Sequential algorithms
and
innocent strategies

share the same execution mechanism

Pierre-Louis Curien

(IRIF, πr^2, CNRS – Paris 7 – INRIA)

April 2019, Galop workshop, Prague

PLAN of the TALK

1. Geometric abstract machine “in the abstract” : tree interaction and pointer interaction (designed in the setting of **Curien-Herbelin**’s abstract Böhm trees)
2. Turbo-reminder on sequential algorithms (3 flavours, with focus on two : as programs, and abstract)
3. Geometric abstract machine in action
4. Turbo-reminder on HO innnocent strategies for PCF types (2 flavours, “meager and fat” = views versus plays)
5. Geometric abstract machine in action
6. (Inconclusive !) conclusion : the message is : “il y a quelque chose à gratter”

Tree interaction

Setting of alternating 2-players' games where Opponent starts.
Strategies as trees (or forests) branching after each Player's move
Interaction by tree superposition :

STRATEGIES	EXECUTION
$x\ a \begin{cases} b\ c \\ b'\ \ \ldots \end{cases}$	$\langle x, \mathbf{1} \rangle\ a \begin{cases} \langle b, \mathbf{3} \rangle\ c \\ b'\ \ \ldots \end{cases}$
$a\ b \begin{cases} c\ \ \ldots \\ d\ \ \ldots \end{cases}$	$\langle a, \mathbf{2} \rangle\ b \begin{cases} \langle c, \mathbf{4} \rangle\ \ \ldots \\ d\ \ \ldots \end{cases}$

The trace of the interaction is the "common branch" $x\ a\ b\ c$:

Step n of the machine played in one of the strategies always followed by step $(n+1)'$ in the same strategy. Next move $(n+1)$ is played in the other strategy (choice of branch dictated by $(n+1)'$).

Pointer interaction

Now, in addition, Player's moves are equipped with a pointer to an ancestor Opponent's move.

STRATEGIES	EXECUTION
$x\ a \begin{cases} b\ [c, \overset{0}{\hookleftarrow}] \\ b'\ \ldots \end{cases}$	$\langle x, \mathbf{1} \rangle\ a \begin{cases} \langle b, \mathbf{3} \rangle\ [c, \overset{0}{\hookleftarrow}] \\ \langle b', \mathbf{5} \rangle\ \ldots \end{cases}$
$a\ [b, \overset{0}{\hookleftarrow}] \begin{cases} c\ [b', \overset{1}{\hookleftarrow}] \\ d\ \ldots \end{cases}$	$\langle a, \mathbf{2} \rangle\ [b, \overset{0}{\hookleftarrow}] \begin{cases} \langle c, \mathbf{4} \rangle\ [b', \overset{1}{\hookleftarrow}] \\ d\ \ldots \end{cases}$

If $(n+1)'$ points to m, then $(n+1)$ should be played under m'.

Concrete data structures

A concrete data structure (or *cds*) $\mathbf{M} = (C, V, E, \vdash)$ is given by three sets C, V, and $E \subseteq C \times V$ of *cells*, *values*, and *events*, and a relation $\vdash$ between finite parts of E (or cardinal ≤ 1 for simplicity) and elements of C, called the enabling relation. We write simply $e \vdash c$ for $\{e\} \vdash c$. A cell c such that $\vdash c$ is called *initial*.

(+ additional conditions : well-foundedness, stability)

Proofs of cells c are sequences in $(CV)^\star$ defined recursively as follows : If c is initial, then it has an empty proof. If $(c_1, v_1) \vdash c$, and if p_1 is a proof of c_1, then $p_1\, c_1\, v_1$ is a proof of c.

Configurations (or strategies, in the game semantics terminology)

A configuration is a subset x of E such that :

(1) $(c, v_1), (c, v_2) \in x \Rightarrow v_1 = v_2$.

(2) If $(c, v) \in x$, then x contains a proof of c.

The conditions (1) and (2) are called consistency and safety, respectively.

The set of configurations of a cds $\mathbf{M}$, ordered by set inclusion, is a partial order denoted by $(D(\mathbf{M}), \leq)$ (or $(D(\mathbf{M}), \subseteq)$).

