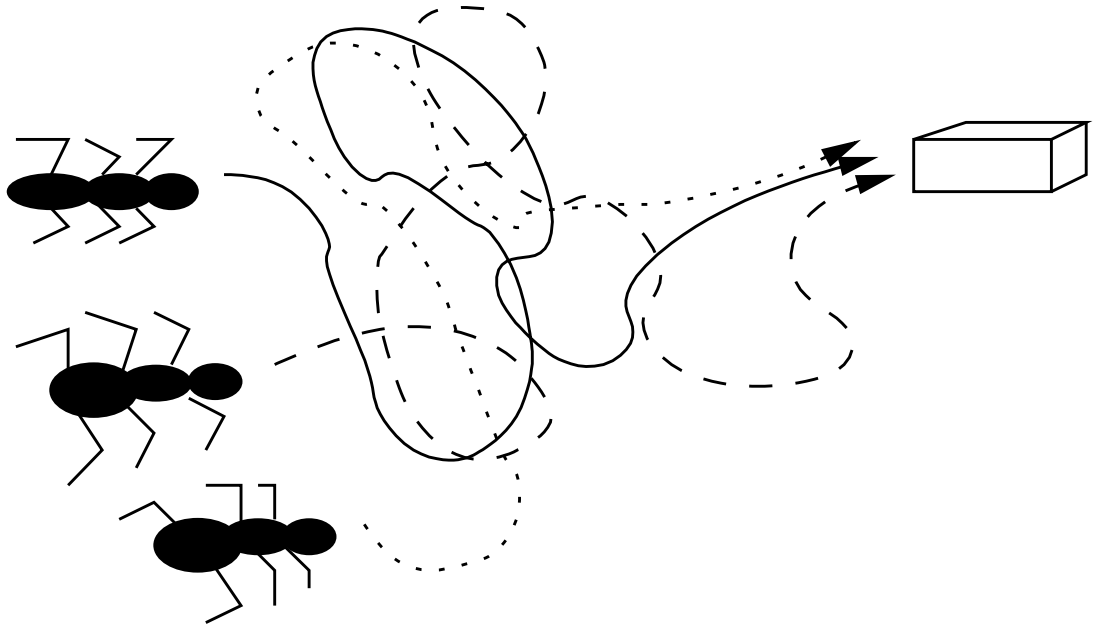
Process Algebra: An Algebraic Theory of Concurrency



$$x \| y = y \| x$$

A Very Brief History of Process Algebra

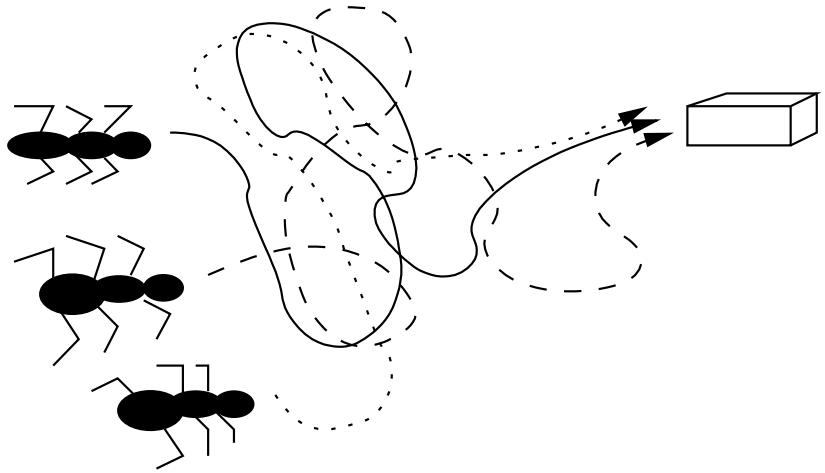
Late 70's: Hoare and Milner developed process algebras CSP and CCS, to specify and formally reason about distributed systems.

Early 80's: Bergstra and Klop developed the process algebra ACP.

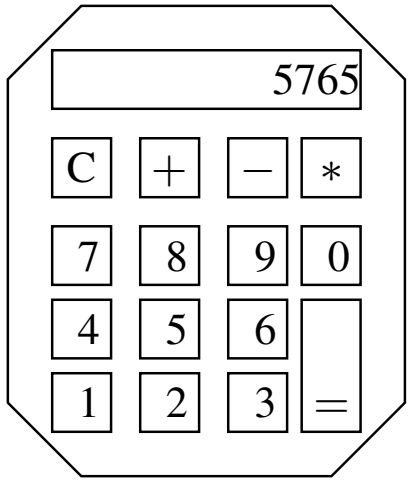
1982: Bergstra and Klop coined the term *process algebra*.

