Communicating and Mobile Systems - the π-calculus

Markus Lumpe

Institute of Computer Science and Applied Mathematics (IAM)

University of Berne

Neubrückstrasse 10, CH-3012 Bern

E-mail: *lumpe@iam.unibe.ch*

WWW: *http://www.iam.unibe.ch/~lumpe*

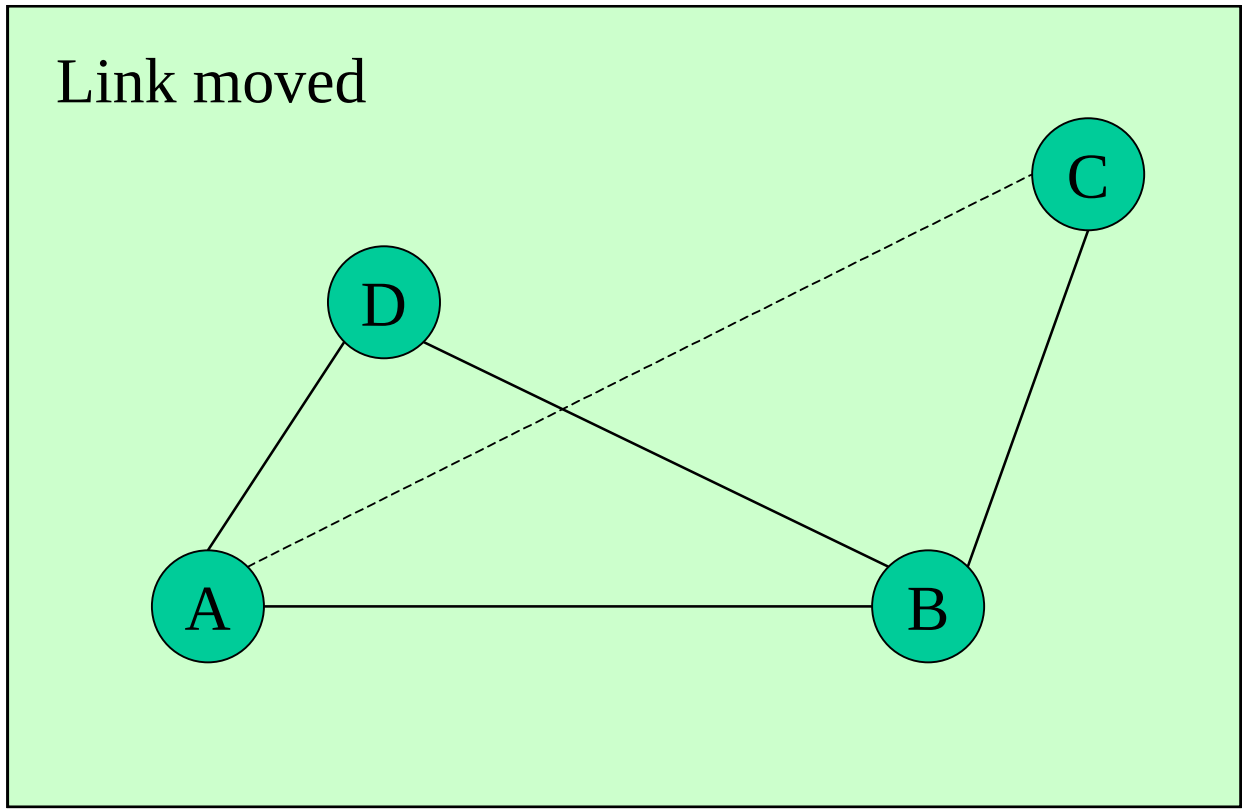
The programming model

- Communication is a fundamental and integral part of computing, whether between different computers on a network, or between components within a single computer.

- Robin Milner's view: Programs are built from communicating parts, rather than adding communication as an extra level of activity.

Programs proceed by means of communication.

Evolving Automata



Automata

Starting point: The components of a system are interacting automata.

An automaton is a quintuple $A = (\Sigma, Q, q_0, \sigma, F)$ with:

- a set Σ of *actions* (sometimes called an alphabet),
- a set $Q = \{ q_0, q_1, \ldots \}$ of *states*,
- a subset F of Q called the *accepting states*,
- a subset σ of $Q \times A \times Q$ called the *transitions*,
- a designated start state q_0.

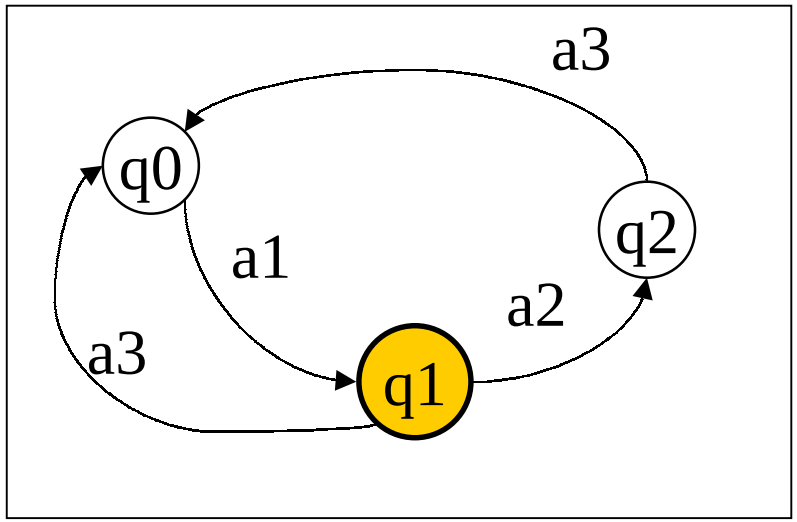
A transition $(q, a, q') \in \sigma$ is usually written $q \xrightarrow{a} q'$.

