Process-oriented architectural design

Luís S. Barbosa

HASLab - INESC TEC
Universidade do Minho
Braga, Portugal

December, 2011

Process-oriented architectural design

The 'rationale'

- components in an architecture are computational entities accessible through published interfaces which interact in a 'continuous' way according to specific protocols
- both interfaces and protocols are specifications of intended behaviour
- labelled transition systems provide an operational model for behaviour
- inside which one may reason equationally (bisimilarity) and inequationally (simulation)
- both components and the glue code that keeps them togethers can be viewed and described uniformly

Process-oriented architectural design

What is missing to bring this 'rationale' into practice?

- a language to describe LTS
- combinators to compose LTS
- a calculus to reason about and transform behaviours, sound and complete wrt suitable LTS equivalences (e.g., bisimilarity)
- a specific logic to specify LTS properties and corresponding proof techniques
- an ADL in which interfaces and iteration protocols are described in a process algebra, which furthermore provides a precise semantics to the architectural description and possibly some tool support

Process-oriented architectural design

What is missing to bring this 'rationale' into practice?

- a language to describe LTS
- combinators to compose LTS
- a calculus to reason about and transform behaviours, sound and complete wrt suitable LTS equivalences (e.g., bisimilarity)
- a specific logic to specify LTS properties and corresponding proof techniques
- an ADL in which interfaces and iteration protocols are described in a process algebra, which furthermore provides a precise semantics to the architectural description and possibly some tool support

Process-oriented architectural design

What is missing to bring this 'rationale' into practice?

processes a language to describe LTS

composition combinators to compose LTS

process algebra a calculus to reason about and transform behaviours, sound and complete wrt suitable LTS equivalences (e.g., bisimilarity)

modal logic a specific logic to specify LTS properties and corresponding proof techniques

spec language an ADL in which interfaces and iteration protocols are described in a process algebra, which furthermore provides a precise semantics to the architectural description and possibly some tool support

Process-oriented architectural design

- process algebras provided the first formal semantics to (at least some components of) several ADLs
- examples: WRIGHT, DARWIN, ACME, AADL (behaviour annex), PADL, ...
- resort to popular process algebras: CSP, CCS, ACP, π-calculus, ...

Reference

Alessandro Aldini, Marco Bernardo, Flavio Corradini.
A Process Algebraic Approach to Software Architecture Design
Springer-Verlag, 2010.

Roadmap

Process algebra (via mCRL2)

- Sequential processes
- Deadlock & termination (LTS revisited)
- Interaction
- Abstraction from internal activity (LTS revisited)
- Processes with data

A process-oriented ADL

An introduction to ARCHERY (by Alejandro Sanchez)

`www.unsl.edu.ar/~asanchez/index.php?page=archery`

Actions & processes

Interaction through multisets of actions

- A multiaction is an elementary unit of interaction that can execute itself atomically in time (no duration), after which it terminates successfully

$$\alpha \quad ::= \quad \tau \mid a(d) \mid \alpha \,|\, \alpha$$

- actions may be parametric on data
- the structure $\langle \mathcal{N}, |, \tau \rangle$ forms an Abelian monoid
- τ is the empty action, which contains no actions and as such cannot be observed

Actions & processes

Process

is a description of how the interaction capacities of a system evolve, i.e., its behaviour

for example,

$$E \triangleq a.b + a.E$$

- analogy: regular expressions vs finite automata

The framework

Process

... abstract representation of a system's behaviour

Algebra

... a mathematical structure satisfying a particular set of axioms

Process Algebra

... a framework for the specification and manipulation of process terms as induced by a collection of operator symbols, encompassing an operational and an axiomatic theory (sound and complete wrt bisimilarity)

Sequential processes

Sequential, non deterministic behaviour

The set $\mathbb{P}$ of processes is the set of all terms generated by the following BNF, for $a \in \mathcal{N}$,

$$p \ ::= \ \alpha \mid \delta \mid p + p \mid p \cdot p \mid \mathsf{P}(d)$$

- atomic process: a for all $a \in \mathcal{N}$
- choice: $+$
- sequential composition: $\cdot$
- inaction or deadlock: δ
- process references introduced through definitions of the form $\mathsf{P}(x : D) = p$, parametric on data