Examples of Concurrent Systems

A colony of ants:



A pocket calculator:



Process Graphs

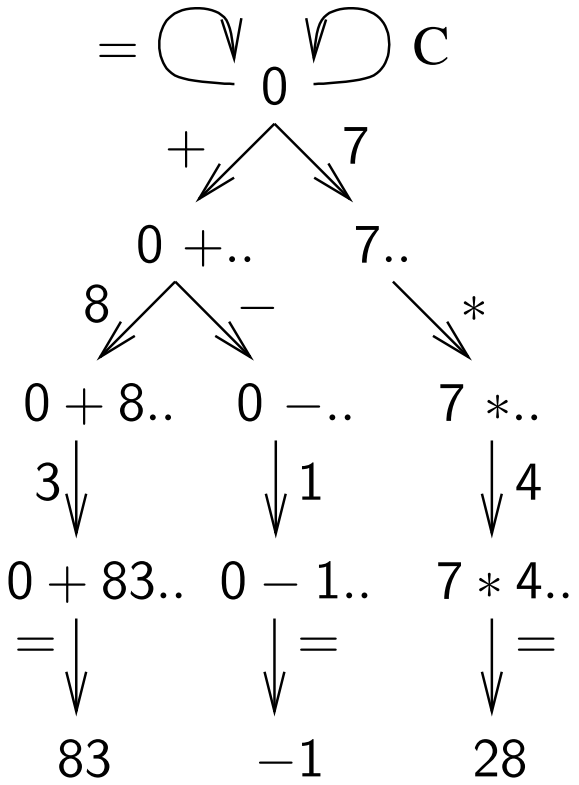
Given a set S of states, and a finite set A of (atomic) actions.

- A transition $s \xrightarrow{a} s'$ expresses that state s can evolve into state s' by execution of action a.
- A transition $s \xrightarrow{a} \surd$ expresses that state s can terminate successfully by execution of action a.

A process graph is a set of transitions, with a special root state.

Example

Part of the process graph of the pocket calculator:



This process graph is built from the process graphs of the buttons on the calculator.

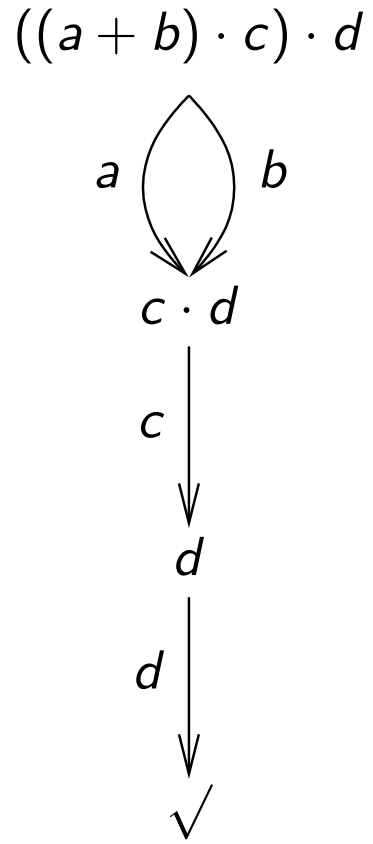
Basic Process Terms

Basic process terms are built from atomic actions, *alternative composition* and *sequential composition*.

- An atomic action a represents indivisible behaviour. It executes itself and then terminates successfully: $a \xrightarrow{a} \surd$.
- Alternative composition: the process term $t_1 + t_2$ executes the behaviour of either t_1 or t_2.
- Sequential composition: the process term $t_1 \cdot t_2$ first executes t_1, and upon successful termination proceeds to execute t_2.

Example

The basic process term $((a+b)\cdot c)\cdot d$ represents the process graph:



Transition Rules of Basic Process Algebra

$$\frac{}{v \xrightarrow{v} \surd}$$

$$\frac{x \xrightarrow{v} \surd}{x + y \xrightarrow{v} \surd} \qquad \frac{x \xrightarrow{v} x'}{x + y \xrightarrow{v} x'}$$

$$\frac{y \xrightarrow{v} \surd}{x + y \xrightarrow{v} \surd} \qquad \frac{y \xrightarrow{v} y'}{x + y \xrightarrow{v} y'}$$

$$\frac{x \xrightarrow{v} \surd}{x \cdot y \xrightarrow{v} y} \qquad \frac{x \xrightarrow{v} x'}{x \cdot y \xrightarrow{v} x' \cdot y}$$

Example Derivation of a Transition

Proof of $((a + b) \cdot c) \cdot d \xrightarrow{b} c \cdot d$ from the transition rules:

$$\dfrac{\dfrac{\dfrac{b \xrightarrow{b} \surd}{a + b \xrightarrow{b} \surd}}{(a + b) \cdot c \xrightarrow{b} c}}{((a + b) \cdot c) \cdot d \xrightarrow{b} c \cdot d}$$

