Towards a theory of computational complexity for concurrent processes

Damiano Mazza
Laboratoire d'Informatique de Paris Nord
CNRS–Université Paris 13

PANDA meeting
École polytechnique, 24 November 2011

Joint work with Ugo Dal Lago (Bologna), Tobias Heindel (CEA), and Daniele Varacca (PPS)

From functions to behaviors

- The original perspective of recursion theory:

  a program computes a function.

- Albeit enormously complex, an operating system is still a program. What function does it compute?

- The more modern perspective of concurrency theory:

  a program has a behavior.

Formalizing behaviors: labelled transition systems

- A widely accepted way of formalizing the notion of "behavior" is that of *labelled transition system*.

- Labels are of the form

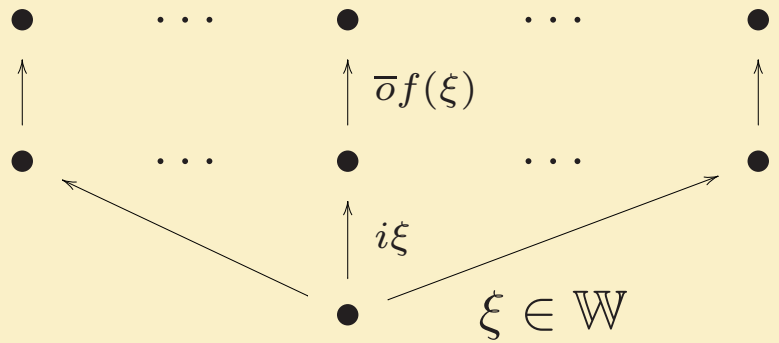
  – $i\xi$ (input action)
  – $\overline{o}\xi$ (output action)
  – τ (internal action)

  where i, o range over *channels* and ξ may in principle belong to any set. We restrict to $\xi \in \mathbb{W} = \{0,1\}^*$. As usual, $|\xi|$ is the length of ξ.

- We formally define a *behavior* as an equivalence class (with respect to *coupled similarity*) of labelled transition systems.

Functions as behaviors

- Of course every function $f : \mathbb{W} \to \mathbb{W}$ may be seen as a behavior:



  which we may write in CCS notation as $i(x).\overline{o}\langle f(x)\rangle$.

- Needless to say, this is far from exhausting all possible behaviors. . .

Computing behaviors

- Not every function is computable; of course, neither is every behavior.
- Process calculi may be used to define a notion of *effective behavior*, in analogy with recursive functions.
- However, process calculi are not quite *computational models*. In particular, they lack a clear *resource semantics*, or *cost model*.
- Our proposal is to define a sensible *process machine*.

A process calculus

- String and boolean expressions ($x \in \mathcal{V}$ and $\xi \in \mathbb{W}$):

$$\begin{array}{rcl} E & ::= & x \mid \xi \mid \mathsf{0}(E) \mid \mathsf{1}(E) \mid \mathrm{tail}(E) \\ B & ::= & \mathrm{tt} \mid \mathrm{ff} \mid \mathsf{0}_?(E) \mid \varepsilon_?(E). \end{array}$$

- Processes (O (resp. I) is either E or an output (resp. input) channel):

$$P, Q \quad ::= \quad \mathbf{0} \mid A\langle E_1, \ldots, E_n \rangle \mid \overline{O}\langle E \rangle.P \mid I(E).P \mid B.(P, Q) \mid P \mid Q,$$

  where A ranges over *process variables*, each with a definition of the form

$$A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P.$$

The process machine

- A finite, read-only *code memory*, where a finite sequence of definitions $A_0 \stackrel{\text{def}}{=} P_0, A_1(\vec{x}_1) \stackrel{\text{def}}{=} P_1, \ldots, A_n(\vec{x}_n) \stackrel{\text{def}}{=} P_n$ is stored;

- an unbounded number of *processors*, each with a *pointer register*, pointing to the code memory, and an unbounded *private memory* in which assignments of the form $x := \xi$ are stored;

- an unbounded number of *internal channels*, each identified by a string and with an unbounded capacity to queue up strings, through which processors may communicate;

- an *interface*, with separate input and output channels, from which strings may be read/sent from/to the external world.

