Stream Abstract Machines: Parallel and Non-deterministic execution

Marco Pedicini (Roma Tre University)
in collaboration with Mario Piazza (Univ. of Chieti-Pescara)

Workshop on Probabilistic Interactive and Higher-Order Computation

Università di Bologna, 22 — 23 Febbraio 2018

Implementation of lambda-calculus reduction

- **Optimal reduction** was introduced in J.J. Levy's PhD Thesis in 1978 and defined (by means of sharing graphs) by J. Lamping in 1990.
- **Geometry of Interaction** was introduced by J.-Y. Girard in 1985.
- In 1992, M. Abadi, J. Gonthier and J.-J. Levy made a link between optimal reduction and Geometry of Interaction.
- **Virtual Reduction** (Danos-Regnier 1993) is a way to make GoI close to optimal implementation.
- **Directed Virtual Reduction** (Danos-Pedicini-Regnier 1997) is a modification of the VR to ease implementation.
- **PELCR** (Pedicini-Quaglia 2007) is a complete implementation which permits parallel execution on multi-core machines.
- **Stream Abstract Machines** introduced in (Lai-Pedicini-Piazza 2015).

Abstract Machines

We have one machine, that is a computational unit with a memory;

- memory is a **local** and **finite** internal state;
- **communication channel** from which it receives instructions.

From the communication channel we expect an infinite stream, since it can be the result of a previous (non terminating) execution, since a machine is built to wait the next instruction...

Stream

A **stream** is a sequence of actions $\sigma : \mathbb{N} \to A$:

$$\ldots \alpha_2 \alpha_1 \alpha_0$$

The **space of actions** A will be specified; the idea is to tailor it on what the machine has to compute.

In general, we will have a program P, and its interpretation $[P]$ which is a stream of actions.
If $\tau = \beta_0 \beta_1 \ldots$, stream shuffling are denoted by

$$\sigma \ltimes \tau := \alpha_0 \beta_0 (\sigma' \ltimes \tau')$$

where σ' is the shift of the stream σ.

Execution

The machine acts by processing actions one by one:

- the **first action** is retrieved from the stream;
- the action itself represents the instruction to **modify the state** of the machine;
- it produces new **residual actions** which are reinjected in the communication channel.

It looks like an interpreter, or an operating system: the computational unit expects an infinite sequence of instructions, when new instructions are injected in the machine they are processed, if no instruction is waiting then the machine loops in idle state.

Parallelism

We can split the computation on multiple units:

- each unit has its own state,
- each action has to be directed to the unit hosting the state information required to perform the instruction.
- at the end, at the price of making actions a little bit complicated, we gain **independence from execution order**.

This is the **power of logic**: this kind of computation is possible thanks to the linear logic machinery:

proof nets and geometry of interaction

whose outcome is a **local and asynchronous** graph reduction technique.

Non-determinism

A machine with multiple unit executes in parallel by **dynamic decomposition** of the state, in such a way that the whole memory is distributed on multiple units:

> it is an **adaptive** strategy: when executing an action, its residuals are relocated on different units in such way that computational load is kept balanced.

The same mechanism enables a kind of non-determinism: the execution of the superimposition of two programs, can be concurrently obtained by injecting the merged stream:

$$[P_1 \oplus P_2] = [P_1] \ltimes [P_2]$$

Probabilistic execution

Since execution is kept local we can weight by probabilities the two programs and assign machine time in accord to probabilities:

$$[pP_1 \oplus (1-p)P_2], \quad \text{where } 0 \leq p \leq 1$$

The resulting stream is the superposition of the two stream, actions coming from the execution of the first stream appear with probability p and the ones coming from P_2 with probability $1-p$.

GOI as a Graph Reduction Technique

- The configuration at a given moment of the computation is represented by a pair:
  - the machine state which is a finite object: the **dynamic graph**
  - and the stream of **pending actions**.
- Any **machine transition** is obtained by applying an action α to the current graph G, from which we get a pair

$$\alpha.G \to (\Delta_\alpha, G \cup \{\alpha\}).$$

$\Delta_\alpha = \{\alpha_1, \beta_1', \ldots, \alpha_m, \beta_m'\}$ is a set of residual actions to be added to pending actions and $G \cup \{\alpha\}$ is the updated dynamic graph.