$$\frac{}{v \xrightarrow{v} \surd}$$

$$\frac{y \xrightarrow{v} \surd}{x + y \xrightarrow{v} \surd}$$

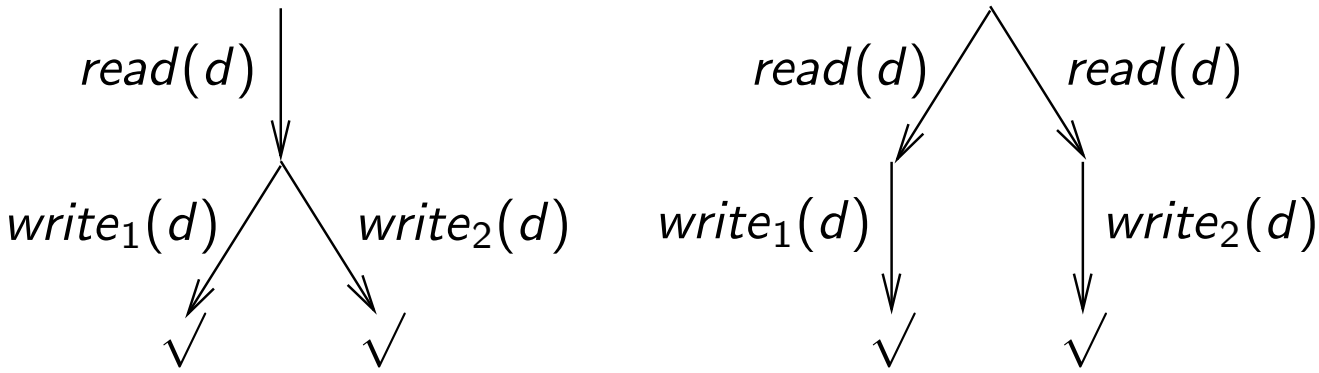
$$\frac{x \xrightarrow{v} \surd}{x \cdot y \xrightarrow{v} y}$$

$$\frac{x \xrightarrow{v} x'}{x \cdot y \xrightarrow{v} x' \cdot y}$$

What is a Suitable Process Equivalence?

A process equivalence based on traces is not satisfactory.

Example:



Both root states display $read(d)\, write_1(d)$ and $read(d)\, write_2(d)$, so they are trace equivalent.

A crucial distinction becomes apparent if disc 1 crashes.
Then the first process always saves datum d on disc 2,
while the second process may get stuck.

Bisimulation Equivalence

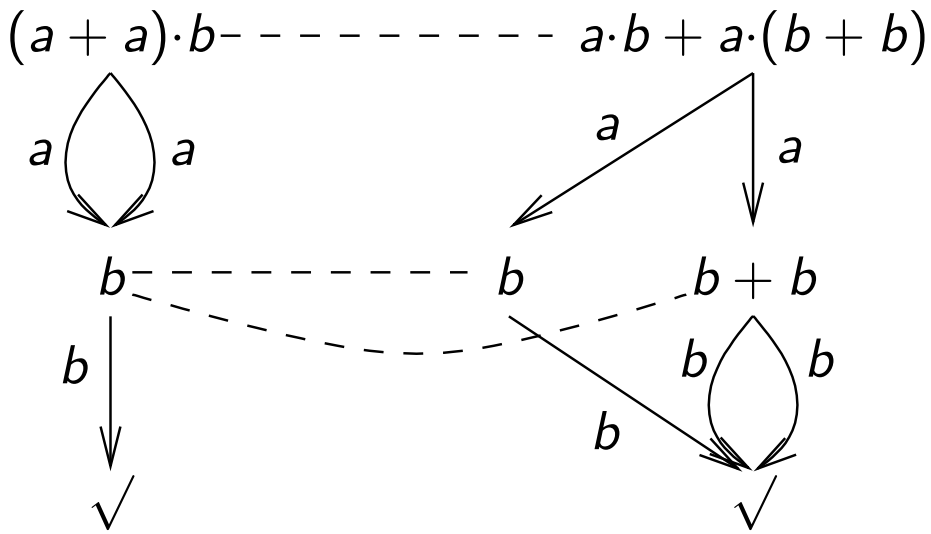
A bisimulation is a binary relation $\mathcal{B}$ on states such that:

1. if $p \mathcal{B} q$ and $p \xrightarrow{a} p'$, then $q \xrightarrow{a} q'$ with $p' \mathcal{B} q'$
2. if $p \mathcal{B} q$ and $q \xrightarrow{a} q'$, then $p \xrightarrow{a} p'$ with $p' \mathcal{B} q'$
3. if $p \mathcal{B} q$ and $p \xrightarrow{a} \surd$, then $q \xrightarrow{a} \surd$
4. if $p \mathcal{B} q$ and $q \xrightarrow{a} \surd$, then $p \xrightarrow{a} \surd$

p and q are bisimilar, denoted $p \underline{\leftrightarrow} q$, if there is a bisimulation relation $\mathcal{B}$ such that $p \mathcal{B} q$.