Sequential Processes

Exercise

Describe the behaviour of

- $a.b.\delta.c + a$
- $(a + b).\delta.c$
- $(a + b).e + \delta.c$
- $a + (\delta + a)$
- $a.(b + c).d.(b + c)$

Sequential processes

Sequential, non deterministic behaviour

The set $\mathbb{P}$ of processes is the set of all terms generated by the following BNF, for $a \in \mathcal{N}$,

$$p \;::=\; \alpha \mid \delta \mid p + p \mid p \cdot p \mid \mathsf{P}(d)$$

- atomic process: a for all $a \in \mathcal{N}$
- choice: $+$
- sequential composition: $\cdot$
- inaction or deadlock: δ
- process references introduced through definitions of the form $\mathsf{P}(x : D) = p$, parametric on data

Axioms: : $+$, $\cdot$, δ

$$
\begin{array}{lll}
A1 & x + y & = y + x \\
A2 & (x + y) + z & = x + (y + z) \\
A3 & x + x & = x \\
A4 & (x + y).z & = x.z + y.z \\
A5 & (x.y).z & = x.(y.z) \\
A6 & x + \delta & = x \\
A7 & \delta \cdot x & = 0
\end{array}
$$

- the equality relation is sound: if $s = t$ holds for basic process terms, then $s \sim t$
- and complete: if $s \sim t$ holds for basic process terms, then $s = t$
- an axiomatic theory enables equational reasoning

Axioms: : $+$, $\cdot$, δ

$$
\begin{array}{lll}
A1 & x + y & = y + x \\
A2 & (x + y) + z & = x + (y + z) \\
A3 & x + x & = x \\
A4 & (x + y).z & = x.z + y.z \\
A5 & (x.y).z & = x.(y.z) \\
A6 & x + \delta & = x \\
A7 & \delta \cdot x & = 0
\end{array}
$$

- the equality relation is sound: if $s = t$ holds for basic process terms, then $s \sim t$
- and complete: if $s \sim t$ holds for basic process terms, then $s = t$
- an axiomatic theory enables equational reasoning

Axioms: : $+$, $\cdot$, δ

Exercise

- show that $\delta.(a + b) = \delta \cdot a + \delta \cdot b$
- show that $a + (\delta + a) = a$
- is it true that $a.(b + c) = a.b + a.c$?

mCRL2: A toolset for process algebra

mCRL2 provides:

- a generic process algebra, based on ACP (Bergstra & Klop, 82), in which other calculi can be embedded
- extended with data and (real) time
- the full μ-calculus as a specification logic
- powerful toolset for simulation and verification of reactive systems

www.mcrl2.org

mCRL2: A toolset for process algebra

Example

```
act    order, receive, keep, refund, return;

proc   Buy = order.OrderedItem

       OrderedItem = receive.ReceivedItem + refund.Buy;
       ReceivedItem = return.OrderedItem + keep;

init   Buy;
```

Deadlock & Termination

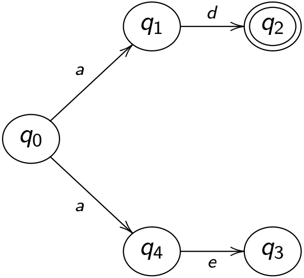
Deadlock state

a reachable state that does not terminate and has no outgoing transitions.

Termination

add a predicate $\downarrow s$ to the definition of a LTS

Termination vs deadlock



Trace equivalence

Trace (from language theory)

A word $\sigma \in \mathcal{N}^*$ is a trace of a state $s \in S$ iff there is another state $t \in S$ such that $s \xrightarrow{\sigma}^* t$

Trace (using ✓ to witness final states)

Tr(s), the set of traces of state s, is the minimal set including

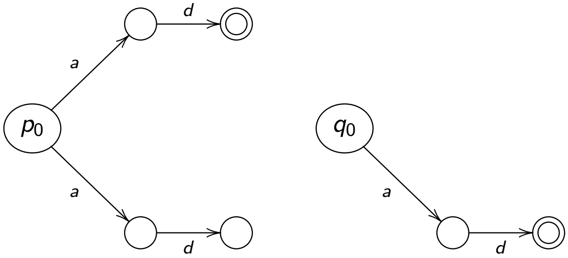
$$\begin{aligned} \epsilon &\in \mathrm{Tr}(s) \\ \checkmark &\in \mathrm{Tr}(s) \quad \text{if} \quad \downarrow s \\ a\sigma &\in \mathrm{Tr}(s) \quad \text{if} \quad \exists_t \cdot s \xrightarrow{a} t \wedge \sigma \in \mathrm{Tr}(t) \end{aligned}$$