The automaton A is said to be finite if Q is finite.

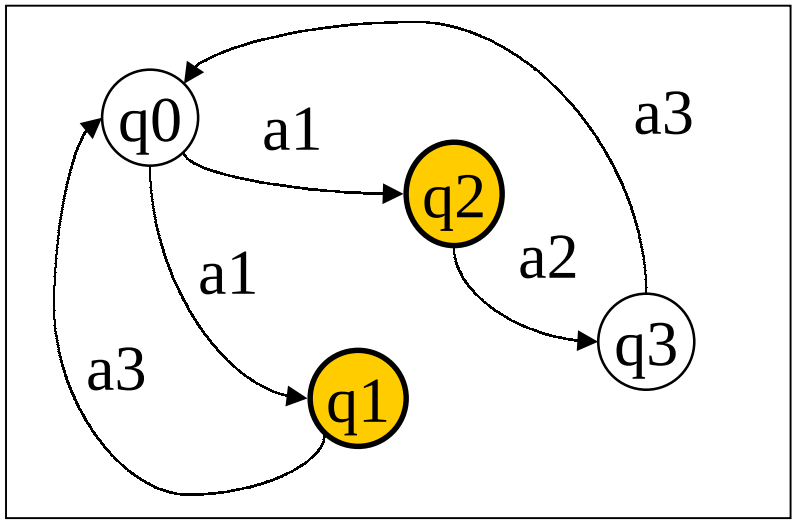
Behaviour of Automata

An automaton is deterministic if for each pair $(q, a) \in Q \times \Sigma$ there is exactly one transition $q \xrightarrow{a} q'$.

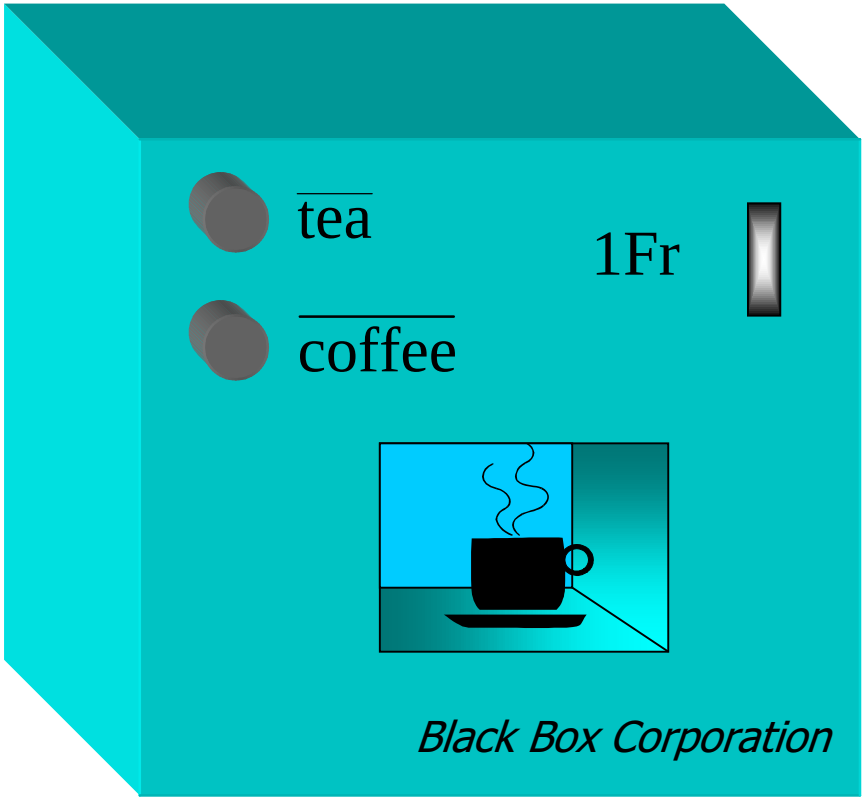
deterministic automata:



non-deterministic automata:



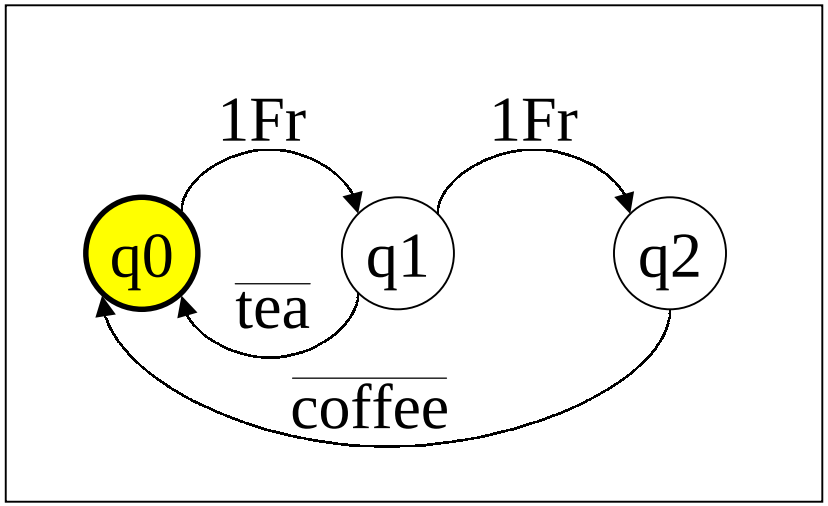
Vending machine



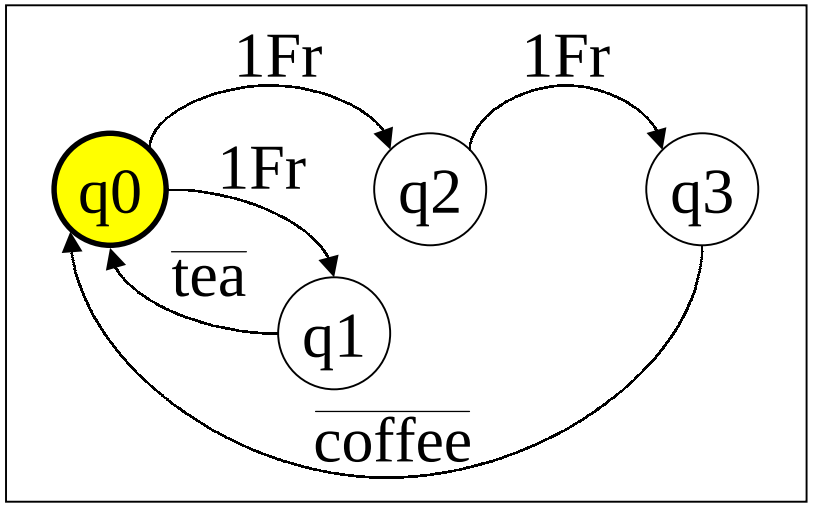
A tea/coffee vending machine is implemented as black box with a three-symbol alphabet $\{1Fr, \overline{tea}, \overline{coffee}\}$.

Internal transition diagrams

Deterministic system S1:



Non-deterministic system S2:



Are both systems equivalent?

S1 = S2?

S1:

$$q0 = 1Fr \cdot q1 + \varepsilon$$
$$q1 = \overline{tea} \cdot q0 + 1Fr \cdot q2$$
$$q2 = \overline{coffee} \cdot q0$$

$$q1 = \overline{tea} \cdot q0 + 1Fr \cdot \overline{coffee} \cdot q0$$
$$q0 = 1Fr \cdot (\overline{tea} \cdot q0 + 1Fr \cdot \overline{coffee} \cdot q0) + \varepsilon$$
$$q0 = 1Fr \cdot (\overline{tea} + 1Fr \cdot \overline{coffee}) \cdot q0 + \varepsilon$$
$$q0 = (1Fr \cdot (\overline{tea} + 1Fr \cdot \overline{coffee}))^*$$

S2:

$$q0 = 1Fr \cdot q1 + 1Fr \cdot q2 + \varepsilon$$
$$q1 = \overline{tea} \cdot q0$$
$$q2 = 1Fr \cdot q3$$
$$q3 = \overline{coffee} \cdot q0$$

$$q2 = 1Fr \cdot \overline{coffee} \cdot q0$$
$$q0 = 1Fr \cdot \overline{tea} \cdot q0 + 1Fr \cdot 1Fr \cdot \overline{coffee} \cdot q0 + \varepsilon$$
$$q0 = 1Fr \cdot (\overline{tea} \cdot q0 + 1Fr \cdot \overline{coffee} \cdot q0) + \varepsilon$$
$$q0 = 1Fr \cdot (\overline{tea} + 1Fr \cdot \overline{coffee}) \cdot q0 + \varepsilon$$
$$q0 = (1Fr \cdot (\overline{tea} + 1Fr \cdot \overline{coffee}))^*$$

The systems S1 and S2 are language-equivalent, but the *observable behaviour* is not the same.

Automata - Summary

Language-equivalence is not suitable for all purposes. If we are interested in interactive behaviour, then a non-deterministic automaton cannot correctly be equated behaviourally with a deterministic one.

A different theory is required!

Labelled transition systems

A labelled transition system over actions *Act* is a pair (Q, T) consisting of:

- a set $Q = \{ q_0, q_1, \ldots \}$ of *states*,
- a ternary relation $T \subseteq (Q \times Act \times Q)$, known as a *transition relation*.

If $(q, \alpha, q') \in T$ we write $q \xrightarrow{\alpha} q'$, and we call q the source and q' the target of the transition.

States and Actions

Important conceptual changes:

- What matters about a string *s* - a sequence of actions - is not whether it drives the automaton into an accepting state (since we cannot detect this by interaction) but whether the automaton is able to perform the sequence of *s* interactively.
- A labelled transition system can be thought of as an automaton without a start or accepting states.
- *Any* state can be considered as the start.

Actions consist of a set L of *labels* and a set $\overline{L}$ of *co-labels with* $\overline{L} = \{\overline{a} | a \in L\}$. We use α, β, … to range over actions *Act*.

Strong Simulation - Idea

- In 1981 D. Park proposed a new approach to define the equivalence of automatons - bisimulation.
- Given a labelled transition system there exists a standard definition of bisimulation equivalence that can be applied to this labelled transition system.
- The definition of bisimulation is given in a *coinductive* style that is, two systems are bisimular if we cannot show that they are not.
- Informally, to say a ‘system S1 simulates system S2’ means that S1’s observable behaviour is at least as rich as that of S2.

