The pi-Calculus (Part 2)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
http://www.risc.uni-linz.ac.at

Process Commitments

When can a reaction $P|Q \to P'|Q'$ occur?

- CCS: $P|Q = (\cdots + a.P')|(\cdots + \bar{a}.Q')$
  - Transitions $P \xrightarrow{a} P'$ and $Q \xrightarrow{\bar{a}} Q'$ are necessary.
    - $P \xrightarrow{\alpha} P'$ … "commitment" of P to take part in a reaction involving α.
  - Residue of reaction is $P'|Q'$.
- π-Calculus: $P|Q = (\cdots + x(\vec{y}).P')|(\cdots + \bar{x}\langle\vec{z}\rangle.Q')$
  - Commitments $P \xrightarrow{x} (\vec{y}).P'$ and $Q \xrightarrow{\bar{x}} \langle\vec{z}\rangle.Q'$ are necessary.
    - Abstraction $(\vec{y}).P'$, concretion $\langle\vec{z}\rangle.Q'$.
  - Residue of reaction is $(\vec{y}).P'@\langle\vec{z}\rangle.Q' = \{\vec{z}/\vec{y}\}P'|Q'$.
    - $F@C$ is the application of an abstraction F and a concretion C.

Revise transition rules of CCS to commitment rules of the π-calculus.

Abstractions and Concretions

- Abstraction: $(\vec{x}).P$
  - Bound names $\vec{x}$, process part P.
  - If $|\vec{x}| = n$, then the abstraction has arity n.
  - Two abstractions F and G are structurally congruent ($F \equiv G$), if they have same arity and, up to alpha-conversion of the bound names, their process parts are structurally congruent.
- Concretion: new $\vec{x}$ $\langle\vec{y}\rangle.P$
  - Restricted names $\vec{x}$, prefix new $\vec{x}$ $\langle\vec{y}\rangle$, process part P.
  - If $|\vec{y}| = n$, then the concretion has arity n.
  - Two concretions C and D are structurally congruent ($C \equiv D$), if they have the same arity and, up to alpha-conversion and re-ordering of the restricted names, their prefixes are identical and their process parts are structurally congruent.

Concretions may include restrictions to keep names secret between the receiver of a message and (the residue of) the sender.

Agents

- Agent: an abstraction or a concretion.
  - Every process is an abstraction and a concretization of arity 0.
  - Every process is thus an agent.
- Generalized Summation: $\sum_{i\in I} \alpha_i A_i$
  - $\alpha_i A_i$ has form xF, $\bar{x}C$, or τP.
    - Output action may include a restriction.
    - Example: $\bar{x}(\text{new } y_1 \langle y_1 y_2\rangle.P)$
- Generalized Reaction Rule:

  REACT $(xF+M)|(\bar{x}C+N) \to F@C$

  - Application $F@C$ yields a process:

    $(\vec{x}).P \;@\; \text{new } \vec{z}\langle\vec{y}\rangle.Q := \text{new } \vec{z}(\{\vec{y}/\vec{x}\}P|Q)$

    - $|\vec{x}| = |\vec{y}|$
    - $\vec{z}$ not free in $(\vec{x}).P$.

The commitment rules operate on agents.

Generalized Process Operations

Agents (new z A) and $A|Q$ of same kind and arity as agent A.

- Generalized Restriction:

  We assume $z \notin \vec{x}$.
  - $\text{new } z\ ((\vec{x}).P) := (\vec{x}).\text{new } z\ P$
  - $\text{new } z\ (\text{new } \vec{x}\ \langle\vec{y}\rangle.P) := \begin{cases} \text{new } z\vec{x}\ \langle\vec{y}\rangle.P & \text{if } z \in \vec{y} \\ \text{new } \vec{x}\ \langle\vec{y}\rangle.\text{new } z\ P & \text{otherwise} \end{cases}$
- Generalized Composition:

  We assume $z \notin \vec{x}$ and $\vec{x}$ not free in Q.
  - $((\vec{x}).P)|Q := (\vec{x}).(P|Q)$
  - $(\text{new } \vec{x}\ \langle\vec{y}\rangle.P)|Q := \text{new } \vec{x}\ \langle\vec{y}\rangle.(P|Q)$
- Structural Congruences:
  - $A|(P|Q) \equiv (A|P)|Q$
  - $\text{new } x\ (A|Q) \equiv \begin{cases} A \mid \text{new } x\ Q & (x \text{ not free in } A) \\ \text{new } x\ A \mid Q & (x \text{ not free in } Q) \end{cases}$
  - $(F|P)@(C|Q) \equiv (F@C)|P|Q$
  - $\text{new } x\ (F@C) \equiv \begin{cases} F@\text{new } x\ C & (x \text{ not free in } F) \\ \text{new } x\ F@C & (x \text{ not free in } C) \end{cases}$

Commitment Rules

- Commitment Relation $\xrightarrow{\alpha}$: Set of those commitments that can be inferred from the following rules, together with alpha-conversion (where α is either a label or τ):

SUM$_c$ $\quad M + \alpha A + N \xrightarrow{\alpha} A$

L-REACT$_c$ $\quad \dfrac{P \xrightarrow{x} F \quad Q \xrightarrow{\bar{x}} C}{P|Q \xrightarrow{\tau} F@C}$ $\qquad$ R-REACT$_c$ $\quad \dfrac{P \xrightarrow{\bar{x}} C \quad Q \xrightarrow{x} F}{P|Q \xrightarrow{\tau} F@C}$

L-PAR$_c$ $\quad \dfrac{P \xrightarrow{\alpha} A}{P|Q \xrightarrow{\alpha} A|Q}$ $\qquad$ R-PAR$_c$ $\quad \dfrac{Q \xrightarrow{\alpha} A}{P|Q \xrightarrow{\alpha} P|A}$

RES$_c$ $\quad \dfrac{P \xrightarrow{\alpha} A}{\text{new } x\ P \xrightarrow{\alpha} \text{new } x\ A}$ if $(\alpha \notin \{x, \bar{x}\})$

REP$_c$ $\quad \dfrac{P \mid !P \xrightarrow{\alpha} A}{!P \xrightarrow{\alpha} A}$

Relationships

- Structural Congruence Respects Commitment: If $P \xrightarrow{\alpha} A$ and $P \equiv Q$, then there exists some B such that $Q \xrightarrow{\alpha} B$ and $A \equiv B$.
  - Structurally congruent processes have the same commitments.
- Reaction Agrees with τ-Commitment: $P \to P'$ if and only if there exists some P'' such that $P \xrightarrow{\tau} P''$ and $P'' \equiv P'$.
  - $\to$ corresponds to the silent commitment $\xrightarrow{\tau}$ (modulo congruence).

Analogous to the relationships in CCS.

Agent Congruence

- Agent congruence: An equivalence relation $\simeq$ on agents is an agent congruence, if the following holds:
  1. If $A \simeq B$, then $\alpha A + M \simeq \alpha B + M$ (for any process M)
  2. If $P \simeq Q$, then new a $P \ \simeq$ new a Q, $P|R \simeq Q|R$, $R|P \simeq R|Q$, $!P \simeq !Q$, new $\vec{x}\langle\vec{y}\rangle.P \simeq$ new $\vec{x}\langle\vec{y}\rangle.Q$.
  3. If $\{\vec{y}/\vec{x}\}P \simeq \{\vec{y}/\vec{x}\}Q$ for all $\vec{y}$, then $(\vec{x}).P \simeq (\vec{x}).Q$.

The notion is more complex than process congruence, because messages may be passed in reactions.

Agent Relations

Extend relations on processes to relations on agents.

- Application of process relation to abstractions resp. concretions:

  Assume binary relation $\mathcal{S}$ on processes, abstractions F, G, concretions C, D.

  - $F\mathcal{S}G$:

    $F\langle\vec{y}\rangle\mathcal{S}G\langle\vec{y}\rangle$, for all $\vec{y}$.
  - $C\mathcal{S}D$:

    $P\mathcal{S}Q$, for some processes P and Q and $\vec{z}, \vec{y}$ such that $C \equiv$ new $\vec{z}$ $\langle\vec{y}\rangle.P$ and $D \equiv$ new $\vec{z}$ $\langle\vec{y}\rangle.Q$.

Abstractions are only related, if all their instantiations are related!