Machine Transition

One step of execution:

Let $\alpha :: S$ be the stream of pending actions, so that the couple $(\alpha :: S, G)$ denotes the current configuration. The transition associated to the action $\alpha \in A$ is then

$$(\alpha :: S, G) \overset{\tau_\alpha}{\to} (S \ltimes \Delta_\alpha, G \cup \{\alpha\})$$

that is, residual actions Δ_α are injected into the list of pending actions.

The basic computational step is borrowed from half-combustion strategy of DVR it includes a symbolic computation in the algebraic structure associated to the graph (the dynamic monoid) and it is a generalisation of the algebraic computations at the base of Geometry of Interaction.

A computation

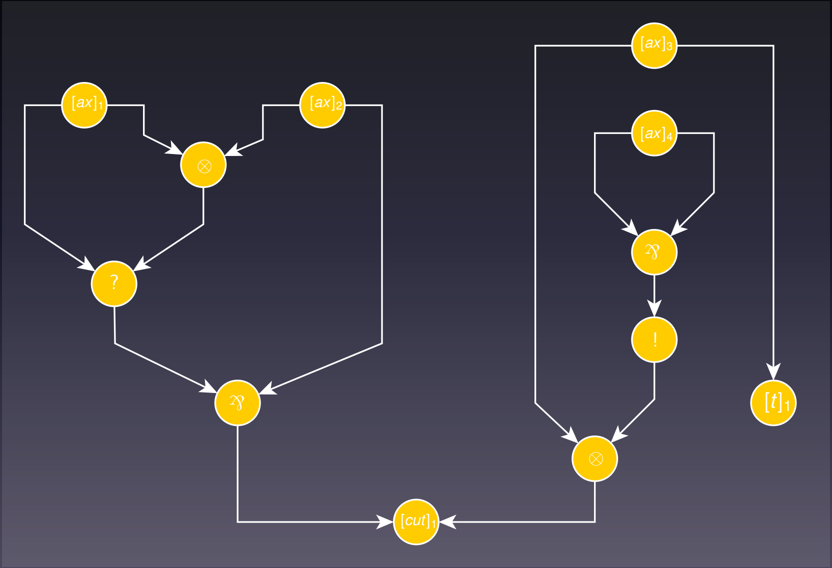
The memory of the machine is initialized with an empty graph, so that the execution of a terminating program on the abstract machine is represented by the finite sequence of transitions

$$(S^0, \emptyset) \xrightarrow{\tau_{\alpha_0}} (S^1, G^1) \xrightarrow{\tau_{\alpha_1}} \cdots \xrightarrow{\tau_{\alpha_{n-1}}} (S^n, G^n) \xrightarrow{\tau_{\alpha_n}} (\emptyset, G^{n+1})$$

where $\alpha_i \in S^i$ for all $i = 0, \ldots, n$. Note that the initial action set $S^0 = [P]$ is the interpretation of the program, and the final graph G^{n+1} represents the result of the evaluation.

In general, the execution of the machine may not terminate and, consequently, be represented by a possibly infinite sequence of elementary steps.

Example: (ΔI)



GoI interpretation

The matrix with entries in the Girard dynamic algebra:

$$\begin{array}{c|cccccc}
 & [\mathrm{ax}]_1 & [\mathrm{ax}]_2 & [\mathrm{ax}]_3 & [\mathrm{ax}]_4 & [\mathrm{cut}]_1 & [\mathrm{t}]_1 \\
\hline
[\mathrm{ax}]_1 & 0 & 0 & 0 & 0 & qx_2 + qx_1q & 0 \\
[\mathrm{ax}]_2 & 0 & 0 & 0 & 0 & qx_1p + p & 0 \\
[\mathrm{ax}]_3 & 0 & 0 & 0 & 0 & q!q + q!p & 0 \\
[\mathrm{ax}]_4 & 0 & 0 & 0 & 0 & p & 1 \\
[\mathrm{cut}]_1 & x_2^*q^* + q^*x_1^*q^* & p^*x_1^*q^* + p^* & (!q^*)q^* + (!p^*)q^* & p^* & 0 & 0 \\
[\mathrm{t}]_1 & 0 & 0 & 0 & 1 & 0 & 0
\end{array}$$