Some terminology

Let x be a set of events of a cds. A cell c is called :

- filled (with v) in x iff $(c, v) \in x$,

- accessible from x iff x contains an enabling of c, and c is not filled in x (notation $c \in A(x)$).

Some examples of cds's

(1) Flat cpo's : for any set $\mathbf{X}$ we have a cds

$\mathbf{X}_\perp = (\{?\}, \mathbf{X}, \{?\}\times\mathbf{X}, \{\vdash ?\})$ with $D(\mathbf{X}_\perp) = \{\emptyset\}\cup\{(?, x) \mid x \in \mathbf{X}\}$

Typically, we have the flat cpo $\mathbf{N}_\perp$ of natural numbers.

(2) Any first-order signature Σ gives rise to a cds $\mathbf{M}_\Sigma$:

— cells are occurrences described by words of natural numbers,

— values are the symbols of the signature,

— all events are permitted,

— $\vdash \epsilon$, and $(u, f) \vdash ui$ for all $1 \leq i \leq arity(f)$.

Product of two cds's

Let M and M' be two cds's. We define the product $\mathrm{M} \times \mathrm{M}' = (C, V, E, \vdash)$ of M and M' by :

- $C = \{c.1 \mid c \in C_{\mathrm{M}}\} \cup \{c'.2 \mid c' \in C_{\mathrm{M}'}\}$,
- $V = V_{\mathrm{M}} \cup V_{\mathrm{M}'}$,
- $E = \{(c.1, v) \mid (c, v) \in E_{\mathrm{M}}\} \cup \{(c'.2, v') \mid (c', v') \in E_{\mathrm{M}'}\}$,
- $(c_1.1, v_1), \ldots, c_n.1, v_n) \vdash c.1 \Leftrightarrow (c_1, v_1), \ldots (c_n, v_n) \vdash c$ (and similarly for M').

Fact : $\mathrm{M} \times \mathrm{M}'$ generates $D(\mathrm{M}) \times D(\mathrm{M}')$.

Sequential algorithms as programs

Morphisms between two cds's $\mathbf{M}$ and $\mathbf{M}'$ are forests described by the following formal syntax :

$$\begin{array}{l} F ::= \{T_1, \ldots, T_n\} \\ T ::= \mathit{request}\ c'\ U \\ U ::= \mathit{valof}\ c\ \mathit{is}\ [\ldots v \mapsto U_v \ldots] \mid \mathit{output}\ v'\ F \end{array}$$

satisfying some well-formedness conditions :

— A $\mathit{request}\ c'$ can occur only if the projection on $\mathbf{M}'$ of the branch connecting it with the root is a proof of c'.

— Along a branch, knowledge concerning the projection on $\mathbf{M}$ is accumulated in the form of a configuration x, and a $\mathit{valof}\ c$ can occur only if c is accessible from the current x. In particular, no repeated $\mathit{valof}\ c$!

Exponent of two cds's

If $\mathbf{M}$, $\mathbf{M}'$ are two cds's, the cds $\mathbf{M} \to \mathbf{M}'$ is defined as follows :

- If x is a finite configuration of $\mathbf{M}$ and $c' \in C_{\mathbf{M}'}$, then xc' is a cell of $\mathbf{M} \to \mathbf{M}'$.
- The values and the events are of two types :
  - If c is a cell of $\mathbf{M}$, then $valof\ c$ is a value of $\mathbf{M} \to \mathbf{M}'$, and $(xc', valof\ c)$ is an event of $\mathbf{M} \to \mathbf{M}'$ iff c is accessible from x ;
  - if v' is a value of $\mathbf{M}'$, then $output\ v'$ is a value of $\mathbf{M} \to \mathbf{M}'$, and $(xc', output\ v')$ is an event of $\mathbf{M} \to \mathbf{M}'$ iff (c', v') is an event of $\mathbf{M}'$.
- The enablings are given by the following rules :

$$
\begin{array}{lll}
\vdash \emptyset c' & \text{iff} & \vdash c' \\
(yc', valof\ c) \vdash xc' & \text{iff} & x = y \cup \{(c, v)\} \\
(xd', output\ w') \vdash xc' & \text{iff} & (d', w') \vdash c'
\end{array}
$$