Strong Equivalence

- Strong simulation: A binary relation $\mathcal{S}$ on processes is a strong simulation, if $P\mathcal{S}Q$ implies
  - if $P \xrightarrow{\alpha} A$, then there exists some B such that $Q \xrightarrow{\alpha} B$ and $A\mathcal{S}B$.
- Strong bisimulation: A binary relation $\mathcal{S}$ on processes is a strong bisimulation, if both S and its converse are a strong simulation.
- Strong equivalence: Two agents A and B are strongly equivalent, written $A \sim B$, if $A\mathcal{S}B$, for some strong bisimulation $\mathcal{S}$.
- Theorem: strong equivalence $\sim$ is an agent congruence.
  - Not a general congruence.
  - $P \simeq Q$ does not imply $z(y).P \simeq z(y).Q$:
    - $\bar{x}|y \sim \bar{x}.y + y.\bar{x}$.
    - $\{x/y\}\bar{x}|y \not\sim \{x/y\}\bar{x}.y + y.\bar{x}$.

Basic Properties of Replication

Strong equivalence does not imply structural congruence.

- Propositions:
  - $!P \ \sim\ !P \mid !P$.
  - $!!P \ \sim\ !P$.
    - But $!P \not\equiv !P|!P$ and $!!P \not\equiv !P$.
- Definition: An agent A is *negative* on x, if its only free occurrences of x are in the form $\bar{x}C$.
  - Cannot receive messages on x or transmit x as a message.
  - If P is negative and $P \xrightarrow{\alpha} A$, then A is negative on x.
- Theorems: Let P, P_1, P_2, F be negative on x.
  1. new x $(P_1 \mid P_2 \mid !xF) \sim$ new x $(P_1 \mid !xF) \mid$ new x $(P_2 \mid !xF)$
  2. new x $(!P \mid !xF) \ \sim$!new x $(P \mid !xF)$
     - P_1 and P_2 cannot communicate on x.
     - new x $(\_ \mid !xF)$ can be pushed inside composition and replication.

1. Strong Equivalence

2. **Observation Equivalence**

3. Extensions of the π-Calculus

Experiments

In CCS, an atomic experiment to distinguish process expressions is represented by the relation $\stackrel{\lambda}{\Rightarrow}$ ($\lambda \in \{a, \bar{a}\}$, for some action a).

- Transition relation $\stackrel{x\langle\vec{y}\rangle}{\rightarrow}$:
  $P \stackrel{x\langle\vec{y}\rangle}{\rightarrow} P'$ iff, for some F, $P \stackrel{x}{\rightarrow} F$ and $F\langle\vec{y}\rangle \equiv P'$.
- Transition relation $\stackrel{x\langle\vec{y}\rangle}{\Rightarrow}$ (the atomic input experiments):
  $P \stackrel{x\langle\vec{y}\rangle}{\Rightarrow} P'$ iff, for some P_0, P_1, $P \Rightarrow P_0 \stackrel{x\langle\vec{y}\rangle}{\rightarrow} P_1 \Rightarrow P'$
- Transition relation $\stackrel{\bar{x}}{\Rightarrow}$ (the atomic output experiments):
  $P \stackrel{\bar{x}}{\Rightarrow}$ new $\vec{z}$ $\langle\vec{y}\rangle.P'$ if, for some P_0, P'',
  $P \Rightarrow P_0 \stackrel{\bar{x}}{\rightarrow}$ new $\vec{z}$ $\langle\vec{y}\rangle.P''$, and $P'' \Rightarrow P'$.

Two relations required for capturing input and output experiments.

Weak Simulations

- Weak Simulation: a binary relation $\mathcal{S}$ on processes is a *weak simulation* if, whenever $P\mathcal{S}Q$, then:
  - if $P \Rightarrow P'$, then there exists Q' such that $Q \Rightarrow Q'$ and $P'\mathcal{S}Q'$;
  - if $P \stackrel{x\langle\vec{y}\rangle}{\Rightarrow} P'$, then there exists Q' such that $Q \stackrel{x\langle\vec{y}\rangle}{\Rightarrow} Q'$, and $P'\mathcal{S}Q'$;
  - if $P \stackrel{\bar{x}}{\Rightarrow} C$, then there exists D such that $Q \stackrel{\bar{x}}{\Rightarrow} D$ and $C\mathcal{S}D$.
- Proposition: a binary relation $\mathcal{S}$ on processes is a weak simulation if, whenever $P\mathcal{S}Q$, then:
  - if $P \rightarrow P'$, then there exists Q' such that $Q \Rightarrow Q'$ and $P'\mathcal{S}Q'$;
  - if $P \stackrel{x\langle\vec{y}\rangle}{\rightarrow} P'$, then there exists Q' such that $Q \stackrel{x\langle\vec{y}\rangle}{\Rightarrow} Q'$, and $P'\mathcal{S}Q'$;
  - if $P \stackrel{\bar{x}}{\rightarrow} C$, then there exists D such that $Q \stackrel{\bar{x}}{\Rightarrow} D$ and $C\mathcal{S}D$.