Strong Simulation - Definition

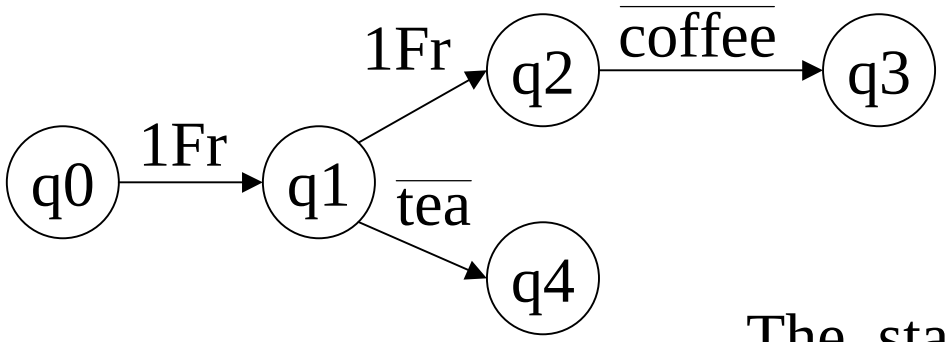
Let (Q, T) be an labelled transition system, and let $\boldsymbol{S}$ be a binary relation over Q. Then S is called a strong simulation over (Q, T) if, whenever p$\boldsymbol{S}$q,

if $p \xrightarrow{\alpha} p'$ then there exists $q' \in Q$ such that $q \xrightarrow{\alpha} q'$ and $p' \boldsymbol{S} q'$.

We say that q strongly simulates p if there exists a strong simulation $\boldsymbol{S}$ such that $p\boldsymbol{S}q$.

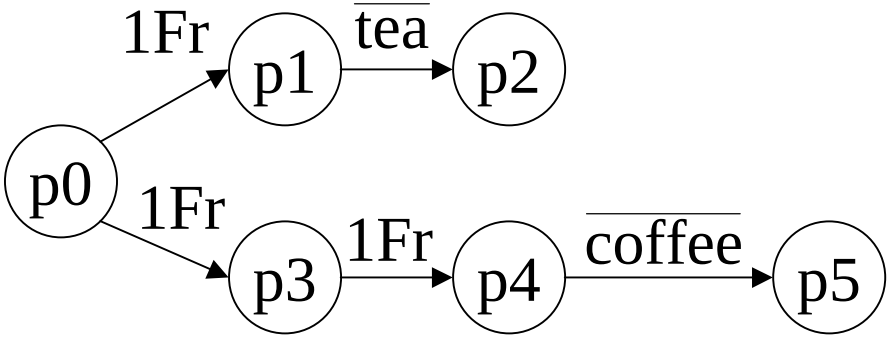
Strong Simulation - Example

S1:



The states q0 and p0 are different. Therefore, the systems S1 and S2 are not considered to be equivalent.

S2:



Strong Simulation - Example II

Define $\boldsymbol{S}$ by

$\boldsymbol{S}$ = {(p0, q0), (p1, q1), (p3, q1), (p2, q4), (p4, q2), (p5, q3)}

then $\boldsymbol{S}$ is a strong simulation; hence q0 strongly simulates p0. To verify this, for every pair $(p, q) \in \boldsymbol{S}$ we have to consider each transition of p, and show that it is properly matched by some transition of q.

However, there exists no strong simulation $\boldsymbol{R}$ that contains the pair (q1, p1), because one of q1's transition could never be matched by p1. Therefore, the states q0 and p0 are different, and the systems S1 and S2 are not considered to be equivalent.

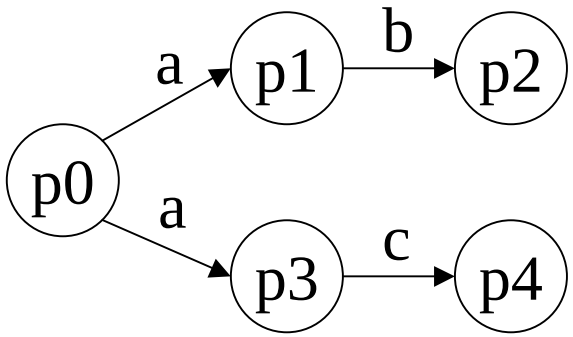
Strong Bisimulation

The converse $\boldsymbol{R}^{-1}$ of any binary relation $\boldsymbol{R}$ is the set of pairs (y, x) such that (x, y) $\in \boldsymbol{R}$.

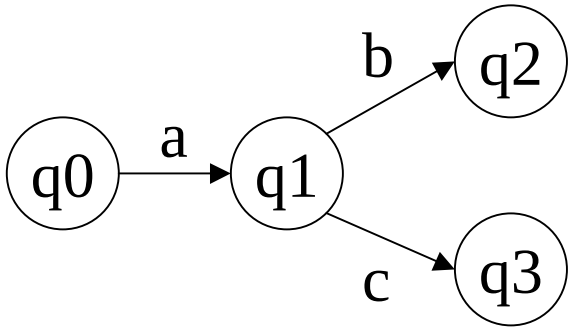
Let (Q, T) be an labelled transition system, and let $\boldsymbol{S}$ be a binary relation over Q. Then $\boldsymbol{S}$ is called a strong bisimulation over (Q, T) if both $\boldsymbol{S}$ and its converse $\boldsymbol{S}^{-1}$ are strong simulations. We say that p and q are strongly bisimular or strongly equivalent, written $p \sim q$, if there exists a strong bisimulation S such that $p\boldsymbol{S}q$.