Trace equivalence

Two states are trace equivalent if $\mathrm{Tr}(s) = \mathrm{Tr}(s')$

Trace equivalence

In any case, fails to preserve deadlock



although preserving sequencing

e.g. before every c an a action b must be done

Language equivalence

Language (from language theory)

A word $\sigma \in \mathcal{N}^*$ is a run (or a complete trace) of a state $s \in S$ iff there is another state $t \in S$, such that $s \xrightarrow{\sigma}^* t$ and $\downarrow t$. The language recognized by a state $s \in S$ is the set of runs of s

Language (using ✓ to witness final states)

Lang(s), the language recognized by a state s, is the minimal set including

$$\begin{aligned} \epsilon &\in \mathrm{Lang}(s) \quad \text{if } s \text{ is a deadlock state} \\ \checkmark &\in \mathrm{Lang}(s) \quad \text{if } \downarrow s \\ a\sigma &\in \mathrm{Lang}(s) \quad \text{if } \exists_t \cdot s \xrightarrow{a} t \wedge \sigma \in \mathrm{Lang}(t) \end{aligned}$$

Language equivalence

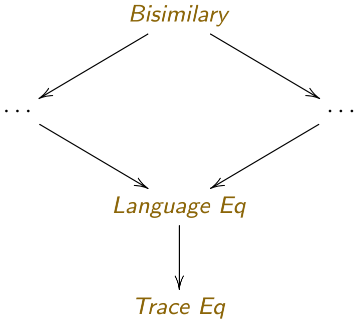
Two states are language equivalent if $\text{Lang}(s) = \text{Lang}(s')$, i.e., if both recognize the same language.

... need more general models and theories:

- Several interaction points
- Need to distinguish normal from anomolous termination
- Non determinisim should be taken seriously: the notion of equivalence based on accepted language is blind wrt non determinism
- Moreover: the reactive character of systems entail that not only the generated language is important, but also the states traversed during an execution of the automata.

Notes

The Van Glabbeek linear - branching time spectrum



... collapses for deterministic transition systems: why?

Parallel composition

$\|$ = interleaving + synchronization

- modelling principle: interaction is the key element in software design
- modelling principle: (distributed, reactive) architectures are configurations of communicating black boxes

$$p ::= \cdots \mid p \parallel p \mid p \mid p \mid p \mathbin{\|\!\!\lfloor} p$$

Parallel composition

- parallel $p \parallel q$: interleaves and synchronises the actions of both processes.
- synchronisation $p \mid q$: synchronises the first actions of p and q and combines the remainder of p with q with $\parallel$, cf axiom:

$$(a.p) \mid (b.q) \sim (a \mid b) \,.\, (p \parallel q)$$

- Processes of the form $a \mid a \mid \cdots \mid a$ are called multiactions

Parallel composition

A semantic parentesis

Lemma: There is no sound and complete finite axiomatisation for this process algebra with $\parallel$ modulo bisimilarity [F. Moller, 1990].

Solution: combine two auxiliar operators:

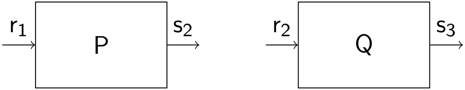
- left merge: $⫴$ (executes a first action of p)
- synchronous product: $|$

such that

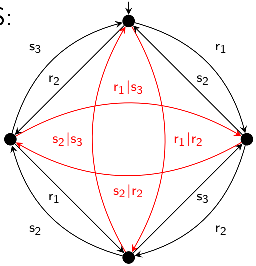
$$\boxed{p \parallel t \;\sim\; (p ⫴ t + t ⫴ p) + p \mid t}$$

Parallel composition

Example $P \parallel Q$



Corresponding LTS:



Interaction

Communication $\Gamma_C(p)$ (com)

- applies a communication function C forcing action synchronization and renaming to a new action:

$$a_1 \mid \cdots \mid a_n \;\rightarrow\; c$$

- data parameters are retained in action c, e.g.

