[\star] \to \star] \vdash x : [\star_{\downarrow 16}] \to \star_{\uparrow 6}} \quad \dfrac{}{y : [\star] \vdash y : \star_{\uparrow 17}}}{y : [\star], x : [[\star] \to \star] \vdash xy : \star_{\uparrow 5}}}{y : [\star] \vdash \lambda x.xy : [[\star_{\uparrow 15}] \to \star_{\downarrow 7}] \to \star_{\uparrow 4}}}{\vdash \lambda y.\lambda x.xy : [\star_{\downarrow 18}] \to [[\star_{\uparrow 14}] \to \star_{\downarrow 8}] \to \star_{\uparrow 3}} \quad \dfrac{}{\vdash \mathsf{I} : \star_{\uparrow 19}}}{\vdash (\lambda y.\lambda x.xy)\mathsf{I} : [[\star_{\uparrow 13}] \to \star_{\downarrow 9}] \to \star_{\uparrow 2}} \quad \dfrac{\dfrac{}{z : [\star] \vdash z : \star_{\uparrow 11}}}{\vdash \lambda z.z : [\star_{\downarrow 12}] \to \star_{\uparrow 10}}}{\vdash (\lambda y.\lambda x.xy)\mathsf{I}(\lambda z.z) : \star_{\uparrow 1}}$$

Computing the Run Time

We observe the following facts:

- there is only **one** initial state,
- there are **no cycles**.

As a consequence:

1. all states are reached exactly **once**, and thus
2. run time = number of states -1 = number of occurrences of $\star$ (-1) in the derivation.

Counting the $\star$: a Semantic Measure of the IAM run time

$$\|\star\| = 1 \qquad \|S \to A\| = \|S\| + \|A\| \qquad \big\|[A_1, \ldots, A_n]\big\| = \textstyle\sum_{1 \le i \le n} \|A_i\|$$

$$\frac{}{\vdash^{0} \lambda x.t : \star} \qquad \frac{\vdash^{w}}{\vdash^{w+\|S \to A\|} \lambda x.t : S \to A}$$

$$\frac{}{\vdash^{\|A\|} x : A} \qquad \frac{\vdash^{w} \quad \left[\vdash^{v_i}\right]_{i \in [1,\ldots,n]}}{\vdash^{w+\sum v_i+\|A\|} tu : A}$$

A Different Weight System: Krivine's Machine Run Time

$$\frac{}{\vdash^{0} \lambda x.t : \star} \qquad \frac{\vdash^{w}}{\vdash^{w+1} \lambda x.t : S \to A}$$

$$\frac{}{\vdash^{1} x : A} \qquad \frac{\vdash^{w} \quad \left[\vdash^{v_i}\right]_{i \in [1,\ldots,n]}}{\vdash^{w+\sum v_i+1} tu : A}$$

Higher-Order Types need Time

- **IAM (unreasonable):** weights, thus run time, depend on types,
- **KAM (reasonable):** weights, thus run time, do *not* depend on types.

Indeed, give a term t such that $t \to^n \lambda x.u$:

- **types** of subterms of t can be **exponential** in $|t|$ and n,
- the **size** of the type derivation for t is **polynomial** in $|t|$ and n.

Ultimately, IAM inefficiency comes from the use of **higher-order** types.

Ongoing and Future Work

Ongoing Work

- Devise a weighted type system for IAM space consumption.
- Investigate the space behavior of the IAM and compare it with the KAM.

Future Work

- Understand what is space consumption in higher-order computation.