States

- We may represent the state of a processor of the machine by a pair (P, M) where P is a process and M a finite function $\mathcal{V} \to \mathbb{W}$.

- The state of channels may be represented by a finite *queue function* Θ from $\mathbb{W}$ to finite sequences of $\mathbb{W}$.

- Then, a *state* of the machine is represented by a pair

$$[\Gamma]\Theta$$

  where $\Gamma = (P_1, M_1), \ldots, (P_n, M_n)$ is a finite multiset of processor states (the active processors) and Θ is a queue function.

Executing processes (1)

Nil: $[(\mathbf{0}, M), \Gamma]\Theta \xrightarrow{\tau} [\Gamma]\Theta$;

Rec: if $A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P$,

$$[(A\langle E_1, \ldots, E_n\rangle, M), \Gamma]\Theta \xrightarrow{\tau} [(P, \{x_1 \mapsto E_1^M, \ldots, x_n \mapsto E_n^M\}), \Gamma]\Theta;$$

Snd: $[(\overline{E_1}\langle E_2\rangle.P, M), \Gamma]\Theta \xrightarrow{\tau} [(P, M), \Gamma]\Theta'$,
where Θ' is equal to Θ everywhere except in E_1^M, where it is equal to $\Theta(E_1^M) \cdot E_2^M$;

Rcv: $[(E(x).P, M), \Gamma]\Theta \xrightarrow{\tau} [(P, M \cup \{x \mapsto \xi\}), \Gamma]\Theta'$,
where we must have $\Theta(E^M) = \xi \cdot q$, and then Θ' is equal to Θ everywhere except in E^M, where it is equal to q;

Executing processes (2)

Out: $[(\overline{o}\langle E\rangle.P, M), \Gamma]\Theta \xrightarrow{\overline{o}E^M} [(P, M), \Gamma]\Theta$;

Inp: $[(i(x).P, M), \Gamma]\Theta \xrightarrow{i\xi} [(P, M \cup \{x := \xi\}), \Gamma]\Theta$;

Cnd: $[(B.(P, Q), M), \Gamma] : \Theta \xrightarrow{\tau} \begin{cases} [(P, M), \Gamma] : \Theta, & \text{if } B^M = \mathrm{tt} \\ [(Q, M), \Gamma] : \Theta, & \text{if } B^M = \mathrm{ff} \end{cases}$

Spn: $[(P \mid Q, M), \Gamma] : \Theta \xrightarrow{\tau} [(P, M), (Q, M), \Gamma] : \Theta$.

Traces

- Observe how, given a process P, the transitions of the machine from $[(P, \emptyset)]\emptyset$ define a labelled transition system $[P]$. With this, we may define effective behaviors.

- Moreover, we may define a *trace* as a sequence of transitions.

- Two transitions are *orthogonal* if they are not both *send*, *receive*, *output*, or *input* transitions on the same channel.

- Orthogonal transitions *commute*, defining an equivalence relation $\sim$ on traces. This will be essential for defining the *temporal* cost model of our machine.

Causality

- Trace equivalence yields an *event structure*. In particular, we have a pre-order $\lesssim$ between traces ($f \lesssim g$ if $g \sim fg'$), which becomes a *causal order* $\leq$ between equivalence classes of traces.

- An equivalence class represents a transition t when all of its traces are of the form ft and, for every trace g, $g \sim ft$ implies $g = g't$ with $g' \sim f$.

- Given a transition t, we may speak of the transitions which are *causally necessary* for it to occur, *i.e.*, $t{\downarrow} = \{t' \mid t' \leq t\}$.

What is space?

