Concurrency 7

Expressive Power of CCS
The $\pi$-calculus

Catuscia Palamidessi
catuscia@lix.polytechnique.fr
The Expressive Power of CCS

• CCS is a Turing-complete formalism. We will show this by proving that we can simulate in CCS the Random Access Machines, which are a Turing-complete formalism.

• Definition: A RAM is a computational model composed by:
  - a finite set of registers $r_1, ..., r_n$ which store natural numbers, one for each register, and can be updated (incremented or decremented) and tested for zero.
  - A program $(1,I_1), ..., (m,I_m)$, where the $I_j$'s are instructions of either of the following two forms:
    - $\text{Incr}(r_j)$: add 1 to Register $r_j$
    - $\text{DecJump}(r_j,s)$: if the content of the register $r_j$ is not zero, then decrease $r_j$ by one and go to next instruction. Otherwise jump to instruction $s$. 

13 Novembre 2003  Concurrency 7
The Expressive Power of CCS

• The state of a RAM $R$ is a tuple $(j, k_1,\ldots,k_n)$ where $j$ is the index of the current instruction and $k_1,\ldots,k_n$ are the contents of the registers.

• The execution is defined by a transition relation among states:
  $$(j, k_1,\ldots,k_n) \rightarrow_R (j', k'_1,\ldots,k'_n)$$

  Meaning that the RAM goes from state $(j, k_1,\ldots,k_n)$ to state $(j', k'_1,\ldots,k'_n)$ by executing the action $I_j$ in the program of $R$.

• We assume that the execution terminates if a special instruction index is reached. We also assume that the first register $(r_1)$ will initially contain the input of the program, and that it will contain the output when the program terminates.
The Expressive Power of CCS

• **Theorem**: Every computable function can be expressed as the input-output relation computed by a RAM.

• We define now a CCS process which encodes a given RAM \( R \).
  
  - Each register is encoded by a labeled instance of a counter: \( C^{(k)}_h \) repr. \( r_h \) with content \( k \)
  
  - The program of \( R \) is encoded as a set of CCS definitions:
    
    \[
    \text{Instr}_j \equiv \begin{cases} \text{inc}_h \cdot \text{Instr}_{i+1} & \text{if } I_i = \text{Succ}(r_h) \text{ in } R \\ \text{dec}_h \cdot \text{Instr}_{i+1} + \text{zero}_h \cdot \text{Instr}_s & \text{if } I_i = \text{DecJump}(r_h,s) \text{ in } R \end{cases}
    \]

  - Assume input \( k \), so the initial configuration is \((1,k,0,...,0)\). The CCS process encoding \( R \) is:
    
    \[
    [(1,k,0,...,0)]_R = (\nu \text{inc})(\nu \text{dec})(\nu \text{zero}) (\text{Instr}_1 | C^{(k)}_1 | C^{(0)}_2 | ... | C^{(0)}_n )
    \]

    where \( \text{inc}, \text{dec} \) and \( \text{zero} \) represent the vectors of the \( \text{inc}_h, \text{dec}_h \) and \( \text{zero}_h \).

• **Theorem (correctness of the encoding)**:
  
  \((j, k_1,...,k_n) \rightarrow R^* (j', k'_1,...,k'_n)\) if and only if \([ (j, k_1,...,k_n) ]_R \xrightarrow{\tau^*} [ (j', k'_1,...,k'_n) ]_R\)

  **Proof**: Exercise
The \( \pi \)-calculus

- Milner, Parrow, Walker 1989
- A concurrent calculus where the communication structure among existing processes can change over time.
  - Link mobility.
The \( \pi \) calculus: scope extrusion

- A private channel name can be communicated and its scope can be extended to include the recipient
  - \textit{Channel}: the name can be used to communicate
  - \textit{Privacy}: no one else can interfere

- An example of link mobility:
The \( \pi \) calculus: scope extrusion

- A private channel name can be communicated and its scope can be extended to include the recipient
  - Channel: the name can be used to communicate
  - Privacy: no one else can interfere

- An example of link mobility:
The π calculus: scope extrusion

- A private channel name can be communicated and its scope can be extended to include the recipient
  - **Channel**: the name can be used to communicate
  - **Privacy**: no one else can interfere

- An example of link mobility:
The $\pi$ calculus: syntax

- Similar to CCS with value passing, but values are channel names, and recursion is replaced by replication ($!$)

\[
\pi ::= x(y) \mid xy \mid \tau
\]

- action prefixes (input, output, silent)
- $x, y$ are channel names

\[
P ::= 0 \quad \text{inaction}
\mid \pi . P \quad \text{prefix}
\mid P \parallel P \quad \text{parallel}
\mid P + P \quad \text{sum}
\mid (\nu x) P \quad \text{restriction, new name}
\mid ! P \quad \text{replication}
\]
Operational semantics (basic idea)

- Transition system \( P \xrightarrow{\mu} Q \)
  where \( \mu \) can be \( x(y), xy, x(y), \) or \( \tau \)

- Rules

  Input \( x(y) \cdot P \xrightarrow{x(z)} P[z/y] \)

  Output \( xy \cdot P \xrightarrow{xy} P \)

  Open \( P \xrightarrow{xy} P' \)

\[(\forall y) P \xrightarrow{x(y)} P'\]
Operational semantics (basic idea)

\[
P \xrightarrow{x(y)} P' \quad Q \xrightarrow{x(y)} Q'
\]

\[
P \mid (\nu y) Q \xrightarrow{\tau} (\nu y) (P' \mid Q')
\]