Example

$(a + a)\cdot b \leftrightarrows a\cdot b + a\cdot(b + b)$



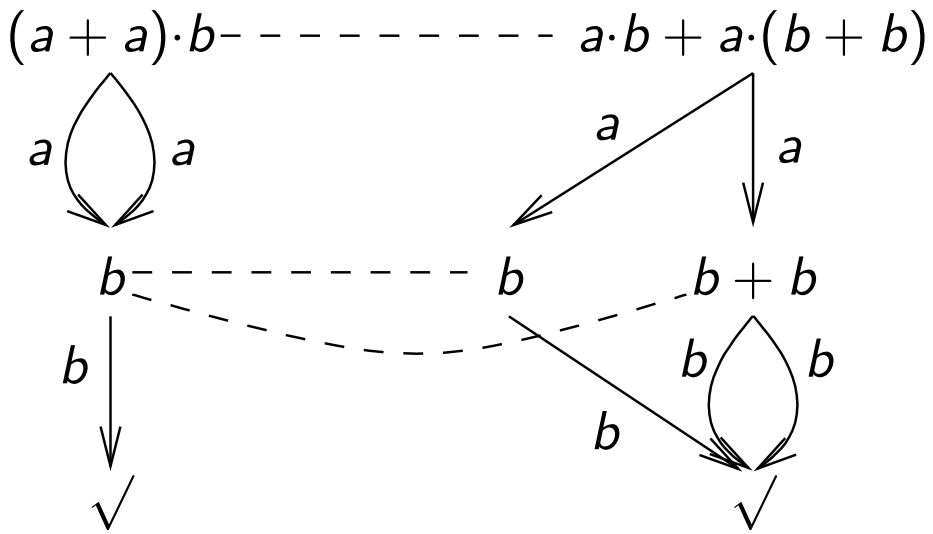
Bisimulation relation $\mathcal{B}$ is defined by: $(a + a)\cdot b \ \mathcal{B} \ a\cdot b + a\cdot(b + b)$

$b \ \mathcal{B} \ b$

$b \ \mathcal{B} \ b + b.$

Example

$(a + a)\cdot b \;\underline{\leftrightarrow}\; a\cdot b + a\cdot(b + b)$



Bisimulation relation $\mathcal{B}$ is defined by: $(a + a)\cdot b \;\mathcal{B}\; a\cdot b + a\cdot(b + b)$

$b \;\mathcal{B}\; b$

$b \;\mathcal{B}\; b + b.$

Hennessy-Milner Logic

Two process terms are bisimilar if and only if they make true exactly the same formulas in Hennessy-Milner Logic

Congruence

Bisimulation equivalence is a congruence over BPA.

That is, if $s \underline{\leftrightarrow} s'$ and $t \underline{\leftrightarrow} t'$, then $s + t \underline{\leftrightarrow} s' + t'$ and $s \cdot t \underline{\leftrightarrow} s' \cdot t'$.

This can actually be deduced from the syntactic form of the transition rules for BPA.

All upcoming process algebraic operators are congruent with respect to bisimulation equivalence.

Congruence

Bisimulation equivalence is a congruence over BPA.

That is, if $s \underline{\leftrightarrow} s'$ and $t \underline{\leftrightarrow} t'$, then $s + t \underline{\leftrightarrow} s' + t'$ and $s \cdot t \underline{\leftrightarrow} s' \cdot t'$.

This can actually be deduced from the syntactic form of the transition rules for BPA.

All upcoming process algebraic operators are congruent with respect to bisimulation equivalence.

Congruence

Bisimulation equivalence is a congruence over BPA.

That is, if $s \underline{\leftrightarrow} s'$ and $t \underline{\leftrightarrow} t'$, then $s + t \underline{\leftrightarrow} s' + t'$ and $s \cdot t \underline{\leftrightarrow} s' \cdot t'$.

This can actually be deduced from the syntactic form of the transition rules for BPA.

All upcoming process algebraic operators are congruent with respect to bisimulation equivalence.

Axioms for BPA Modulo Bisimulation

$$
\begin{array}{lrcl}
\text{A1} & x + y & = & y + x \\
\text{A2} & (x + y) + z & = & x + (y + z) \\
\text{A3} & x + x & = & x \\
\text{A4} & (x + y) \cdot z & = & x \cdot z + y \cdot z \\
\text{A5} & (x \cdot y) \cdot z & = & x \cdot (y \cdot z)
\end{array}
$$

Equality Relation

The axioms induce an equality relation $=$ on basic process terms:

- (Substitution)
  If $s = t$ is an axiom and σ maps all variables in s and t to basic process terms, then $\sigma(s) = \sigma(t)$.

- (Equivalence)

  $t = t$ for all basic process terms t

  if $s = t$, then $t = s$

  if $s = t$ and $t = u$, then $s = u$

- (Context)
  If $s = s'$ and $t = t'$, then $s + t = s' + t'$ and $s \cdot t = s' \cdot t'$.

Soundness of the Axioms

The axioms for BPA are sound modulo $\underline{\leftrightarrow}$:

$$\text{if } s = t \text{ then } s \underline{\leftrightarrow} t.$$

Since bisimulation equivalence is a congruence over BPA, soundness of the axioms can be verified by checking that $\sigma(s) \underline{\leftrightarrow} \sigma(t)$ for all axioms $s = t$ and substitutions σ.

Soundness of the Axioms

The axioms for BPA are sound modulo $\underline{\leftrightarrow}$:

$$\text{if } s = t \text{ then } s \underline{\leftrightarrow} t.$$

Since bisimulation equivalence is a congruence over BPA, soundness of the axioms can be verified by checking that $\sigma(s) \underline{\leftrightarrow} \sigma(t)$ for all axioms $s = t$ and substitutions σ.

Ground-Completeness of the Axioms

The axioms for BPA are ground-complete modulo $\underline{\leftrightarrow}$:

$$\text{if } s \mathrel{\underline{\leftrightarrow}} t \text{ then } s = t.$$

We write $s =_{\mathrm{AC}} t$ if s and t can be equated by axioms A1 and A2.

