Sequential algorithms and innocent strategies share the same execution mechanism

Pierre-Louis Curien

(IRIF, πr², 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 innocent 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:

<table>
<thead>
<tr>
<th>STRATEGIES</th>
<th>EXECUTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \ a \ \begin{cases} b &amp; c \ b' &amp; \ldots \end{cases}$</td>
<td>$\langle x, 1 \rangle \ a \ \begin{cases} \langle b, 3 \rangle &amp; c \ b' &amp; \ldots \end{cases}$</td>
</tr>
<tr>
<td>$a \ b \ \begin{cases} c &amp; \ldots \ d &amp; \ldots \end{cases}$</td>
<td>$\langle a, 2 \rangle \ b \ \begin{cases} \langle c, 4 \rangle &amp; \ldots \ d &amp; \ldots \end{cases}$</td>
</tr>
</tbody>
</table>

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 $(n + 1)'$).
Now, in addition, Player’s moves are equipped with a pointer to an ancestor Opponent’s move.

\[
\begin{align*}
    &\text{STRATEGIES} & \quad & \text{EXECUTION} \\
    & x \ a \ \\
    & \begin{cases} b \ [c, \leftarrow] \ \\
        b' \ \ldots \ 
    \end{cases} & \quad & \langle x, 1 \rangle \ a \ \\
    & \begin{cases} \langle b, 3 \rangle \ [c, \leftarrow] \ \\
        \langle b', 5 \rangle \ \ldots \ 
    \end{cases} \\
    & a \ [b, \leftarrow] \ \\
    & \begin{cases} c \ [b', \leftarrow] \ \\
        d \ \ldots \ 
    \end{cases} & \quad & \langle a, 2 \rangle \ [b, \leftarrow] \ \\
    & \begin{cases} \langle c, 4 \rangle \ [b', \leftarrow] \ \\
        d \ \ldots \ 
    \end{cases}
\end{align*}
\]

If \((n + 1)’\) points to \(m\), then \((n + 1)\) should be played under \(m’\).
Concrete data structures

A concrete data structure (or cds) $M = (C, V, E, \vdash)$ is given by three sets $C, V, \text{ and } E \subseteq C \times V$ of cells, values, and events, and a relation $\vdash$ between finite parts of $E$ (or cardinal $\leq 1$ 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 $(CV)^*$ 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 $\mathcal{M}$, ordered by set inclusion, is a partial order denoted by $(\mathcal{D}(\mathcal{M}), \leq)$ (or $(\mathcal{D}(\mathcal{M}), \subseteq)$).
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 \( X \) we have a cds

\[
X_\bot = (\{\top\}, X, \{\top\} \times X, \{\bot\}) \quad \text{with} \quad D(X_\bot) = \{\emptyset\} \cup \{(? , x) \mid x \in X\}
\]

Typically, we have the flat cpo \( \mathbb{N}_\bot \) of natural numbers.

(2) Any first-order signature \( \Sigma \) gives rise to a cds \( M_\Sigma \):

- cells are occurrences described by words of natural numbers,
- values are the symbols of the signature,
- all events are permitted,
- \( \vdash \epsilon \), and \( (u, f) \vdash u_i \) for all \( 1 \leq i \leq \text{arity}(f) \).
Product of two cds’s

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

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

Fact: $M \times M'$ generates $D(M) \times D(M')$. 
Sequential algorithms as programs

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

$$F ::= \{T_1, \ldots, T_n\}$$
$$T ::= \text{request } c' \text{ } U$$
$$U ::= \text{valof } c \text{ is } \{\ldots v \mapsto U_v \ldots\} \mid \text{output } v' \text{ } F$$

satisfying some well-formedness conditions:

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

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

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

— If $x$ is a finite configuration of $M$ and $c' \in C_{M'}$, then $xc'$ is a cell of $M \to M'$.

— The values and the events are of two types:
  
  — If $c$ is a cell of $M$, then $\text{valof } c$ is a value of $M \to M'$, and $(xc', \text{valof } c)$ is an event of $M \to M'$ iff $c$ is accessible from $x$;
  
  — if $v'$ is a value of $M'$, then $\text{output } v'$ is a value of $M \to M'$, and $(xc', \text{output } v')$ is an event of $M \to M'$ iff $(c', v')$ is an event of $M'$.