- Let $[(P_1, M_1), \ldots, (P_n, M_n)]\Theta$ be a state of the machine. Its *size* is the sum of the sizes of all strings appearing in the codomain of $M_1, \ldots, M_n$ and Θ (counting multiplicities).

- If f is a trace, we define $\mathrm{space}(f)$ to be the maximum size of the states visited by f.

- $f \sim f'$ implies $\mathrm{space}(f) = \mathrm{space}(f')$.

- Therefore, the *space cost* $\mathrm{space}(t)$ of a transition t is uniquely defined as $\mathrm{space}(ft)$, where ft is any trace in the equivalence class representing t.

What is time?

- Each transition t is assigned a *weight* $\$t$:

  Nil: constant;
  Rec: the time it takes to evaluate the expressions, plus a constant;
  Snd,Rcv,Out,Inp: the time it takes to evaluate the expressions, plus the length of the string sent/received, plus a constant;
  Cnd: the time it takes to evaluate the Boolean expression, plus a constant;
  Spn: the time it takes to duplicate the memory, plus a constant.

- The *time cost* of a transition t is

$$\mathrm{time}(t) = \max_{C \in \mathrm{tot}(t\!\downarrow)} \sum_{t' \in C} \$t'.$$

  where $\mathrm{tot}(t\!\downarrow)$ denotes the set of all chains below t.

Input size

- Let t be a transition. We define $\mathrm{inp}(t)$ to be the subset of $t{\downarrow}$ consisting all transitions whose label is of the form $i\xi$. We say that the *size* of such an input transition t' is $|\xi|$, and we denote it by $|t'|$.

- We define the *input size* of a transition t as

$$\|t\| = \sum_{t' \in \mathrm{inp}(t)} |t'|.$$

Complexity classes

- Let f, g be proper complexity functions.
- We define $\mathbf{BTS}(f, g)$ as the set of behaviors $\mathfrak{b}$ such that:
  - there exists a process P such that $[P] \in \mathfrak{b}$;
  - for every output transition t generated in the execution of P,

$$\begin{aligned} \mathrm{time}(t) &\leq f(\|t\|), \\ \mathrm{space}(t) &\leq g(\|t\|). \end{aligned}$$

Sanity check: the functional case

- We denote by FUN the set of functional behaviors. If $\mathfrak{b} \in \mathrm{FUN}$, we may associate with it a language (subset of $\mathbb{W}$) $\mathrm{lang}\mathfrak{b}$. Then, we define

$$\mathbf{FUNTS}(f,g) = \mathrm{lang}(\mathbf{BTS}(f,g) \cap \mathrm{FUN}).$$

- Some results:
  - $\mathbf{TIME}(f(n)) \subseteq \mathbf{FUNTS}(f(n), f(n))$;
  - $\mathbf{FUNTS}(f(n), g(n)) \subseteq \mathbf{TIME}(f(n)(g(n)+n)^2)$;
  - $\mathbf{ATIME}(f(n)) \subseteq \mathbf{FUNTS}(f(n), 2^{f(n)})$.

- These imply:
  - $\mathbf{P} = \bigcup_{k<\omega} \mathbf{FUNTS}(n^k, n^k)$, $\mathbf{EXP} = \bigcup_{k<\omega} \mathbf{FUNTS}(2^{n^k}, 2^{n^k})\ldots$
  - $\mathbf{PSPACE} \subseteq \bigcup_{k<\omega} \mathbf{FUNTS}(n^k, 2^{n^k})$.

Perspectives

- The relationship with standard (functional) complexity classes must be explored further. We know that, at present, we can only hope to capture superlinear time and space (*i.e.*, we cannot capture $\mathbf{NC}$ or $\mathbf{L}$).

- However, our framework is extremely flexible; we have used very little of it for the moment. For instance, for what concerns extra-functional behaviors, we know that the way the complexity is to be measured must be given *with the behavior*.

- This allows to capture some interesting extra-functional behaviors, such as *servers* and *streams*.