An example of a sequential algorithm

The following is the intepretation of

$$\lambda f.\texttt{case}\ f\,\texttt{T}\,\texttt{F}\ [\texttt{T} \to \texttt{F}] : (\texttt{bool}_{11} \times \texttt{bool}_{12} \to \texttt{bool}_1) \to \texttt{bool}_\epsilon$$

$$\textit{request}\,\texttt{?}_\epsilon\ \textit{valof}\,\bot\bot\texttt{?}_1 \begin{cases} \textit{is valof}\ \texttt{?}_{11}\ \textit{valof}\ \texttt{T}\bot\texttt{?}_1 \left\{ \textit{is valof}\ \texttt{?}_{12}\ \textit{valof}\ \texttt{TF?}_1 \left\{ \textit{is output}\ \texttt{T}_1\ \textit{output}\ \texttt{F}_\epsilon \right.\right. \\ \textit{is valof}\ \texttt{?}_{12}\ \textit{valof}\ \bot\texttt{F?}_1 \left\{ \textit{is valof}\ \texttt{?}_{11}\ \textit{valof}\ \texttt{TF?}_1 \left\{ \textit{is output}\ \texttt{T}_1\ \textit{output}\ \texttt{F}_\epsilon \right.\right. \\ \textit{is output}\ \texttt{T}_1\ \textit{output}\ \texttt{F}_\epsilon \end{cases}$$

to be contrasted with the interpretation of the same term as a set of views in HO semantics :

$$\texttt{?}_\epsilon\ \texttt{?}_1 \begin{cases} \texttt{?}_{11}\ \texttt{T}_{11} \\ \texttt{?}_{12}\ \texttt{F}_{12} \\ \texttt{T}_1\ \texttt{F}_\epsilon \end{cases}$$

An example of execution of sequential algorithms

$F' : \mathbf{B} \times \mathbf{M}_\Sigma \to \mathbf{B}$ explores successively the root of its second input, its first input, and the first son of its second input (if of the form $(f(\Omega, \Omega))$ to produce F, while $F = \langle F_1, F_2 \rangle$, where $F_1 : \mathbf{M}_\Sigma \to \mathbf{B}$ (resp. $F_2 : \mathbf{M}_\Sigma \to \mathbf{M}_\Sigma$) produces F without looking at its argument (resp. is the identity).

Branch of $F'' = F' \circ F : \mathbf{M}_\Sigma \to \mathbf{B}$ being built :

$$\{\langle request\ ?, \mathbf{1}\rangle\ valof\ \epsilon\ \langle is\ f, \mathbf{2}\rangle\ valof\ 1\ \langle is\ f, \mathbf{3}\rangle\ output\ \mathtt{F}$$

Branch of F' being explored :

$$\{\langle request\ ?, \mathbf{1.1}\rangle\ valof\ \epsilon_2\ \underline{\langle is\ f_2, \mathbf{2.2}\rangle}\ valof\ ?_1\ \langle is\ F_1, \mathbf{2}.\mathbf{4}\rangle\ \underline{valof\ 1_2}\ \langle is\ f_2, \mathbf{3.2}\rangle\ output\ \mathtt{F}$$

Branches of F being explored :

$$\begin{cases} \langle request\ ?_1, \mathbf{2.3}\rangle\ output\ \mathtt{F}_1 \\ \langle request\ \epsilon_2, \mathbf{1.2}\rangle\ valof\ \epsilon\ \langle is\ f, \mathbf{2.1}\rangle\ output\ f_2\ \langle request\ 1_2, \mathbf{2.5}\rangle\ valof\ 1\ \langle is\ f, \mathbf{3.1}\rangle\ output\ f_2 \end{cases}$$

Pointer interaction : $\mathbf{2.5}'$ points to $(\mathbf{2.2})$, hence $\mathbf{2.5}$ is played under $(\mathbf{2.2})'$. Pointers are implicit in sequential algorithms, i.e., can be uniquely reconstructed : each $valof\ c$ points to $is\ v$, where $is\ v$ follows $valof\ d$ and $(d, v) \vdash c$.