Checking Bisimulation

S1:



S2:



S1 ~ S2?

To construct $\boldsymbol{S}$ start with (p0, q0) and check whether S2 can match all transitions of S1:

$\boldsymbol{S}$ = { (p0, q0), (p1, q1), (p3, q1), (p2, q2), (p4, q3) }

System S2 can simulate system S1. Now check, whether $\boldsymbol{S}^{-1}$ is a simulation or not:

$\boldsymbol{S}^{-1}$ = { (q0, p0), (q1, p1), (q1, p3), (q2, p2), (q3, p4) }

Start with (q0, p0) $\in \boldsymbol{S}^{-1}$.

1: q0 has one transition ‘a’ that can be matched by two transitions of S1 (target p1 and p3, respectively) and we have (q1, p1) $\in \boldsymbol{S}^{-1}$ and (q1, p3) $\in \boldsymbol{S}^{-1}$.

2: q1 has two transitions ‘b’ and ‘c’, which, however, cannot appropriately be matched by the related states p1 and p3 of system S1 (p1 has only a ‘b’ transition whilst p3 has only a ‘c’ transition).

We have, therefore, S1 $\not\sim$ S2.

Some Facts on Bisimulations

$\sim$ is an equivalence relation.

If $\boldsymbol{S}_i$, $i = 1, 2,...$ is a family of strong bisimulations, then the following relations are also strong bisimulations:

- $Id_{\boldsymbol{P}}$
- $\boldsymbol{S}_1 \circ \boldsymbol{S}_2 = \{(P, Q) \in P \times P$ if R exists with $(P, R) \in \boldsymbol{S}_1, (R, Q) \in \boldsymbol{S}_2 \}$
- $\boldsymbol{S}_i^{-1}$
- $\bigcup_{i \in I} \boldsymbol{S}_i$

Some Facts on Bisimulations II

$\boldsymbol{S}_1 \circ \boldsymbol{S}_2 = \{(P, Q) \in P \times P$ if R exists with $(P, R) \in \boldsymbol{S}_1, (R, Q) \in \boldsymbol{S}_2 \}$

Proof:

Let $(P, Q) \in \boldsymbol{S}_1 \circ \boldsymbol{S}_2$. Then there exists a R with $(P, R) \in \boldsymbol{S}_1$ *and* $(R, Q) \in \boldsymbol{S}_2$.

($\rightarrow$) If $P \xrightarrow{\alpha} P'$, then since $(P, R) \in \boldsymbol{S}_1$ there exists R' and $R \xrightarrow{\alpha} R'$ and $(P', R') \in \boldsymbol{S}_1$. Furthermore, since $(R, Q) \in \boldsymbol{S}_2$ there exists a Q' with $Q \xrightarrow{\alpha} Q'$ and $(R', Q') \in \boldsymbol{S}_2$. Due to the definition of $\boldsymbol{S}_1 \circ \boldsymbol{S}_2$ it holds that $(P', Q') \in \boldsymbol{S}_1 \circ \boldsymbol{S}_2$ as required.

($\leftarrow$) similar to ($\rightarrow$).

Bisimulation - Summary

Bisimulation is an equivalence relation defined over a labelled transition system which respects non-determinism. The bisimulation technique can therefore be used to compare the observable behaviour of interacting systems.

Note: Strong bisimulation does not cover unobservable behaviour which is present in systems that have operators to define reaction (i.e., internal actions).

The π-Calculus

- The π-calculus is a model of concurrent computation based upon the notion of *naming*.
- The π-calculus is a calculus in which the topology of communication can evolve dynamically during evaluation.
- In the π-calculus communication links are identified by *names*, and computation is represented purely as the communication of names across links.
- The π-calculus is an extension of the process algebra CCS, following the work by Engberg and Nielsen who added mobility to CCS while preserving its algebraic properties.
- The most popular versions of the π-calculus are the monadic π-calculus, the polyadic π-calculus, and the simplified polyadic π-calculus.

The π-Calculus - Basic Ideas

• The most primitive in the π-calculus is a *name*, Names, infinitely many, are x, y,… $\in \boldsymbol{N}$; they have no structure.
• In the π-calculus we only have one other kind of entity: a *process*. We use P, Q, … to range over processes.

Polyadic prefixes:

• input prefix: $x(\tilde{y})$

"input some names $y_1,\ldots,y_n$ along the link named x"

• output prefix: $\bar{x}\langle\tilde{y}\rangle$

"output the names $y_1,\ldots,y_n$ along the link named x"

The π-Calculus - Syntax

Note: We only consider the simplified polyadic version.