To establish that $\mathcal{S}$ is a weak simulation, it is only necessary to check single transitions.

Observation Equivalence

- Weak Bisimulation: a binary relation $\mathcal{S}$ on processes is a weak bisimulation, if both $\mathcal{S}$ and its converse are weak simulations.
- Observation Equivalence: two agents A and B are observation equivalent, written $A \simeq B$, if $A\mathcal{S}B$, for some weak bisimulation $\mathcal{S}$.
- Propositions:
  - $P \sim Q$ implies $P \simeq Q$
  - $P \simeq \tau.P$
- Theorem: observation equivalence is an agent congruence.

Processes are not observationally equivalent only if they can be distinguished by some input or output experiment.

Unique Solution of Equations

- Theorem: Let $\vec{X} = X_1, X_2, \ldots$ be a (possibly infinite) sequence of variables over abstractions and let $\vec{F}$ and $\vec{G}$ be two solutions of the system of equations

$$X_1(\vec{x}_1) \simeq \alpha_{11}A_{11} + \cdots + \alpha_{1n_1}A_{1n_1}$$
$$X_2(\vec{x}_2) \simeq \alpha_{21}A_{21} + \cdots + \alpha_{2n_2}A_{2n_2}$$
$$\ldots$$

  where each term αA on the right sides takes one of the forms
  - xH where H is an abstraction $(\vec{v}).X_k\langle\vec{w}\rangle$
  - $\bar{x}C$ where C is a concretion new $\vec{u}\ \langle\vec{v}\rangle.X_k\langle\vec{w}\rangle$

  Then $F_i \simeq G_i$, for all i.

A system of agents is uniquely (up to observation equivalence) defined by a mutually recursive set of agent equations.

1. Strong Equivalence

2. Observation Equivalence

3. **Extensions of the π-Calculus**

A Type System for the π-Calculus

$Control_1 := \overline{lose_1}\langle talk_2, switch_2\rangle.\overline{gain_2}\langle talk_2, switch_2\rangle.Control_2.$
$Control_2 := \overline{lose_2}\langle talk_1, switch_1\rangle.\overline{gain_1}\langle talk_1, switch_1\rangle.Control_1.$

$Trans\langle talk_2, switch_2, gain_2, lose_2\rangle =$
$\quad talk_2.Trans\langle talk_2, switch_2, gain_2, lose_2\rangle +$
$\quad lose_2(t,s).\overline{switch_2}\langle t, s\rangle.Idtrans\langle gain_2, lose_2\rangle.$

- Channel type $lose_1 : \mathrm{CHAN}(\tau, \sigma)$
  - Output action $\overline{lose_1}\langle talk_2, switch_2\rangle$
  - Message types $talk_2 : \tau, switch_2 : \sigma$
- Channel type $switch_2 : \mathrm{CHAN}(\tau, \sigma)$
  - Output action $\overline{switch_2}\langle talk_1, switch_1\rangle$
  - Message types $talk_1 : \tau, switch_1 : \sigma$
- Consequence $\sigma = \mathrm{CHAN}(\tau, \sigma)$
  - A channel may carry messages of its own type.

No hierarchy of channel types possible, we need another approach.