$$\Gamma_{\{a|b \rightarrow c\}}(a(8) \mid b(8)) \;=\; c(8)$$
$$\Gamma_{\{a|b \rightarrow c\}}(a(12) \mid b(8)) \;=\; a(12) \mid b(8)$$
$$\Gamma_{\{a|b \rightarrow c\}}(a(8) \mid a(12) \mid b(8)) \;=\; a(12) \mid c(8)$$

- left hand-sides in C must be disjoint: e.g., $\{a \mid b \rightarrow c, a \mid d \rightarrow j\}$ is not allowed

Interface control

Restriction: $\nabla_B(p)$ (`allow`)

- specifies which multiactions from a non-empty multiset of action names are allowed to occur
- disregards the data parameters of the multiactions

$$\nabla_{\{d,a|b\}}(d(12) + a(8) + (b(false,4) \mid c)) \;=\; d(12) + (b(false,4) \mid c)$$

- τ is always allowed to occur

Interface control

Block: $\partial_B(p)$ (`block`)

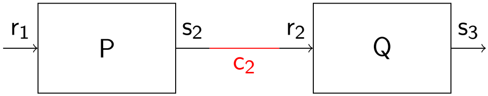
- specifies which multiactions from a set of action names are not allowed to occur
- disregards the data parameters of the multiactions

$$\partial_{\{b\}}(d(12) + a(8) + (b(\mathit{false}, 4) \mid c)) \;=\; d(12) + a(8)$$

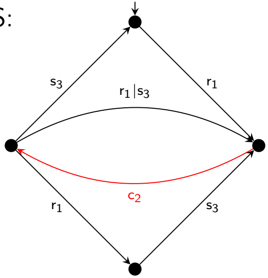
- the effect is that of renaming to δ
- τ cannot be blocked

Interaction

Example $\partial_{r_2,s_2}((\Gamma_{\{s_2|r_2\to c_2\}}(P \parallel Q))$



Corresponding LTS:



Interaction

Enforce communication

- $\nabla_{\{c\}}(\Gamma_{\{a|b \to c\}}(p))$
- $\partial_{\{a,b\}}(\Gamma_{\{a|b \to c\}}(p))$

Interface control

Renaming $\rho_M(p)$ (rename)

- renames actions in p according to a mapping M
- also disregards the data parameters, but when a renaming is applied the data parameters are retained:

$$\partial_{\{d \to h\}}(d(12) + s(8) \mid d(\mathit{false}) + d.a.d(7)) \\ = h(12) + s(8) \mid h(\mathit{false}) + h.a.h(7)$$

- τ cannot be renamed

Interface control

Hiding $\tau_H(p)$ (hide)

- hides (or renames to τ) all actions with an action name in H in all multiactions of p. renames actions in p according to a mapping M
- disregards the data parameters

$$\tau_{\{d\}}(d(12) + s(8) \mid d(false) + h.a.d(7))$$
$$= \tau + s(8) \mid \tau + h.a.\tau = \tau + s(8) + h.a.\tau$$

- τ and δ cannot be renamed
- what is the LTS of $\tau_{\{t_2\}}(\partial_{r_2,s_2}(\Gamma_{\{s_2|r_2 \to c_2\}}(P \parallel Q)))$?

Example

New buffers from old

```
act    inn,outt,ia,ib,oa,ob,c :  Bool;

proc   BufferS = sum n:  Bool.inn(n).outt(n).BufferS;

       BufferA = rename({inn -> ia, outt -> oa}, BufferS);
       BufferB = rename({inn -> ib, outt -> ob}, BufferS);

       S = allow({ia,ob}, comm({oa|ib -> c}, BufferA || BufferB));

init   hide({c}, S);
```

Abstraction

Main idea:
Take a set of actions as internal or non-observable

Approaches

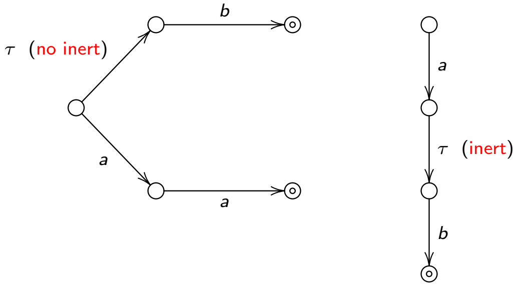
- R. Milner's weak bisimulation [Mil80]
- Van Glabbeek and Weijland's branching bisimulation [GW96]