P, Q	$::=$	$P \mid P$	Parallel composition
		$(\upsilon\, x)\, P$	Restriction
		$x(y_1,\ldots,y_n).P$	Input
		$\overline{x}\langle y_1,\ldots,y_n\rangle$	Output
		$!P$	Replication (input-only)
		$\mathbf{0}$	Null

Reduction Semantics

Milner proposed first a reduction semantics technique. Using the reduction semantics technique allows us to separate the laws which govern the neighbourhood relation among processes from the rules that specify their interaction.

$$!P \equiv P \mid !P$$
$$P \equiv P \mid \mathbf{0}$$
$$P \mid Q \equiv Q \mid P$$
$$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$$
$$(\upsilon\, x)P \mid Q \equiv (\upsilon\, x)(P \mid Q),\ x \notin \mathrm{fn}(Q)$$

$$\frac{Q \longrightarrow R}{Q \mid P \longrightarrow R \mid P} \qquad \frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q \equiv Q'}{P \longrightarrow Q} \qquad \frac{P \longrightarrow Q}{(\upsilon\, x)P \longrightarrow (\upsilon\, x)Q}$$

$$x(y_1,\ldots,y_n).P \mid \overline{x}\langle z_1,\ldots,z_n\rangle \longrightarrow P\{y_1,\ldots,y_n \backslash z_1,\ldots,z_n\}$$

Evolution

$\bar{x}\langle y\rangle \mid x(u).\bar{u}\langle v\rangle \mid \bar{x}\langle z\rangle$ *can evolve to* $\bar{y}\langle v\rangle \mid \bar{x}\langle z\rangle$ *or* $\bar{x}\langle y\rangle \mid \bar{z}\langle v\rangle$

$(\upsilon\, x)(\bar{x}\langle y\rangle \mid x(u).\bar{u}\langle v\rangle) \mid \bar{x}\langle z\rangle$ *can evolve to* $\bar{y}\langle v\rangle \mid \bar{x}\langle z\rangle$

$\bar{x}\langle y\rangle \mid !x(u).\bar{u}\langle v\rangle \mid \bar{x}\langle z\rangle$ *can evolve to*

$\bar{y}\langle v\rangle \mid !x(u).\bar{u}\langle v\rangle \mid \bar{x}\langle z\rangle$ *or* $\bar{x}\langle y\rangle \mid !x(u).\bar{u}\langle v\rangle \mid \bar{z}\langle v\rangle$

and

$\bar{y}\langle v\rangle \mid !x(u).\bar{u}\langle v\rangle \mid \bar{z}\langle v\rangle$

Church's Encoding of Booleans

$$\textbf{\textit{True}}(b) \equiv b(t, f).\bar{t}$$
$$\textbf{\textit{False}}(b) \equiv b(t, f).\overline{f}$$
$$\textbf{\textit{Not}}(b,c) \equiv b(t, f).\bar{c}(f,t)$$

$$(\upsilon\, c)(\textbf{\textit{Not}}(b,c) \mid \textbf{\textit{True}}(c)) = \textbf{\textit{False}}(b)$$
$$(\upsilon\, c)(b(t, f).\bar{c}(f,t) \mid c(t, f).\bar{t}) = b(t, f).\overline{f}$$

?

Actions:

$a(\tilde{b})$	Input action; x is the name at which it occurs, $\tilde{b}$ is the tuple of names which are received
$\bar{a}\langle\tilde{b}\rangle$	Output action; x is the name at which it occurs, $\tilde{b}$ is the tuple of names which are emitted
$(\upsilon\,\tilde{x})a\langle\tilde{b}\rangle$	Output action; x is the name at which it occurs, $\tilde{b}$ is the tuple of names which are emitted; $(\upsilon\,\tilde{x})$ denotes private names which are carried out from their current scope (*scope extrusion*)
τ	Silent action; this action denotes unobservable internal communication.

Labelled Transition Semantics

$$\text{IN}: a(\tilde{x}).P \xrightarrow{a(\tilde{b})} P\{\tilde{x} \setminus \tilde{b}\} \qquad \text{OUT}: \bar{a}\langle \tilde{b} \rangle \xrightarrow{\bar{a}\langle \tilde{b} \rangle} \mathbf{0}$$

$$\text{OPEN}: \frac{P \xrightarrow{(\upsilon\, \tilde{x})\bar{a}\langle \tilde{b} \rangle} P' \quad y \neq a \quad y \in \tilde{b} \text{ - } \tilde{x}}{(\upsilon\, y)P \xrightarrow{(\upsilon\, y, \tilde{x})\bar{a}\langle \tilde{b} \rangle} P'}$$

$$\text{COM}: \frac{P \xrightarrow{\bar{a}\langle \tilde{b} \rangle} P' \qquad Q \xrightarrow{a(\tilde{b})} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\text{CLOSE}: \frac{P \xrightarrow{(\upsilon\, \tilde{x})\bar{a}\langle \tilde{b} \rangle} P' \quad Q \xrightarrow{a(\tilde{b})} Q' \quad \tilde{x} \notin \text{fn}(Q)}{P \mid Q \xrightarrow{\tau} (\upsilon\, \tilde{x})(P' \mid Q')}$$

$$\text{PAR}: \frac{P \xrightarrow{\alpha} P' \qquad \text{bn}(\alpha) \cap \text{fn}(Q) = \varnothing}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

$$\text{RES}: \frac{P \xrightarrow{\alpha} P' \qquad x \notin \text{n}(\alpha)}{(\upsilon\, x)P \xrightarrow{\alpha} (\upsilon\, x)P'}$$

$$\text{REPL}: \frac{a(x).P \xrightarrow{a(\tilde{b})} P\{\tilde{x} \setminus \tilde{b}\}}{!a(x).P \xrightarrow{a(\tilde{b})} P\{\tilde{x} \setminus \tilde{b}\} \mid\ !a(x).P}$$

$$\text{fn}(a(\tilde{b})) = \{a\}$$
$$\text{fn}(a\langle \tilde{b} \rangle) = \{a, \tilde{b}\}$$
$$\text{fn}((\upsilon\, \tilde{x})a\langle \tilde{b} \rangle) = \{a, \tilde{b}\} \text{ - } \{\tilde{x}\}$$
$$\text{fn}(\tau) = \varnothing$$

$$\text{bn}(a(\tilde{b})) = \{\tilde{b}\}$$
$$\text{bn}(a\langle \tilde{b} \rangle) = \varnothing$$
$$\text{bn}((\upsilon\, \tilde{x})a\langle \tilde{b} \rangle) = \{\tilde{x}\}$$
$$\text{bn}(\tau) = \varnothing$$

Some Facts

• The side conditions in the transition rules ensure that names do not become accidentally be bound or captured.
In the rule RES the side condition prevents transitions like

$$(\upsilon\, x)a(b).P \xrightarrow{a(x)} (\upsilon\, x)P\{b\backslash x\}$$

which would violate the static binding assumed for restriction.