— The enablings are given by the following rules:

\[
\begin{align*}
\vdash \emptyset c' & \quad \text{iff} \quad \vdash c' \\
(yc', \text{valof } c) \vdash xc' & \quad \text{iff} \quad x = y \cup \{(c, v)\} \\
(xd', \text{output } w') \vdash xc' & \quad \text{iff} \quad (d', w') \vdash c'
\end{align*}
\]
An example of a sequential algorithm

The following is the interpretation of

$$\lambda f. \text{case } f \text{T F}[T \rightarrow F] : (\text{bool}_{11} \times \text{bool}_{12} \rightarrow \text{bool}_{1}) \rightarrow \text{bool}_{\epsilon}$$

\[\text{request} ?_\epsilon \text{ valof } \bot \bot \text{?}_1 \begin{cases} \text{is valof } ?_{11} \text{ valof } T \bot \text{?}_1 \{ \text{is valof } ?_{12} \text{ valof } T F \text{?}_1 \{ \text{is output } T_1 \text{ output } F_\epsilon \\
\text{is valof } ?_{12} \text{ valof } F \text{?}_1 \{ \text{is valof } ?_{11} \text{ valof } T F \text{?}_1 \{ \text{is output } T_1 \text{ output } F_\epsilon \\
\text{is output } T_1 \text{ output } F_\epsilon \\ \end{cases} \end{cases} \]

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

\[?_\epsilon \text{?}_1 \begin{cases} ?_{11} \text{T}_{11} \\
?_{12} \text{F}_{12} \\
T_1 \text{F}_\epsilon \end{cases} \]
An example of execution of sequential algorithms

$F' : B \times M_\Sigma \rightarrow B$ 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 $F_1 : M_\Sigma \rightarrow B$ (resp. $F_2 : M_\Sigma \rightarrow M_\Sigma$) produces $F$ without looking at its argument (resp. is the identity).

Branch of $F'' = F' \circ F : M_\Sigma \rightarrow B$ being built :

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

Branch of $F'$ being explored :

$$\{\langle \text{request }?, 1.1 \rangle \text{ valof } \epsilon_2 \langle \text{is } f_2, 2.2 \rangle \text{ valof } ?_1 \langle \text{is } F_1, 2.4 \rangle \text{ valof } 1_2 \langle \text{is } f_2, 3.2 \rangle \text{ output } F\}$$

Branches of $F$ being explored :

$$\begin{cases}
\langle \text{request }?_1, 2.3 \rangle \text{ output } F_1 \\
\langle \text{request }\epsilon_2, 1.2 \rangle \text{ valof } \epsilon \langle \text{is } f, 2.1 \rangle \text{ output } f_2 \langle \text{request } 1_2, 2.5 \rangle \text{ valof } 1 \langle \text{is } f, 3.1 \rangle \text{ output } f_2
\end{cases}$$

Pointer interaction : $2.5'$ points to $(2.2)$, hence $2.5$ is played under $(2.2)'$. Pointers are implicit in sequential algorithms, i.e., can be uniquely reconstructed : each $\text{valof } c$ points to $\text{is } v$, where $\text{is } v$ follows $\text{valof } d$ 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 \(M \to M' \leadsto\) **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 \((xc', u) \in E_{M \to M'}, \) then we set \(a^+(xc') = u.\) If we spell this out (for \(y \leq x\)):

\[
\begin{align*}
(yc', \text{valof } c) \in a \text{ and } c \in A(x) \Rightarrow a^+(xc') &= \text{valof } c \\
(yc', \text{output } v') \in a \Rightarrow a^+(xc') &= \text{output } v'
\end{align*}
\]

\(\leadsto\) **“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 $M''$:

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

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

Taking the syntax for PCF types \( \sigma ::= \text{nat} \mid \sigma \rightarrow \sigma \), we have the following typing rules:

\[
\Gamma, x_1 : \sigma_1, \ldots x_n : \sigma_n \vdash W : \text{nat} \\
\Gamma \vdash \lambda x_1 \ldots x_n. W : \sigma_1 \rightarrow \ldots \rightarrow \sigma_n \rightarrow \text{nat} \\
\ldots \Gamma, x : \sigma \vdash M_i : \sigma_i \ldots \ldots \Gamma, x : \sigma \vdash W_j : \text{nat} \ldots \]
\[ \Gamma \vdash n : \text{nat} \quad \Gamma, x : \sigma \vdash \text{case } xM_1 \ldots M_p [m_1 \rightarrow W_1 \ldots m_q \rightarrow W_q] : \text{nat} \]

where, in the last rule, \( \sigma = \sigma_1 \rightarrow \ldots \rightarrow \sigma_p \rightarrow \text{nat} \)
PCF Böhm trees as strategies: an example

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

$$[[A^1 \rightarrow \ldots \rightarrow A^n \rightarrow \text{nat}]]_u = [[A^1]]_{u_1} \rightarrow \ldots \rightarrow [[A^n]]_{u_n} \rightarrow \text{nat}_u$$

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 $\lambda f.\, \text{case } f\, 3 \, [4 \rightarrow 7, \, 6 \rightarrow 9]$ reads as follows:

$$
\lambda f.\, \text{case } f \begin{cases} 
(3) & 4 \rightarrow 7 \\
6 \rightarrow 9 
\end{cases} \quad h = ?_\epsilon[?_1, \leftarrow] \begin{cases} 
?_{11}[3_{11}, \leftarrow] & 0 \\
4_1[7_\epsilon, \leftarrow] & 1 \\
6_1[9_\epsilon, \leftarrow] & 1 
\end{cases}
$$
PCF Böhm trees as strategies: full compilation

We need an auxiliary functions

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

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

We translate \( M : A \) to \( \llbracket M \rrbracket^1 \), where

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

\[
\llbracket n \rrbracket_u^L = n_u \quad \text{(pointer reconstructed by well-bracketing)}
\]

\[
\llbracket \text{case } x\vec{M} [\ldots m \to W_m \ldots] \rrbracket_u^L = [?_{vj}, i \leftarrow] \begin{cases} \cdots \llbracket M_l \rrbracket_{vj l} \\ \cdots \llbracket W_m \rrbracket_u^L \\ \cdots \end{cases}
\]

where \( \text{access}(x, L, 0) = [?_{vj}, i \leftarrow] \) and \( 1 \leq l \leq \text{arity}(A, vj) \).
An example of execution of HO strategies: the strategies

\[ K_{\text{Kierstead}1} = \lambda f. \text{case } f(\lambda x. \text{case } f(\lambda y. \text{case } x)) \]

applied to

\[ \lambda g. \text{case } g(\text{case } gT [T \rightarrow T, F \rightarrow F]) [T \rightarrow F, F \rightarrow T] \]
An example of execution of HO strategies: the execution

\[
\langle ?, 1 \rangle [?, 0] \\
\quad \left\{ \begin{array}{l}
\langle ?, 3 \rangle [?, 1] \\
\quad \left\{ \begin{array}{l}
\langle ?, 5 \rangle [?, 1] \\
\quad \left\{ \begin{array}{l}
\langle T_{111}, 15 \rangle [T_{11}, 1] \\
\quad \langle F_1, 17 \rangle [F_{11}, 1] \\
\quad \langle ?_{11}, 9 \rangle [?, 1] \\
\quad \left\{ \begin{array}{l}
\langle F_{111}, 11 \rangle [F_{11}, 1] \\
\quad \langle T_1, 13 \rangle [T_{11}, 1] \\
\quad \langle T_1, 19 \rangle [T_{\epsilon}, 1]
\end{array} \right.
\end{array} \right.
\end{array} \right.
\end{array}
\right.
\end{array} \right.
\end{array} \right.
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
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 \( \lambda f. \text{case } f \ T \ F \ [T \rightarrow F] \)
— Sequential algorithms have memory (or work in call-by-need manner), e.g. the model “normalises”

\[ \lambda x. \text{case } x \ [3 \rightarrow \text{case } x \ [3 \rightarrow 4]] \]

into

\[ \text{request } ?_\epsilon \ \text{valof } ?_1 \ \{ \text{is } 3_1 \ \text{output } 4_\epsilon \] 

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).