Internal actions

τ abstracts internal activity

inert τ: internal activity is undetectable by observation

non inert τ: internal activity is indirectly visible



Internal actions

Adding τ to the set of actions has a number of consequences

- only external actions are observable
- the effects of an internal action can only be observed if it determines a choice
- entails the need of a weaker notion of bisimulation to relate e.g.

$$p.q \text{ and } p.(\tau + \tau.\tau).q$$

Branching bisimulation

- Intuition similar to that of strong bisimulation: But now, instead of letting a single action be simulated by a single action, an action can be simulated by a sequence of internal transitions, followed by that single action.
- An internal action τ can be simulated by any number of internal transitions (even by none).
- If a state can terminate, it does not need to be related to a terminating state: it suffices that a terminating state can be reached after a number of internal transitions.

Branching bisimulation

Definition

Given $\langle S_1, \mathcal{N}, \downarrow_\infty, \longrightarrow_\infty \rangle$ and $\langle S_2, \mathcal{N}, \downarrow_\in, \longrightarrow_\in \rangle$ over $\mathcal{N}$, relation $R \subseteq S_1 \times S_2$ is a branching bisimulation iff for all $\langle p, q \rangle \in R$ and $a \in \mathcal{N}$,

1. If $p \xrightarrow{a}_1 p'$, then
   - either $a = \tau$ and $p' R q$
   - or, there is a sequence $q \xrightarrow{\tau}_2 \cdots \xrightarrow{\tau}_2 q'$ of (zero or more) τ-transitions such that pRq' and $q' \xrightarrow{a}_2 q''$ with $p' R q''$.
2. If $p \downarrow_1$, then there is a sequence $q \xrightarrow{\tau}_2 \cdots \xrightarrow{\tau}_2 q'$ of (zero or more) τ-transitions such that pRq' and $q' \downarrow_2$.

1'., 2'. symmetrically ...

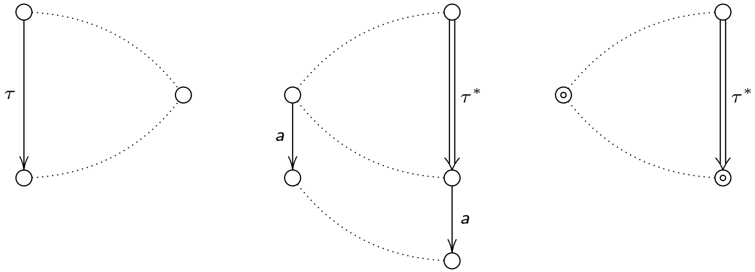
Branching bisimilarity

Definition

$$p \approx_b q \;\Leftrightarrow\; \langle \exists\, R \;::\; R \text{ is a branching bisimulation and } \langle p, q \rangle \in R \rangle$$

Branching bisimulation

... preserves the branching structure



Branching bisimilarity

... does not preserve τ-loops

satisfying a notion of fairness: if a τ-loop exists, then no infinite execution sequence will remain in it forever if there is a possibility to leave

Branching bisimilarity

Problem

If an alternative is added to the initial state then transition systems that were branching bisimilar may cease to be so.

Example: add a *b*-labelled branch to the initial states of

Rooted branching bisimilarity

Startegy

Impose a rootedness condition [R. Milner, 80]:

Initial τ-transitions can never be inert, i.e., two states are equivalent if they can simulate each other?s initial transitions, such that the resulting states are branching bisimilar.

Rooted branching bisimulation

Definition

Given $\langle S_1, \mathcal{N}, \downarrow_\infty, \longrightarrow_\infty \rangle$ and $\langle S_2, \mathcal{N}, \downarrow_\in, \longrightarrow_\in \rangle$ over $\mathcal{N}$, relation $R \subseteq S_1 \times S_2$ is a rooted branching bisimulation iff

1. it is a branching bisimulation
2. for all $\langle p, q \rangle \in R$ and $a \in \mathcal{N}$,
    - If $p \xrightarrow{a}_1 p'$, then there is a $q' \in S_2$ such that $q \xrightarrow{a}_2 q'$ and $p' \approx q'$
    - If $q \xrightarrow{a}_2 q'$, then there is a $p' \in S_1$ such that $p \xrightarrow{a}_1 p'$ and $p' \approx q'$