• In the given system bound names of an input are instantiated as soon as possible, namely in the rule for input - it is therefore an *early* transition system. Late instantiation is done in the rule for communication.
• The given system implements an asynchronous variant of the π-calculus. Therefore, output action are not directly observable.
• There is no rule for α-conversion. It is assumed that α-conversion is always possible.

Experiments

$$(\upsilon\, c)(\textbf{\textit{Not}}(b,c) \mid \textbf{\textit{True}}(c)) = \textbf{\textit{False}}(b)$$
$$(\upsilon\, c)(b(t,f).\bar{c}(f,t) \mid c(t,f).\bar{t}) = b(t,f).\bar{f}$$

?

Experiment 1:

$$(\upsilon\, c)(b(t,f).\bar{c}(f,t) \mid c(t,f).\bar{t})$$
$$\xrightarrow{\bar{b}\langle x,y\rangle} \quad (\upsilon\, c)(\bar{c}(y,x) \mid c(t',f').\bar{t'})$$
$$\xrightarrow{\tau} \quad (\upsilon\, c)(\bar{y})$$
$$\xrightarrow{y()} \quad \mathbf{0}$$

Experiment 2:

$$b(t,f).\bar{f}$$
$$\xrightarrow{\bar{b}\langle x,y\rangle} \quad \bar{y}$$
$$\xrightarrow{y()} \quad \mathbf{0}$$

Using strong bisimulation, the systems are not equivalent. Furthermore, an asynchronous observer can only indirectly see that an output message has been consumed.

Bisimulation - A Board Game

The central idea of bisimulation is that an external observer performs experiments with both processes *P* and *Q* observing the results in turn in order to match each others process behaviour step-by-step.

Checking the equivalence of processes this way one can think of this as a game played between two persons, the ***unbeliever***, who thinks that *P* and *Q* are not equivalent, and the ***believer***, who thinks that *P* and *Q* are equivalent. The underlying strategy of this game is that the unbeliever is trying to perform a process transition which cannot be matched by the believer.

Synchronous Interactions

There exists two kinds of experiments to check process equivalence: *input-experiments* and *output-experiments*. Both experiments are triggered by their corresponding opposite action.

In the synchronous case, input actions for a process *P* are only generated if there exists a matching receiver that is enabled within *P*. The existence of an input transition such that *P* evolves to *P'* reflects precisely the fact that a message offered by the observer has actually been consumed.

Asynchronous Interactions

In an synchronous system the sender of an output message does not know when the message is actually consumed. In other words, at the time of consumption of the message, its sender is not participating in the event anymore. Therefore, an asynchronous observer, in contrast to a synchronous one, cannot directly detect the input actions of the observed process. We need therefore a different notion of input-experiment.

Solution: Asynchronous input-experiments are incorporated into the definition of bisimulation such that inputs of processes have to be simulated only indirectly by observing the output behaviour of the process in context of arbitrary messages (e.g., $P \mid \overline{a}\langle \tilde{b} \rangle$).

The Silent Action

Strong bisimulation does not respect silent actions (τ-transitions).

Silent transitions denote unobservable internal communication. From the observer's point of view we can only notice that the system takes more time to respond.

Silent actions do not denote any interacting behaviour. Therefore, we may consider two systems P and Q to be equivalent if they only differ in the number of internal communications.

We write $P \stackrel{\alpha}{\Rightarrow} P'$ if $P\,(\stackrel{\tau}{\longrightarrow})^{*} \stackrel{\alpha}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^{*} P'$. In other words, a given observable action can have an arbitrary number of preceding or following internal communications.

Asynchronous Bisimulation

A binary relation ***S*** over processes *P* and *Q* is a weak (observable) bisimulation if it is symmetric and *P* ***S*** *Q* implies

- whenever $P \xrightarrow{\alpha} P'$, where α is either τ or output with $\mathrm{bn}(\alpha) \cap \mathrm{fn}(P|Q) = \varnothing$, then *Q'* exists such that $Q \stackrel{\alpha}{\Rightarrow} Q'$ and *P'* ***S*** *Q'*.

- $(P \mid \overline{a}\langle \tilde{b} \rangle)$ ***S*** $(Q \mid \overline{a}\langle \tilde{b} \rangle)$ for all messages $\overline{a}\langle \tilde{b} \rangle$.

Two processes *P* and *Q* are weakly bisimular, written $P \approx Q$, if there is a weak bisimulation ***S*** with *P* ***S*** *Q*.