The axioms for BPA are turned into a term rewriting system, modulo AC of the $+$:

$$\begin{array}{rcl} x + y & =_{\mathrm{AC}} & y + x \\ (x + y) + z & =_{\mathrm{AC}} & x + (y + z) \end{array}$$

$$\begin{array}{rcl} x + x & \to & x \\ (x + y) \cdot z & \to & x \cdot z + y \cdot z \\ (x \cdot y) \cdot z & \to & x \cdot (y \cdot z) \end{array}$$

Each basic process term is reduced to a normal form.

If $s \mathrel{\underline{\leftrightarrow}} t$, and s and t have normal forms s' and t', respectively, then $s' =_{\mathrm{AC}} t'$. Hence,

$$s = s' =_{\mathrm{AC}} t' = t$$

Ground-Completeness of the Axioms

The axioms for BPA are ground-complete modulo $\underline{\leftrightarrow}$:

$$\text{if } s \underline{\leftrightarrow} t \text{ then } s = t.$$

We write $s =_{\mathrm{AC}} t$ if s and t can be equated by axioms A1 and A2.

The axioms for BPA are turned into a term rewriting system, modulo AC of the $+$:

$$\begin{array}{rcl} x + y & =_{\mathrm{AC}} & y + x \\ (x + y) + z & =_{\mathrm{AC}} & x + (y + z) \end{array} \qquad \begin{array}{rcl} x + x & \to & x \\ (x + y) \cdot z & \to & x \cdot z + y \cdot z \\ (x \cdot y) \cdot z & \to & x \cdot (y \cdot z) \end{array}$$

Each basic process term is reduced to a normal form.

If $s \underline{\leftrightarrow} t$, and s and t have normal forms s' and t', respectively, then $s' =_{\mathrm{AC}} t'$. Hence,

$$s = s' =_{\mathrm{AC}} t' = t$$

Example

$$(a+a)\cdot(c\cdot d)+(b\cdot c)\cdot(d+d) \;=\; ((b+a)\cdot(c+c))\cdot d$$

$$\begin{aligned}
& (\underline{a+a})\cdot(c\cdot d)+(b\cdot c)\cdot(d+d) \\
\overset{\mathrm{A3}}{\to}\ & a\cdot(c\cdot d)+(b\cdot c)\cdot(\underline{d+d}) \\
\overset{\mathrm{A3}}{\to}\ & a\cdot(c\cdot d)+\underline{(b\cdot c)\cdot d} \\
\overset{\mathrm{A5}}{\to}\ & a\cdot(c\cdot d)+b\cdot(c\cdot d)
\end{aligned}$$

$$\begin{aligned}
& ((b+a)\cdot(\underline{c+c}))\cdot d \\
\overset{\mathrm{A3}}{\to}\ & \underline{((b+a)\cdot c)\cdot d} \\
\overset{\mathrm{A5}}{\to}\ & \underline{(b+a)\cdot(c\cdot d)} \\
\overset{\mathrm{A4}}{\to}\ & b\cdot(c\cdot d)+a\cdot(c\cdot d)
\end{aligned}$$

The two resulting normal forms are equal modulo AC of the $+$, so the two original basic process terms are equal.

Communication

A communication function $\gamma : A \times A \rightarrow A$ produces for each pair of atomic actions a and b their communication $\gamma(a, b)$.

The communication function γ is commutative and associative:

$$\begin{aligned} \gamma(a, b) &\equiv \gamma(b, a) \\ \gamma(\gamma(a, b), c) &\equiv \gamma(a, \gamma(b, c)) \end{aligned}$$

Merge

The merge $\|$ executes its arguments in parallel:

$$\frac{x \xrightarrow{v} \surd}{x\|y \xrightarrow{v} y} \qquad \frac{x \xrightarrow{v} x'}{x\|y \xrightarrow{v} x'\|y} \qquad \frac{y \xrightarrow{v} \surd}{x\|y \xrightarrow{v} x} \qquad \frac{y \xrightarrow{v} y'}{x\|y \xrightarrow{v} x\|y'}$$

The merge can also execute a communication between initial transitions of its arguments:

$$\frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} \surd}{x\|y \xrightarrow{\gamma(v,w)} \surd} \qquad \frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} y'}{x\|y \xrightarrow{\gamma(v,w)} y'}$$

$$\frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} \surd}{x\|y \xrightarrow{\gamma(v,w)} x'} \qquad \frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} y'}{x\|y \xrightarrow{\gamma(v,w)} x'\|y'}$$

Merge

The merge $\|$ executes its arguments in parallel:

$$\frac{x \xrightarrow{v} \surd}{x\|y \xrightarrow{v} y} \qquad \frac{x \xrightarrow{v} x'}{x\|y \xrightarrow{v} x'\|y} \qquad \frac{y \xrightarrow{v} \surd}{x\|y \xrightarrow{v} x} \qquad \frac{y \xrightarrow{v} y'}{x\|y \xrightarrow{v} x\|y'}$$