Equivalent definitions of sequential algorithms

We have 3 equivalent definitions of **sequential algorithms** :

1. as programs (our focus here) $\leadsto$ ABSTRACT MACHINE
2. as configurations of $\mathbf{M} \to \mathbf{M}'$ $\leadsto$ CART. CLOSED STRUCTURE
3. as abstract algorithms (or as pairs of a function and a computation strategy for it). Abstract algorithms are the **fat** version of configurations : if $(yc', u) \in a$, $y \leq x$, and $(xc', u) \in E_{\mathbf{M} \to \mathbf{M}'}$, then we set $a^+(xc') = u$. If we spell this out (for $y \leq x$) :

$$\begin{array}{lcl} (yc', \mathit{valof}\ c) \in a \text{ and } c \in A(x) & \Rightarrow & a^+(xc') = \mathit{valof}\ c \\ (yc', \mathit{output}\ v') \in a & \Rightarrow & a^+(xc') = \mathit{output}\ v' \end{array}$$

$\leadsto$ "CONCEPTUAL" COMPOSITION

Composing abstract algorithms

Let M, M' and M'' be cds's, and let f and f' be two abstract algorithms from M to M' and from M' to M'', respectively. The function g, defined as follows, is an abstract algorithm from M to M'' :

$$g(xc'') = \begin{cases} \mathit{output}\ v'' & \text{if } f'((f{\bullet}x)c'') = \mathit{output}\ v'' \\ \\ \mathit{valof}\ c & \text{if } \begin{cases} f'((f{\bullet}x)c'') = \mathit{valof}\ c' \text{ and} \\ f(xc') = \mathit{valof}\ c\ . \end{cases} \end{cases}$$

Perspective

Thus, sequential algorithms admit a meager form (as programs or as configurations) and a fat form (as abstract algorithms)

Similarly, innocent strategies as sets of plays are in fat form, while the restriction to their set of views is their meager form

— Fat composition is defined synthetically.
— Meager composition is defined via an abstract machine : the same for both = the Geometric Abstract Machine (with the proviso that the execution of sequential algorithms uses an additional call-by-need mechanism added to the machine).

PCF Böhm trees

$$M := \lambda\vec{x}.W \qquad \text{(the length of } \vec{x} \text{ may be zero)}$$
$$W := n \mid \texttt{case}\ x\vec{M}\ [\ldots m \to W_m \ldots]$$

Taking the syntax for PCF types $\sigma ::= \texttt{nat} \parallel \sigma \to \sigma$, we have the following typing rules :

$$\frac{\Gamma, x_1 : \sigma_1, \ldots x_n : \sigma_n \vdash W : \texttt{nat}}{\Gamma \vdash \lambda x_1 \ldots x_n.W : \sigma_1 \to \ldots \to \sigma_n \to \texttt{nat}}$$

$$\frac{}{\Gamma \vdash n : \texttt{nat}} \qquad \frac{\ldots \Gamma, x : \sigma \vdash M_i : \sigma_i \ldots \quad \ldots \Gamma, x : \sigma \vdash W_j : \texttt{nat} \ldots}{\Gamma, x : \sigma \vdash \texttt{case}\ xM_1 \ldots M_p\ [m_1 \to W_1 \ldots m_q \to W_q] : \texttt{nat}}$$

where, in the last rule, $\sigma = \sigma_1 \to \ldots \to \sigma_p \to \texttt{nat}$

PCF Böhm trees as strategies : an example

All PCF Böhm trees can be transcribed as trees. We decorate PCF types A as $[\![A]\!]^{\epsilon}$, where each copy of $\mathtt{nat}$ is decorated with a word $u \in \mathbb{N}^*$:

$$[\![A^1 \to \ldots \to A^n \to \mathtt{nat}]\!]_u = [\![A^1]\!]_{u1} \to \ldots \to [\![A^n]\!]_{un} \to \mathtt{nat}_u$$

All moves in the HO arenas for PCF types are of the form $?_u$ or n_u.