Some Facts

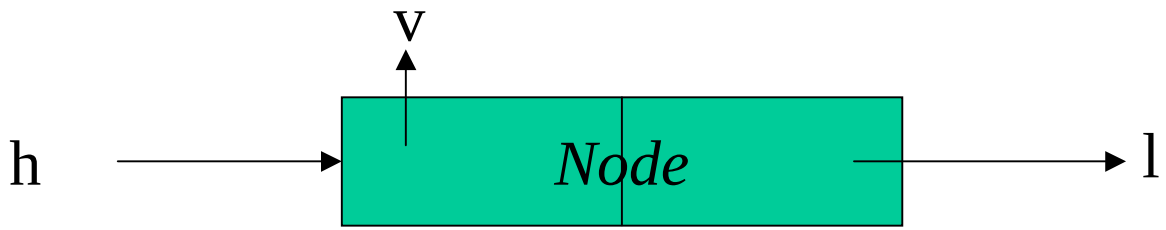
- $\approx$ is an equivalence relation.
- $\approx$ is a congruence relation.
- Leading τ-transitions are significant, i.e., they cannot be omitted.
- Asynchronous bisimulation is the framework that enables us to state $P = Q$ iff $P \approx Q$ and vice versa.

An Simple Object Model

$$
\begin{aligned}
\mathit{ReferenceCell} \equiv\ & (\upsilon\, v, s, g) \\
& (\ \overline{v}\langle 0\rangle \\
& \ |\, !s(n,r).v(\_).(\overline{v}\langle n\rangle \,|\, \overline{r}\langle\rangle) \\
& \ |\, !g(r).v(i).(\overline{v}\langle i\rangle \,|\, \overline{r}\langle i\rangle)\ )
\end{aligned}
$$

A List

A list is either *Nil* or *Cons* of value and a list.



The constant *Nil*, the construction *Cons*(*V*, *L*), and a list of *n* values are defined as follows:

$$
\begin{aligned}
Nil &= h(n,c).\overline{n} \\
Cons(V,L) &= (\upsilon\, v,l)(h(n,c).\overline{c}(v,l) \mid V\langle v\rangle \mid L\langle l\rangle) \\
[V_1,\ldots V_n] &= Cons(V_1, Cons(\ldots, Cons(V_n, Nil)\ldots))
\end{aligned}
$$

A Concurrent Language

V ::= X | Y | …	Variable
F ::= + | - | … | 0 | 1 | …	Function symbols
C ::= V = E	Assignment
C ; C	Sequential Composition
if E then C else C	Conditional Statement
while E do C	While Statement
let D in C end	Declaration
C par C	Parallel Composition
input V	Input
output E	Output
skip	
D ::= var V	Variable Declaration
E ::= V	Variable Expression
$F(E_1,\ldots, E_n)$	Function Call

Ambiguous Meaning

X = 0;
X = X + 1 par X = X +2

What is the value of X at the end of the second statement?

Basic Elements

• We assume that each element of the source language is assigned a process expression.

Variables: $X(init) \equiv (\upsilon\, v, setX, getX)$
$(\ \bar{v}\langle init\rangle$
$|\, !setX(n,r).v(\_).(\bar{v}\langle n\rangle \,|\, \bar{r}\langle\rangle)$
$|\, !getX(r).v(i).(\bar{v}\langle i\rangle \,|\, \bar{r}\langle i\rangle)\ )$

Skip: $\overline{done}\langle\rangle$

C_1 ; C_2 = $(\upsilon\, c)(C_1\{done \backslash c\} \,|\, c().C_2)$

C_1 par **C_2** = $(\upsilon\, l,r,t)(\bar{t}\langle \text{true}\rangle \,|\, C_1\{done \backslash l\} \,|\, C_2\{done \backslash r\} \,|$
$(l().t(b).(\ if\ \ b\ then\ r().Skip\ else\ \bar{l}\langle\rangle \,|\, \bar{t}\langle\ false\rangle) \,|$
$(r().t(b).(\ if\ \ b\ then\ l().Skip\ else\ \bar{r}\langle\rangle \,|\, \bar{t}\langle\ false\rangle))$

Expressions

$$X = (\upsilon\, ack)(\overline{getX}(ack) \mid ack(v).\overline{res}(v))$$

$$F(E_1,\ldots,E_n) = arg_1(x_1).....arg_n(x_n).\overline{F}(x_1,...,x_n,res)$$

$$M[F(E_1,\ldots,E_n)] = (\upsilon\, arg_1,\ldots,arg_n)(M[E_1]\{res\backslash arg_1\} \mid \ldots \mid M[E_n]\{res\backslash arg_n\} \mid M[F])$$

Operation Sequence

X = 0;
X = X + 1 par X = X +2

What is the value of X at the end of the second statement?

According to the former definitions the value of X is either 1, 2, or 3. The three values are possible since every atomic action can occur in an arbitrary and meshed order.

To guarantee a specific result (e.g., 1 or 2), we need to employ semaphors.

What have we learned?

- Classical automata theory does not cope correctly with interacting behaviour
- Bisimulation is an equivalence relation defined over a labelled transition system which respects non-determinism andcan therefore be used to compare the observable behaviour of interacting systems.
- The π-calculus is a name-passing system in which program progress is expressed by communication.
- Which the π-calculus we can model higher-level programming abstractions like objects and lists.
- A concurrent programming language can be assigned a semantics based on the π-calculus.

Software Composition Group

University of Berne

Research:

- Piccola - a small composition language
- The πL-calculus - a formal foundation for software composition.
- COORDINA - coordination models and languages

Resources: **http://www.iam.unibe.ch/~scg**

Evolving Automata