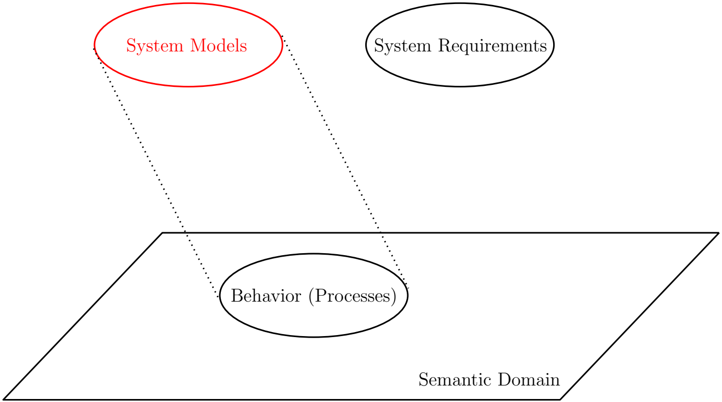
System Validation: Describing Sequential Processes

Mohammad Mousavi and Jeroen Keiren

HALMSTAD UNIVERSITY

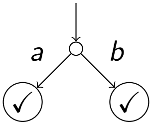
Open Universiteit

General Overview



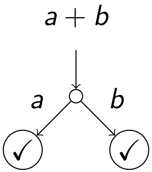
Alternative composition

$$a + b$$



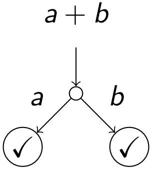
Alternative composition

- Syntax $p + q$
- Intuition the process behaves as either p or q



Alternative composition

- Syntax $p + q$
- Intuition the process behaves as either p or q



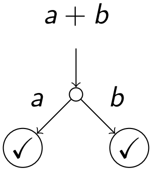
Axioms

A1	$x + y = y + x$
A2	$x + (y + z) = (x + y) + z$
A3	$x + x = x$

Write $x \subseteq y$ for $x + y = y$.

Alternative composition

- Syntax $p + q$
- Intuition the process behaves as either p or q

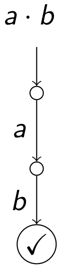


Axioms

A1	$x + y = y + x$
A2	$x + (y + z) = (x + y) + z$
A3	$x + x = x$

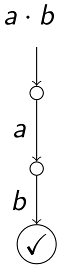
Write $x \subseteq y$ for $x + y = y$.

Sequential composition



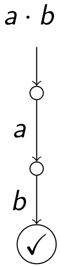
Sequential composition

- Syntax $p \cdot q$
- Intuition the process behaves as p and upon termination of p, as q.



Sequential composition

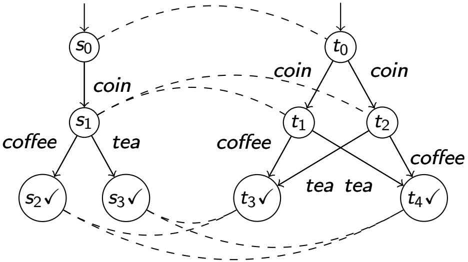
- Syntax $p \cdot q$
- Intuition the process behaves as p and upon termination of p, as q.



Axioms

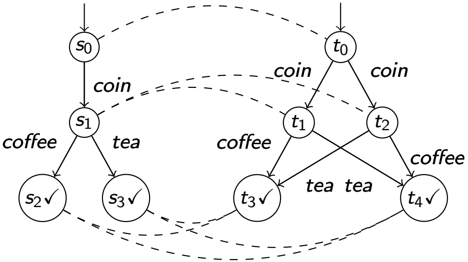
A4	$(x + y)\cdot z = x\cdot z + y\cdot z$
A5	$(x\cdot y)\cdot z = x\cdot(y\cdot z)$

Example



s_0
t_0
coin
coin
coin
s_1
t_1
t_2
coffee
tea
coffee
coffee
tea tea
$s_2 \checkmark$
$s_3 \checkmark$
$t_3 \checkmark$
$t_4 \checkmark$

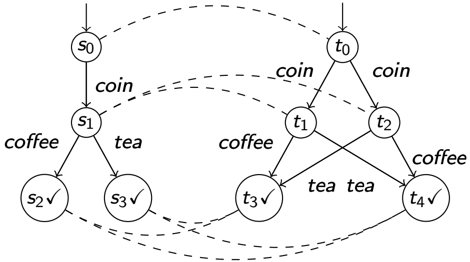
Example



$$coin \cdot (coffee + tea)$$

$$(coin \cdot (coffee + tea)) + (coin \cdot (tea + coffee))$$

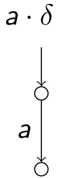
Example



$$
\begin{aligned}
& coin \cdot (coffee + tea) \\
\overset{A1,A3}{=} \; & (coin \cdot (coffee + tea)) + (coin \cdot (tea + coffee))
\end{aligned}
$$