Rooted branching bisimilarity

Definition

$p \approx_{rb} q \;\Leftrightarrow\; \langle \exists\, R \;::\; R$ is a rooted branching bisimulation and $\langle p, q \rangle \in R \rangle$

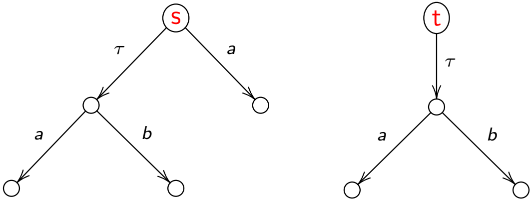
Lemma

$$\sim \;\subseteq\; \approx_{rb} \;\subseteq\; \approx_b$$

Of course, in the absence of τ actions, $\sim$ and $\approx_b$ coincide.

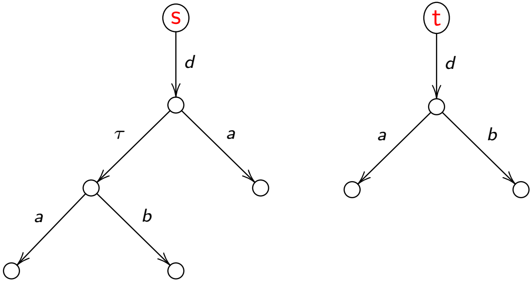
Example

branching but not rooted



Example

rooted branching bisimilar



Weak bisimulation

Definition [Milner,80]

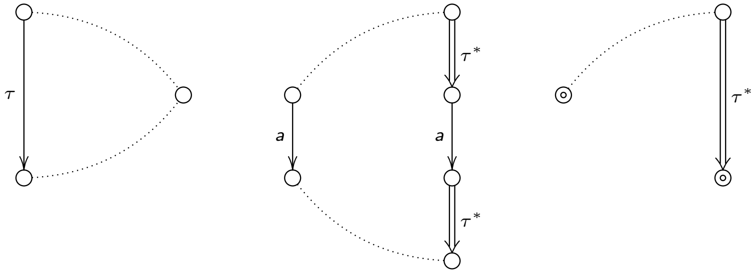
Given $\langle S_1, \mathcal{N}, \downarrow_\infty, \longrightarrow_\infty \rangle$ and $\langle S_2, \mathcal{N}, \downarrow_\in, \longrightarrow_\in \rangle$ over $\mathcal{N}$, relation $R \subseteq S_1 \times S_2$ is a weak bisimulation iff for all $\langle p, q \rangle \in R$ and $a \in \mathcal{N}$,

1. If $p \xrightarrow{a}_1 p'$, then
   - either $a = \tau$ and $p' R q$
   - or, there is a sequence $q \xrightarrow{\tau}_2 \cdots \xrightarrow{\tau}_2 t \xrightarrow{a}_2 t' \xrightarrow{\tau}_2 \cdots \xrightarrow{\tau}_2 q'$ involving zero or more τ-transitions, such that $p' R q'$.
2. If $p \downarrow_1$, then there is a sequence $q \xrightarrow{\tau}_2 \cdots \xrightarrow{\tau}_2 q'$ of (zero or more) τ-transitions such that $q' \downarrow_2$.

1'., 2'. symmetrically ...

Weak bisimulation

... does not preserve the branching structure



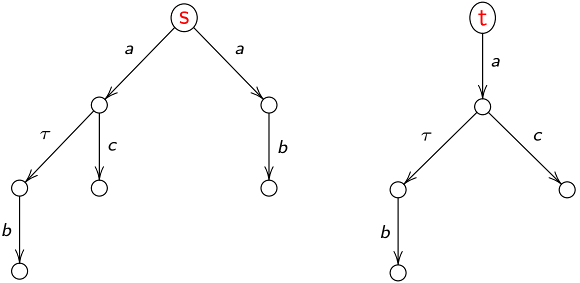
Weak bisimilarity

Definition

$$p \approx_w q \;\Leftrightarrow\; \langle \exists\, R \;::\; R \text{ is a branching bisimulation and } \langle p, q \rangle \in R \rangle$$

Example

weak but not branching



Rooted weak bisimulation

Definition

Given $\langle S_1, \mathcal{N}, \downarrow_\infty, \longrightarrow_\infty \rangle$ and $\langle S_2, \mathcal{N}, \downarrow_\in, \longrightarrow_\in \rangle$ over $\mathcal{N}$, relation $R \subseteq S_1 \times S_2$ is a rooted weak bisimulation iff for all $\langle p, q \rangle \in R$ and $a \in \mathcal{N}$,