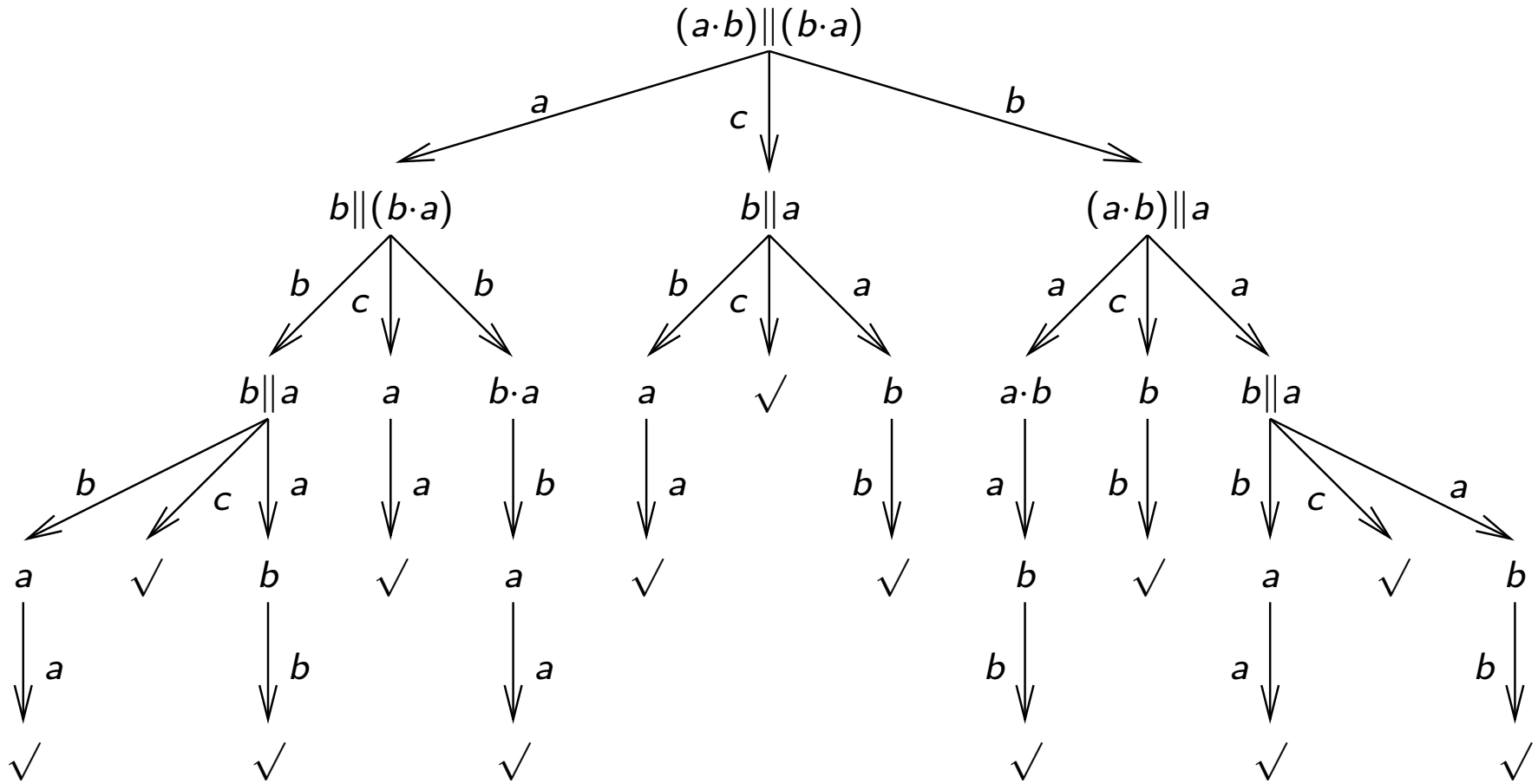
The merge can also execute a communication between initial transitions of its arguments:

$$\frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} \surd}{x\|y \xrightarrow{\gamma(v,w)} \surd} \qquad \frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} y'}{x\|y \xrightarrow{\gamma(v,w)} y'}$$

$$\frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} \surd}{x\|y \xrightarrow{\gamma(v,w)} x'} \qquad \frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} y'}{x\|y \xrightarrow{\gamma(v,w)} x'\|y'}$$

Example

Let $\gamma(a,a) \equiv \gamma(a,b) \equiv \gamma(b,b) \equiv c$. The process graph of $(a{\cdot}b)\|(b{\cdot}a)$ is:



Left Merge and Communication Merge

Two auxiliary operators are needed to axiomatise the merge.

The left merge $\between$ executes an initial transition of its first argument:

$$\frac{x \xrightarrow{v} \surd}{x \between y \xrightarrow{v} y} \qquad \frac{x \xrightarrow{v} x'}{x \between y \xrightarrow{v} x' \| y}$$

The communication merge $|$ executes a communication between initial transitions of its arguments:

$$\frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} \surd}{x|y \xrightarrow{\gamma(v,w)} \surd} \qquad \frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} y'}{x|y \xrightarrow{\gamma(v,w)} y'} \qquad \frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} \surd}{x|y \xrightarrow{\gamma(v,w)} x'} \qquad \frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} y'}{x|y \xrightarrow{\gamma(v,w)} x' \| y'}$$

Left Merge and Communication Merge

Two auxiliary operators are needed to axiomatise the merge.