Moreover $?_u$ has polarity 0 (resp. P) if u is of even (resp. odd) length, while n_u has polarity P (resp. O) if u is of even (resp. odd) length.

The PCF Böhm tree $\lambda f.\mathtt{case}\ f3\ [4 \to 7, 6 \to 9]$ reads as follows :

$$\lambda f.\ \mathtt{case}\ f \begin{cases} (3) \\ 4 \to 7 \\ 6 \to 9 \end{cases} \qquad h = ?_{\epsilon}[?_1, \overset{0}{\hookleftarrow}] \begin{cases} ?_{11}[3_{11}, \overset{0}{\hookleftarrow}] \\ 4_1[7_{\epsilon}, \overset{1}{\hookleftarrow}] \\ 6_1[9_{\epsilon}, \overset{1}{\hookleftarrow}] \end{cases}$$

PCF Böhm trees as strategies : full compilation

We need an auxiliary functions

$$arity(A,\epsilon) = n \quad arity(A, iu) = arity(A^i, u) \quad (A = A^1 \to \ldots \to A^n \to \mathtt{nat})$$

$$access(x, (\vec{x}, u) \cdot L, i) = \begin{cases} [?_{uj}, \overset{i}{\hookleftarrow}] & \text{if } x \in \vec{x} \text{ with } x = x_j \\ access(x, L, i+1) & \text{otherwise} \end{cases}$$

We translate $M : A$ to $[\![M]\!]_\epsilon^{[\,]}$, where

$$[\![\lambda\vec{x}.W]\!]_u^L = ?_u \; [\![W]\!]_u^{(\vec{x},u)\cdot L}$$

$[\![n]\!]_u^L = n_u$ (pointer reconstructed by well-bracketing)

$$[\![\mathtt{case}\ x\vec{M}\ [\ldots m \to W_m \ldots]]\!]_u^L = [?_{vj}, \overset{i}{\hookleftarrow}] \begin{cases} \vdots \\ [\![M_l]\!]_{vjl}^L \\ \vdots \\ \vdots \\ m_{vj}\ [\![W_m]\!]_u^L \\ \vdots \end{cases}$$

where $access(x, L, 0) = [?_{vj}, \overset{i}{\hookleftarrow}]$ and $1 \leq l \leq arity(A, vj)$.

An example of execution of HO strategies : the strategies

$$Kierstead_1 = \lambda f.\texttt{case}\ f(\lambda x.\texttt{case}\ f(\lambda y.\texttt{case}\ x))$$

applied to

$$\lambda g.\texttt{case}\ g(\texttt{case}\ gT\ [T \to T, F \to F])\ [T \to F, F \to T]$$

$$?_\epsilon[?_1, \overset{0}{\hookleftarrow}] \begin{cases} ?_{11}[?_1, \overset{1}{\hookleftarrow}] \begin{cases} ?_{11}\underline{[?_{111}, \overset{1}{\hookleftarrow}]} \begin{cases} T_{111}[T_{11}, \overset{1}{\hookleftarrow}] \\ F_{111}[F_{11}, \overset{1}{\hookleftarrow}] \end{cases} \\ T_1[T_{11}, \overset{1}{\hookleftarrow}] \\ F_1[F_{11}, \overset{1}{\hookleftarrow}] \end{cases} \\ T_1[T_\epsilon, \overset{1}{\hookleftarrow}] \\ F_1[F_\epsilon, \overset{1}{\hookleftarrow}] \end{cases}$$

$$?_1[?_{11}, \overset{0}{\hookleftarrow}] \begin{cases} ?_{111}[?_{11}, \overset{1}{\hookleftarrow}] \begin{cases} ?_{111}[\mathtt{F}_{111}, \overset{0}{\hookleftarrow}] \\ \mathtt{T}_{11}[\mathtt{T}_{111}, \overset{1}{\hookleftarrow}] \\ \mathtt{F}_{11}[\mathtt{F}_{111}, \overset{1}{\hookleftarrow}] \end{cases} \\ \mathtt{T}_{11}[\mathtt{F}_1, \overset{1}{\hookleftarrow}] \\ \mathtt{F}_{11}[\mathtt{T}_1, \overset{1}{\hookleftarrow}] \end{cases}$$