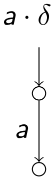
Previous Page Next Page

Deadlock



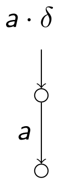
Deadlock

- Syntax δ
- Intuition a process that cannot do anything but let time pass



Deadlock

- Syntax δ
- Intuition a process that cannot do anything but let time pass

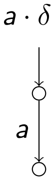


Axioms

A6	$x + \delta = x$
A7	$\delta \cdot x = \delta$

Deadlock

- Syntax δ
- Intuition a process that cannot do anything but let time pass



Axioms

A6	$x + \delta = x$
A7	$\delta \cdot x = \delta$

Conditional operator

$$(n < 42) \to a \diamond b$$

Conditional operator

$$(n < 42) \to a \diamond b$$

- Syntax $c \to p \diamond q$, where c is of type `Bool`
- Intuition if c holds, behave as p, otherwise as q

Conditional operator

$$(n < 42) \to a \diamond b$$

- Syntax $c \to p \diamond q$, where c is of type `Bool`
- Intuition if c holds, behave as p, otherwise as q

Axioms

Cond1	$true \to x \diamond y = x$
Cond2	$false \to x \diamond y = y$
THEN	$c \to x = c \to x \diamond \delta$

Conditional operator

$$(n < 42) \rightarrow a \diamond b$$

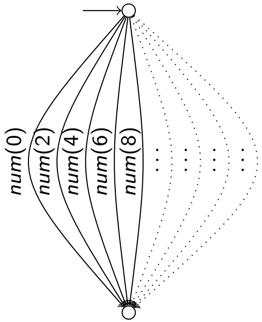
- Syntax $c \rightarrow p \diamond q$, where c is of type `Bool`
- Intuition if c holds, behave as p, otherwise as q

Axioms

Cond1	$true \rightarrow x \diamond y = x$
Cond2	$false \rightarrow x \diamond y = y$
THEN	$c \rightarrow x = c \rightarrow x \diamond \delta$

Sum operator

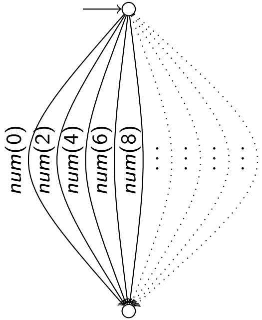
$\sum_{v:Nat} num(2*v);$



Sum operator

- Syntax $\sum_{d:D} X(d)$
- Intuition generalize alternative composition: may behave as $X(d)$, for each value d of type D

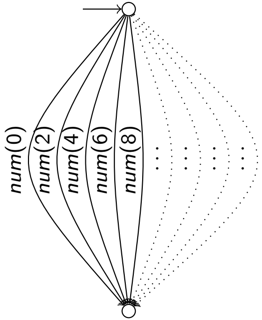
$\sum_{v:Nat} num(2 * v);$



Sum operator

- Syntax $\sum_{d:D} X(d)$
- Intuition generalize alternative composition: may behave as $X(d)$, for each value d of type D

$\sum_{v:Nat} num(2*v);$



Axioms

SUM1	$\sum_{d:D} x = x$
SUM3	$\sum_{d:D} X(d) = X(e) + \sum_{d:D} X(d)$
SUM4	$\sum_{d:D}(X(d) + Y(d)) = \sum_{d:D} X(d) + \sum_{d:D} Y(d)$
SUM5	$(\sum_{d:D} X(d))\cdot y = \sum_{d:D} X(d)\cdot y$

Example

One time usable buffer, with messages of type Message

$$\sum_{m:Message} read(m) \cdot forward(m)$$

Example

One time usable buffer, with messages of type Message

$$\sum_{m:Message} read(m) \cdot forward(m)$$

Problem How to handle repetition?

Recursion

Define set of equations with variables as left hand side:

$$P = x$$

where x a process, that can refer to variables such as P

Recursion

Define set of equations with variables as left hand side:

$$P = x$$

where x a process, that can refer to variables such as P

- allows definition of infinite processes
- can store data in parameters

Example

Reusable 1-place FIFO buffer, with messages of type Message

$$Buffer = \sum_{m:Message} read(m) \cdot forward(m) \cdot Buffer$$

Example

Reusable 1-place FIFO buffer, with messages of type Message

$$Buffer = \sum_{m:Message} read(m) \cdot forward(m) \cdot Buffer$$

or, in mCRL2:

```
sort Message;

act read,forward: Message;

proc Buffer = sum m: Message . read(m) . forward(m) . Buffer;

init Buffer;
```

Example

Infinite queue

$$Queue(l{:}List(Message)) = \sum_{m:Message} read(m) \cdot Queue(l \triangleleft m)$$
$$+ (l \neq [] \rightarrow forward(head(l)) \cdot Queue(tail(l)))$$

Example

Infinite queue

$$
\begin{aligned}
Queue(l{:}List(Message)) = & \sum_{m:Message} read(m) \cdot Queue(l \lhd m) \\
& + (l \neq [] \rightarrow forward(head(l)) \cdot Queue(tail(l)))
\end{aligned}
$$

Previous Page | Next Page

Example

Infinite queue

$$Queue(l{:}List(Message)) = \sum_{m:Message} read(m) \cdot Queue(l \lhd m)$$
$$+ (l \neq [] \rightarrow forward(head(l)) \cdot Queue(tail(l)))$$

or, in mCRL2:

```
sort Message;

act read,forward: Message;

proc Queue(l: List(Message)) =
     sum m: Message . read(m) . Queue(l <| m)
   + (l != []) -> forward(head(l)) . Queue(tail(l));

init Queue([]);
```

Thank you very much.