Concurrency 7

Expressive Power of CCS

The π-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 $r_1,\dots,r_n$ which store natural numbers, one for each register, and can be updated (incremented or decremented) and tested for zero.
  - A program $(1,I_1), \dots ,(m,I_m)$, where the I_j's are instructions of either of the following two forms:
    - $Incr(r_j)$: add 1 to Register r_j
    - $DecJump(r_j,s)$: if the content of the register r_j is not zero, then decrease r_j by one and go to next intruction. Otherwise jump to instruction s.

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: $C_h^{(k)}$ repr. r_h with content k

  - The program of R is encoded as a set of CCS definitions:

    $Instr_j \equiv \underline{inc}_h.Instr_{i+1}$ if $I_i = Succ(r_h)$ in R

    $Instr_j \equiv \underline{dec}_h.Instr_{i+1} + \underline{zero}_h.Instr_s$ if $I_i = DecJump(r_h,s)$ in R

  - Assume input k, so the initial configuration is (1,k,0,...,0). The CCS process encoding R is:

    $[\ (1,k,0,\dots,0)\ ]_R = (\nu\ \mathbf{inc})(\nu\ \mathbf{dec})(\nu\ \mathbf{zero})\ (Instr_1 \mid C_1^{(k)} \mid C_2^{(0)} \mid \dots \mid C_n^{(0)}\ )$

    where **inc**, **dec** and **zero** represent the vectors of the $\underline{inc}_h$, $\underline{dec}_h$ and $zero_h$.

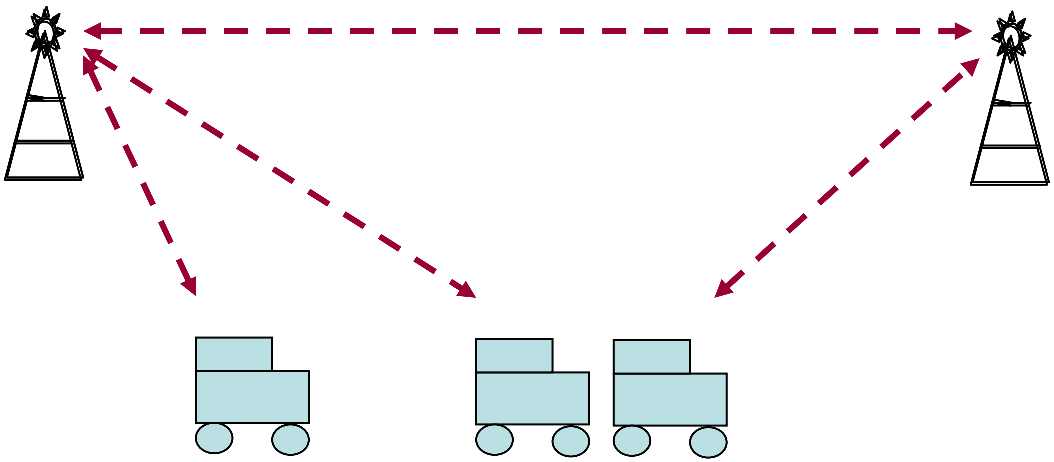
- **Theorem (correctness of the encoding):**

  $(j, k_1,\dots,k_n) \rightarrow_R^* (j', k'_1,\dots,k'_n)$ if and only if $[(j, k_1,\dots,k_n)]_R \stackrel{\tau}{\rightarrow}^* [(j', k'_1,\dots,k'_n)]_R$

  Proof: Exercise

The π-calculus

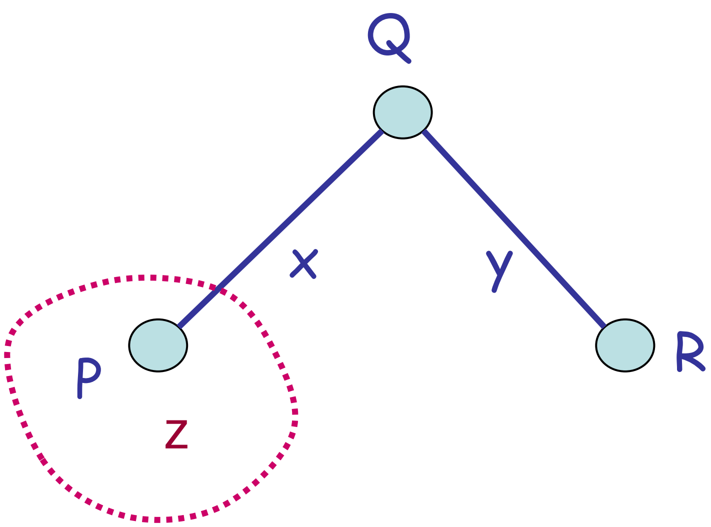
- Milner, Parrow, Walker 1989
- A concurrent calculus where the communication structure among existing processes can change over time.
  - Link mobility.