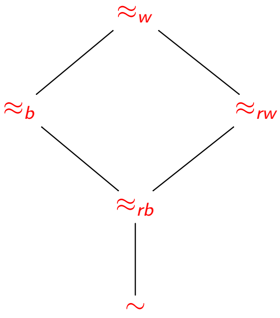
- If $p \xrightarrow{\tau}_1 p'$, then there is a non empty sequence of τ such that $q \xrightarrow{\tau}_2 \xrightarrow{\tau}_2 \ldots \xrightarrow{\tau}_2 \xrightarrow{\tau}_2 q'$ and $p' \approx_w q'$
- Symmetrically ...

Rooted weak bisimilarity

Definition

$$p \approx_{rw} q \;\Leftrightarrow\; \langle \exists\, R \;::\; R \text{ is a rooted weak bisimulation and } \langle p, q \rangle \in R \rangle$$

Lemma



(ordered by $\subseteq$)

Axioms: : $+$, $\cdot$, δ, τ

$A1$	$x + y$	$= y + x$
$A2$	$(x + y) + z$	$= x + (y + z)$
$A3$	$x + x$	$= x$
$A4$	$(x + y).z$	$= x.z + y.z$
$A5$	$(x.y).z$	$= x.(y.z)$
$A6$	$x + \delta$	$= x$
$A7$	$\delta \cdot x$	$= 0$
$A8$	$x.\tau$	$= x$
$A9$	$x.(\tau.(y + z) + y)$	$= x.(y + z)$

- extra axioms are valid wrt branching bisimilarity

Data types

- Equalities: equality, inequality, conditional (`if(-,-,-)`)
- Basic types: booleans, naturals, reals, integers, ... with the usual operators
- Sets, multisets, sequences ... with the usual operators
- Function definition, including the λ-notation
- Inductive types: as in

```
sort   BTree = struct leaf(Pos) | node(BTree, BTree)
```

Signatures and definitions

Sorts, functions, constants, variables ...

```
sort   S, A;

cons   s,t:S, b:set(A);

map    f:  S x S -> A;
       c:  A;

var    x:S;

eqn    f(x,s) = s;
```

Signatures and definitions

A full functional language ...

```
sort   BTree = struct leaf(Pos) | node(BTree, BTree);

map    flatten:  BTree -> List(Pos);

var    n:Pos, t,r:BTree;

eqn    flatten(leaf(n)) = [n];
       flatten(node(t,r)) = t++r;
```

Processes with data

Why?

- Precise modeling of real-life systems
- Data allows for finite specifications of infinite systems

How?

- data and processes parametrized
- summation over data types: $\sum_{n:N} s(n)$
- processes conditional on data: $b \rightarrow p \diamond q$

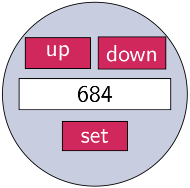
Examples

A counter

```
act    up, down;
       setcounter:Pos;

proc   Ctr(x:Pos) = up.Ctr(x+1)
             + (x>0) -> down.Ctr(x-1)
             + sum m:Pos.(setcounter(m).Ctr(m))

init   Ctr(345);
```



Examples

A prime checker

$$
\begin{array}{ll}
\texttt{map} & \texttt{primes : Set(N);} \\
\texttt{eqn} & \texttt{primes} = \{n : \texttt{N} \mid \forall_{p,q \in \texttt{N}}\ p, q > 1 \Rightarrow (p * q) \neq n\}\texttt{;} \\
\texttt{act} & \texttt{yes, no;} \\
 & \texttt{ask:N;} \\
 & \\
\texttt{proc} & \mathit{Checker} = \Sigma_n \texttt{ask}(n).(n \in \texttt{primes} \rightarrow \texttt{yes} \diamond \texttt{no}).\mathit{Checker} \\
 & \\
\texttt{init} & \mathit{Checker}
\end{array}
$$

Examples

A dynamic binary tree

```
act   left,right;
map   N:Pos;
eqn   N = 512;
proc  X(n:Pos)=(n<=N)->(left.X(2*n)+right.X(2*n+1))<>delta;
init  X(1);
```