An example of execution of HO strategies : the execution

$$\langle ?_\epsilon, \mathbf{1}\rangle[?_1, \overset{0}{\hookleftarrow}] \begin{cases} \langle ?_{11}, \mathbf{3}\rangle[?_1, \overset{1}{\hookleftarrow}] \begin{cases} \langle ?_{11}, \mathbf{5}\rangle[?_{111}, \overset{1}{\hookleftarrow}] \begin{cases} \langle \mathtt{T}_{111}, \mathbf{15}\rangle[\mathtt{T}_{11}, \overset{1}{\hookleftarrow}] \end{cases} \\ \langle \mathtt{F}_1, \mathbf{17}\rangle[\mathtt{F}_{11}, \overset{1}{\hookleftarrow}] \end{cases} \\ \langle ?_{11}, \mathbf{7}\rangle[?_1, \overset{1}{\hookleftarrow}] \begin{cases} \langle ?_{11}, \mathbf{9}\rangle[?_{111}, \overset{1}{\hookleftarrow}] \begin{cases} \langle \mathtt{F}_{111}, \mathbf{11}\rangle[\mathtt{F}_{11}, \overset{1}{\hookleftarrow}] \end{cases} \\ \langle \mathtt{T}_1, \mathbf{13}\rangle[\mathtt{T}_{11}, \overset{1}{\hookleftarrow}] \end{cases} \\ \langle \mathtt{T}_1, \mathbf{19}\rangle[\mathtt{T}_\epsilon, \overset{1}{\hookleftarrow}] \end{cases}$$

$$\begin{cases} \langle ?_1, \mathbf{2}\rangle[?_{11}, \overset{0}{\hookleftarrow}] \begin{cases} \langle ?_{111}, \mathbf{6}\rangle[?_{11}, \overset{1}{\hookleftarrow}] \begin{cases} \langle ?_{111}, \mathbf{10}\rangle[\mathtt{F}_{111}, \overset{0}{\hookleftarrow}] \\ \langle \mathtt{T}_{11}, \mathbf{14}\rangle[\mathtt{T}_{111}, \overset{1}{\hookleftarrow}] \end{cases} \\ \langle \mathtt{F}_{11}, \mathbf{18}\rangle[\mathtt{T}_1, \overset{1}{\hookleftarrow}] \end{cases} \\ \langle ?_1, \mathbf{4}\rangle[?_{11}, \overset{0}{\hookleftarrow}] \begin{cases} \langle \mathtt{T}_{11}, \mathbf{16}\rangle[\mathtt{F}_1, \overset{1}{\hookleftarrow}] \end{cases} \\ \langle ?_1, \mathbf{8}\rangle[?_{11}, \overset{0}{\hookleftarrow}] \begin{cases} \langle \mathtt{F}_{11}, \mathbf{12}\rangle[\mathtt{T}_1, \overset{1}{\hookleftarrow}] \end{cases} \end{cases}$$

A form of conclusion

Sequential algorithms and HO innocent strategies differ in at least two respects :

- Sequential algorithms are intensional even for purely functional programs, cf. example $\lambda f.\mathtt{case}\ f\,\mathtt{T}\,\mathtt{F}\ [\mathtt{T} \to \mathtt{F}]$
- Sequential algorithms have memory (or work in call-by-need manner), e.g. the model "normalises"

$$\lambda x.\mathtt{case}\ x\ [3 \to \mathtt{case}\ x\ [3 \to 4]]$$

into

$$request\ ?_\epsilon\ valof\ ?_1 \left\{is\ 3_1\ output\ 4_\epsilon\right.$$

As for the second aspect, one could think of a multiset version of the exponent of two cds' (cf. the two familiar "bangs" in the relational and coherent semantics of linear logic).