The corresponding graph

Nodes are axioms, cuts and conclusions (terminal nodes):

$$V = \{[\mathrm{ax}]_1, [\mathrm{ax}]_2, [\mathrm{ax}]_3, [\mathrm{ax}]_4, [\mathrm{cut}]_1, [\mathrm{t}]_1\}$$

and edges $((v_t, v_s), w)$ get a weight $w \in \Lambda^*$ where v_t is the target node and v_s is the source node. In this example, the "sparse" representation, consisting of the list of edges with a non-null weight, is more compact:

$$\begin{aligned} E = \{ & (([\mathrm{cut}]_1, [\mathrm{ax}]_1), qx_1q), (([\mathrm{cut}]_1, [\mathrm{ax}]_1), qx_2), (([\mathrm{cut}]_1, [\mathrm{ax}]_2), qx_1p), \\ & (([\mathrm{cut}]_1, [\mathrm{ax}]_2), p), (([\mathrm{cut}]_1, [\mathrm{ax}]_3), q!q), (([\mathrm{cut}]_1, [\mathrm{ax}]_3), q!p), \\ & (([\mathrm{cut}]_1, [\mathrm{ax}]_4), p), (([\mathrm{t}]_1, [\mathrm{ax}]_4), 1)\}. \end{aligned}$$

Sequential Abstract Machine

The **fetch-decode-execute** loop of the machine:

$$((\mathsf{POP}; \mathsf{NENV})^*; \mathsf{POP}; \mathsf{ENV}; \mathsf{HC})^*$$

$$\mathsf{ENV}\ \frac{(\langle\alpha\rangle, \mathrm{NULL}, S, G) \qquad \alpha \neq \mathbf{0}}{(\langle\alpha\rangle, \mathsf{target}(\alpha), S, G \cup \{\mathsf{target}(\alpha)\})}$$

$$\mathsf{POP}\ \frac{(\langle\rangle, \mathrm{NULL}, \alpha :: S, G)}{(\langle\alpha\rangle, \mathrm{NULL}, S, G)}$$

$$\mathsf{NENV}\ \frac{(\langle\alpha\rangle, \mathrm{NULL}, S, G) \qquad \alpha = \mathbf{0}}{(\langle\rangle, \mathrm{NULL}, S, G)}$$

$$\mathsf{HC}\ \frac{(\langle\alpha\rangle, \mathsf{target}(\alpha), S, G)}{(\langle\rangle, \mathrm{NULL}, S \ltimes \mathsf{execute}(\alpha), G \cup \{\alpha\})}$$

Girard dynamic algebra Λ^*

The so-called *Girard dynamic algebra* Λ^* is the dynamic monoid generated by the constants p, q, and a family $W = \{w_i\}_i$ of exponential generators, with a morphism $!(.)$, such that for any $u \in \Lambda^*$:

$$\begin{array}{lrcll} \text{(annihilation)} & x^*y & = & \delta_{xy} & \text{for } x, y = p, q, w_i, \\ \text{(swapping)} & !(u)w_i & = & w_i!^{e_i}(u), & \end{array}$$

where δ_{xy} is the Kronecker operator,e_i is an integer associated with w_i called the *lift* of w_i, i is called the *name* of w_i and we often write w_{i,e_i} to explicitly note the lift of the generator. Notice that swapping and annihilation rules imply that for every $a, b \in \Lambda^*$ either $b^*a = 0$ or it has **a stable form**, that is Λ^* satisfies SF condition.