Sorts

- Collection of sorts Σ
  - $x : \sigma$ ... to name x, sort σ is assigned.
    - To every name, a sort is assigned.
    - Every sort is assigned to infinitely many names.
- Set of sort lists Σ^*
  - $\vec{x} : \vec{\sigma}$ ... to name sequence $\vec{x}$, sort list $\vec{\sigma}$ is assigned.
    - $\vec{x} = x_1, \ldots, x_n$, $\vec{\sigma} = \sigma_1, \ldots, \sigma_n$.
    - $x_i : \sigma_i, 1 \leq i \leq n$.
  - $F : \vec{\sigma}$
    - Abstraction $F = (\vec{x})P$
    - $\vec{x} : \vec{\sigma}$

Sorts shall ensure that names are used properly by processes.

Sortings

Take a set Σ of sorts.

- A sorting $ob : \Sigma \to \Sigma^*$ over Σ:
  - Partial function mapping sorts to sort sequences.
- A process (family) respects a sorting ob:
  - For every subterm $x(\vec{y}).P$ or $\overline{x}\langle\vec{y}\rangle.P$ of the process (family):
    if $x : \sigma$, then $\vec{y} : ob(\sigma)$.
- Example: $\Sigma = \{\text{TALK}, \text{SWITCH}, \text{GAIN}, \text{LOSE}\}$
  - $talk_i : \text{TALK}, switch_i : \text{SWITCH}, gain_i : \text{GAIN}, lose_i : \text{LOSE}$
  - This sorting is (the only one) respected by the mobile phone system:

$$ob : \begin{cases} \text{TALK} \mapsto \epsilon \\ \text{SWITCH} \mapsto \text{TALK}, \text{SWITCH} \\ \text{GAIN} \mapsto \text{TALK}, \text{SWITCH} \\ \text{LOSE} \mapsto \text{TALK}, \text{SWITCH} \end{cases}$$

  - We have another respected sorting for $\Sigma = \{\text{TALK}, \text{SWITCH}\}$ (which?)

Sorting and Process Relations

- Proposition: Structural congruence and reaction preserve sortings:
  - If $P \equiv P'$ or $P \to P'$ and P respects ob, then P' respects ob.
  - Sorting thus constrains the pattern of interaction.
- Example: Process $P|Q$ with free names $x : \mathrm{A}, y : \mathrm{B}, u : \mathrm{C}, v : \mathrm{D}$
  - Process respects the sorting

$$ob : \begin{cases} \mathrm{A} \mapsto \mathrm{B} \\ \mathrm{B} \mapsto \epsilon \\ \mathrm{C} \mapsto \mathrm{D} \\ \mathrm{D} \mapsto \epsilon \end{cases}$$

  - Assume P contains only x free.
  - Then P can receive y from Q and use it but it cannot receive u or v.

Sorting assists in analyzing behavior.

Processes as Messages

Why only allow names as messages?

- Extend action prefixes π to transmit processes:
  - Allow process variables $p, q, r, \ldots$ in input actions.
  - Allow process expressions in output actions.
- Example: process R is transmitted.
  - $P = \overline{x}\langle R\rangle.P'$.
  - $Q = x(r).(r|r|Q')$.
  - $P|Q \to P'|R|R|Q'$.
- Translation:
  - $\widehat{P} = (\text{new } z)\overline{x}\langle z\rangle.(!z.R \mid P')$.
  - $\widehat{Q} = x(z).(\overline{z}|\overline{z}|Q')$.
  - $\widehat{P}|\widehat{Q} \Rightarrow (\text{new } z)!z.r|P'|R|R|Q' \sim 0|P'|R|R|Q' \equiv P'|R|R|Q'$.

The higher-order π-calculus can be translated into the plain calculus.

Abstractions as Messages

Transmitted processes may be also parametric.

- Example: abstraction F is transmitted.
  - $P = \overline{x}\langle F\rangle.P'$.
  - $Q = x(f).(f\langle u\rangle|f\langle v\rangle|Q')$.
  - $P|Q \to P'|F\langle u\rangle|F\langle v\rangle|Q'$.
- Translation:
  - $\widehat{P} = (\text{new } z)\overline{x}\langle z\rangle.(!z.F \mid P')$.
  - $\widehat{Q} = x(z).(\overline{z}\langle u\rangle|\overline{z}\langle v\rangle|Q')$.
  - $\widehat{P}|\widehat{Q} \Rightarrow \ldots \sim \ldots \equiv P'|F\langle u\rangle|F\langle v\rangle|Q'$.

Translation takes correctly care of name scopes (free names of abstraction are bound in sender, free names of arguments are bound in receiver).