The left merge $\between$ executes an initial transition of its first argument:

$$\frac{x \xrightarrow{v} \surd}{x \between y \xrightarrow{v} y} \qquad \frac{x \xrightarrow{v} x'}{x \between y \xrightarrow{v} x' \| y}$$

The communication merge $|$ executes a communication between initial transitions of its arguments:

$$\frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} \surd}{x|y \xrightarrow{\gamma(v,w)} \surd} \qquad \frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} y'}{x|y \xrightarrow{\gamma(v,w)} y'} \qquad \frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} \surd}{x|y \xrightarrow{\gamma(v,w)} x'} \qquad \frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} y'}{x|y \xrightarrow{\gamma(v,w)} x' \| y'}$$

Conservative Extension

PAP is a conservative extension of BPA, i.e., the process graphs of basic process terms remain the same.

This can again be deduced from the syntactic form of the transition rules for BPA and PAP.

All upcoming process algebraic extensions are conservative.

Remark: $+$ binds weaker than other operators.

Conservative Extension

PAP is a conservative extension of BPA, i.e., the process graphs of basic process terms remain the same.

This can again be deduced from the syntactic form of the transition rules for BPA and PAP.

All upcoming process algebraic extensions are conservative.

Remark: $+$ binds weaker than other operators.

Conservative Extension

PAP is a conservative extension of BPA, i.e., the process graphs of basic process terms remain the same.

This can again be deduced from the syntactic form of the transition rules for BPA and PAP.

All upcoming process algebraic extensions are conservative.

Remark: $+$ binds weaker than other operators.

Conservative Extension

PAP is a conservative extension of BPA, i.e., the process graphs of basic process terms remain the same.

This can again be deduced from the syntactic form of the transition rules for BPA and PAP.

All upcoming process algebraic extensions are conservative.

Remark: $+$ binds weaker than other operators.

Axioms for PAP

$$\text{M1} \qquad x \| y = (x \between y + y \between x) + x | y$$

$$\begin{aligned}
\text{LM2} \qquad & v \between y = v \cdot y \\
\text{LM3} \qquad & (v \cdot x) \between y = v \cdot (x \| y) \\
\text{LM4} \qquad & (x + y) \between z = x \between z + y \between z
\end{aligned}$$

$$\begin{aligned}
\text{CM5} \qquad & v | w = \gamma(v, w) \\
\text{CM6} \qquad & v | (w \cdot y) = \gamma(v, w) \cdot y \\
\text{CM7} \qquad & (v \cdot x) | w = \gamma(v, w) \cdot x \\
\text{CM8} \qquad & (v \cdot x) | (w \cdot y) = \gamma(v, w) \cdot (x \| y) \\
\text{CM9} \qquad & (x + y) | z = x | z + y | z \\
\text{CM10} \qquad & x | (y + z) = x | y + x | z
\end{aligned}$$

Axioms for PAP

$$
\begin{array}{lrcl}
\text{M1} & x \| y & = & (x \between y + y \between x) + x|y \\
 & & & \\
\text{LM2} & v \between y & = & v \cdot y \\
\text{LM3} & (v \cdot x) \between y & = & v \cdot (x \| y) \\
\text{LM4} & (x + y) \between z & = & x \between z + y \between z \\
 & & & \\
\text{CM5} & v|w & = & \gamma(v, w) \\
\text{CM6} & v|(w \cdot y) & = & \gamma(v, w) \cdot y \\
\text{CM7} & (v \cdot x)|w & = & \gamma(v, w) \cdot x \\
\text{CM8} & (v \cdot x)|(w \cdot y) & = & \gamma(v, w) \cdot (x \| y) \\
\text{CM9} & (x + y)|z & = & x|z + y|z \\
\text{CM10} & x|(y + z) & = & x|y + x|z
\end{array}
$$

Axioms for PAP

$$
\begin{array}{lrcl}
\text{M1} & x \| y & = & (x \between y + y \between x) + x|y \\
 & & & \\
\text{LM2} & v \between y & = & v \cdot y \\
\text{LM3} & (v \cdot x) \between y & = & v \cdot (x \| y) \\
\text{LM4} & (x + y) \between z & = & x \between z + y \between z \\
 & & & \\
\text{CM5} & v|w & = & \gamma(v, w) \\
\text{CM6} & v|(w \cdot y) & = & \gamma(v, w) \cdot y \\
\text{CM7} & (v \cdot x)|w & = & \gamma(v, w) \cdot x \\
\text{CM8} & (v \cdot x)|(w \cdot y) & = & \gamma(v, w) \cdot (x \| y) \\
\text{CM9} & (x + y)|z & = & x|z + y|z \\
\text{CM10} & x|(y + z) & = & x|y + x|z
\end{array}
$$

Soundness and Ground-Completeness

The axioms for PAP are sound modulo $\underline{\leftrightarrow}$:

$$s = t \;\Rightarrow\; s \mathrel{\underline{\leftrightarrow}} t$$

Again soundness can be checked for each axiom separately.