GOI Actions

An action α belongs to the set

$$A := (N \times \{+, -\})^2 \times \Lambda^* \cup \{\mathbf{0}\}.$$

Roughly speaking, α is either a labeled edge or a null action. Labels are in Λ^*, the so-called *Girard Dynamic Monoid*, whose product satisfies the **decidable stable form condition** and may thus be expressed by two functions

$$\mathsf{A} : \Lambda^* \to \Lambda^*$$

and

$$\mathsf{B} : \Lambda^* \to \Lambda^*$$

giving the positive and the * part of the stable form of a product; N is a countable set of names, on which a function new() providing a new fresh name (i.e., a reference to a node which is in any graph) is defined.

Dynamic graphs

The dump component of a configuration is a *polarised dynamic graph*:

$G = (V, E)$, where V and E denote a set of nodes and a set of edges

For nodes $v \in V$ we have

$$\mathrm{h} : V \to N$$

and its inverse

$$\rho : N \to V$$

which provide a reference corresponding to a node, and the node identified by the name, respectively.

Labelled edges

Any edge of E labeled with a weight taken in Λ^* and polarities attached to source and target nodes:

$$((x^{\epsilon_s}, y^{\epsilon_t}), w)\,.$$

In what follows, we make reference to sequences of graphs $G_i = (V_i, E_i)$, each for any step of execution, and to a function ρ providing a cumulated (in time) node reference $\rho : N \to \bigcup_i V_i$ and $\rho_i : N \to V_i$, whenever we want to stress the specific graph G_i.

Elementary computational step

The elementary computational step turns out to be the half combustion of an action α with respect the current dump graph G_i.

The function

$$\mathrm{HC}_i : A \to 2^A,$$

and the evaluation $\mathrm{HC}_i(\alpha)$ which is a set of newly produced actions;

α is an action,

$$\alpha = ((x^{\epsilon_s}, y^{\epsilon_t}), w)$$

where $x, y \in N$, $\epsilon_s, \epsilon_t \in \{+, -\}$ and $w \in \Lambda^*$; $y \in N$, $\rho_i : N \to V_i$ so that $\rho_i(y) \in V_i$ can be split in two parts according to the target polarities of the edges pointing to it.

Polarised compositions

Let us denote $y^{-\epsilon_t} = \{\beta_1, \ldots \beta_m\}$ the set of edges pointing to y with opposite target polarity with respect to ϵ_t, the target polarity of α.
Any of the β_i is an edge in a polarised dynamic graph, namely $\beta_i = ((x_i^{\epsilon_i}, y^{-\epsilon_t}), w_i)$.

Persistent compositions

Then, for any pair β_i we have that their composition $\mathsf{hc}(\alpha, \beta_i)$ is defined if

$$\mathsf{A}(w^* w_i) \quad \text{and} \quad \mathsf{B}(w^* w_i)$$

are defined as well.

That is, $w^* w_i \neq 0$ and

$$\mathsf{hc}(\alpha, \beta_i) = \{\beta_i', \alpha_i\}$$

where $z = \mathsf{new}()$,

$$\alpha_i = ((z^+, x_i^{-\epsilon_i}), \mathsf{A}(w^* w_i))$$

and

$$\beta_i' = ((z^-, x_i^{\epsilon_i}), \mathsf{B}(w^* w_i)).$$

Residual Edges

The set of *residual edges* with all the pairs such that

$$\text{hc}(\alpha, \beta_i)$$

is defined

$$\text{RES}(\alpha) := \bigcup_{\beta_i} \text{hc}(\alpha, \beta_i)$$

and this set is applied in the rule HC to define the stream execute(α) which is combined with the current stream S in the current configuration ($S \ltimes$ execute(α)).

Multiple Unit Machines

If we want to represent *k* units we use a tuple of units that we denote by tensors on components to indicate that elements have to be globally compatible, at each time step via the reference function; any configuration is

$$(p, S_1 \otimes \cdots \otimes S_k, E_1 \otimes \cdots \otimes E_k, C_1 \otimes \cdots \otimes C_k, D_1 \otimes \cdots \otimes D_k)$$

where $p \in \{1, \ldots, k\}$ indicates on which unit the next step has to be performed.

...thanks!