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

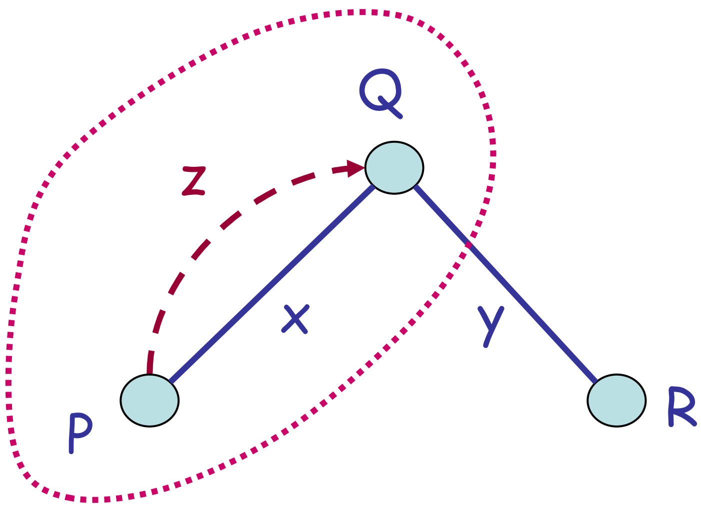
- An example of link mobility:



The 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

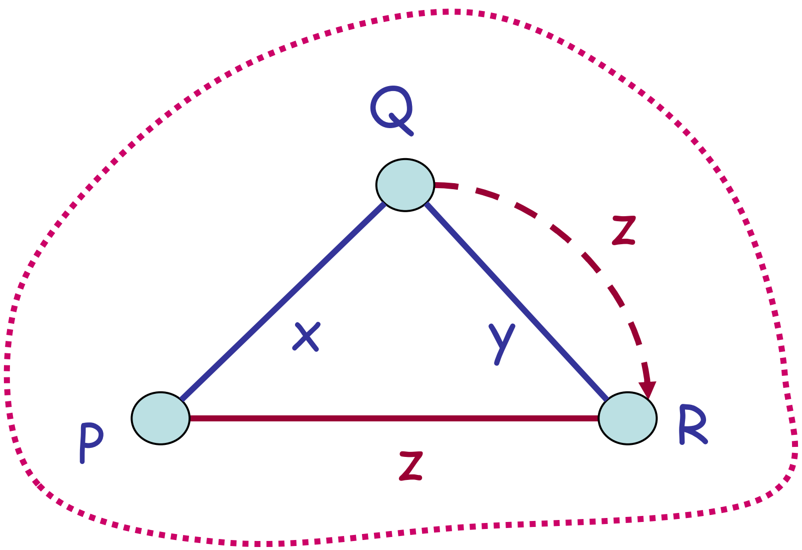
- An example of link mobility:



The 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 π calculus: syntax

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

π ::= $x(y) \mid \underline{x}y \mid \tau$ — action prefixes (input, output, silent)

x, y are channel names

P ::=	0	inaction
|	$\pi . P$	prefix
|	$P \mid P$	parallel
|	$P + P$	sum
|	$(\nu\, x)\, P$	restriction, new name
|	$!\, P$	replication

Operational semantics (basic idea)

- Transition system $P \xrightarrow{\mu} Q$
  where μ can be $x(y)$, $\underline{x}y$, $\underline{x}(y)$, or τ
- Rules

Input $x(y) \,.\, P \xrightarrow{x(z)} P[z/y]$

Output $\underline{x}y \,.\, P \xrightarrow{\underline{x}y} P$

Open
$$\frac{P \xrightarrow{\underline{x}y} P'}{(\nu\, y)\, P \xrightarrow{\underline{x}(y)} P'}$$

Operational semantics (basic idea)

Close

$$\frac{P \xrightarrow{x(y)} P' \qquad Q \xrightarrow{\underline{x}(y)} Q'}{P \mid (\nu\, y)\, Q \xrightarrow{\tau} (\nu\, y)\, (P' \mid Q')}$$