The axioms for PAP are ground-complete modulo $\underline{\leftrightarrow}$:

$$s \mathrel{\underline{\leftrightarrow}} t \;\Rightarrow\; s = t$$

Again ground-completeness can be proved using a term rewriting approach, to eliminate parallel operators from process terms.

Soundness and Ground-Completeness

The axioms for PAP are sound modulo $\underline{\leftrightarrow}$:

$$s = t \;\Rightarrow\; s \mathrel{\underline{\leftrightarrow}} t$$

Again soundness can be checked for each axiom separately.

The axioms for PAP are ground-complete modulo $\underline{\leftrightarrow}$:

$$s \mathrel{\underline{\leftrightarrow}} t \;\Rightarrow\; s = t$$

Again ground-completeness can be proved using a term rewriting approach, to eliminate parallel operators from process terms.

Deadlock

The deadlock δ does not display any behaviour.

There is no transition rule for this constant.

The domain of the communication function γ is extended with δ:

$$\gamma : A \times A \to A \cup \{\delta\}.$$

If a and b do not communicate, then $\gamma(a, b) \equiv \delta$.

If $\gamma(a, b) \equiv \delta$, then $a|b = \delta$, so in that case $a|b$ does not display any transitions.

Deadlock

The deadlock δ does not display any behaviour.

There is no transition rule for this constant.

The domain of the communication function γ is extended with δ:

$$\gamma : A \times A \to A \cup \{\delta\}.$$

If a and b do not communicate, then $\gamma(a, b) \equiv \delta$.

If $\gamma(a, b) \equiv \delta$, then $a|b = \delta$, so in that case $a|b$ does not display any transitions.

Encapsulation

The encapsulation operator ∂_H, with $H \subseteq A$, renames all actions from H in its argument into δ:

$$\frac{x \xrightarrow{v} \surd \quad (v \notin H)}{\partial_H(x) \xrightarrow{v} \surd} \qquad \frac{x \xrightarrow{v} x' \quad (v \notin H)}{\partial_H(x) \xrightarrow{v} \partial_H(x')}$$

Encapsulation operators can be used to force actions into communication.

Example: $\partial_{\{a,b\}}(a \| b)$ can only execute $\gamma(a, b)$

(assuming that $\gamma(a, b) \notin \{a, b, \delta\}$).

Encapsulation

The encapsulation operator ∂_H, with $H \subseteq A$, renames all actions from H in its argument into δ:

$$\frac{x \xrightarrow{v} \surd \quad (v \notin H)}{\partial_H(x) \xrightarrow{v} \surd} \qquad \frac{x \xrightarrow{v} x' \quad (v \notin H)}{\partial_H(x) \xrightarrow{v} \partial_H(x')}$$

Encapsulation operators can be used to force actions into communication.

Example: $\partial_{\{a,b\}}(a \| b)$ can only execute $\gamma(a, b)$

(assuming that $\gamma(a, b) \notin \{a, b, \delta\}$).

Axioms for Deadlock and Encapsulation

$$
\begin{array}{lrcll}
\text{A6} & x + \delta & = & x & \\
\text{A7} & \delta \cdot x & = & \delta & \\
 & & & & \\
\text{D1} & \partial_H(v) & = & v & (v \notin H) \\
\text{D2} & \partial_H(v) & = & \delta & (v \in H) \\
\text{D3} & \partial_H(\delta) & = & \delta & \\
\text{D4} & \partial_H(x + y) & = & \partial_H(x) + \partial_H(y) & \\
\text{D5} & \partial_H(x \cdot y) & = & \partial_H(x) \cdot \partial_H(y) & \\
 & & & & \\
\text{LM11} & \delta \between x & = & \delta & \\
\text{CM12} & \delta | x & = & \delta & \\
\text{CM13} & x | \delta & = & \delta &
\end{array}
$$

The axioms are *sound* and *ground-complete* for ACP modulo $\leftrightarrow$.

Axioms for Deadlock and Encapsulation

$$
\begin{array}{lrcll}
\text{A6} & x+\delta & = & x & \\
\text{A7} & \delta\cdot x & = & \delta & \\
 & & & & \\
\text{D1} & \partial_H(v) & = & v & (v\notin H)\\
\text{D2} & \partial_H(v) & = & \delta & (v\in H)\\
\text{D3} & \partial_H(\delta) & = & \delta & \\
\text{D4} & \partial_H(x+y) & = & \partial_H(x)+\partial_H(y) & \\
\text{D5} & \partial_H(x\cdot y) & = & \partial_H(x)\cdot\partial_H(y) & \\
 & & & & \\
\text{LM11} & \delta\between x & = & \delta & \\
\text{CM12} & \delta|x & = & \delta & \\
\text{CM13} & x|\delta & = & \delta &
\end{array}
$$

The axioms are *sound* and *ground-complete* for ACP modulo $\underline{\leftrightarrow}$.

Example

Let $\gamma(a, b) \equiv c$ be the only communication between actions.

$$\begin{aligned} a \| b &\overset{\text{M1}}{=} a \between b + b \between a + a | b \\ &\overset{\text{LM2, CM5}}{=} a \cdot b + b \cdot a + c \end{aligned}$$

$$\begin{aligned} \partial_{\{a,b\}}(a \| b) &= \partial_{\{a,b\}}(a \cdot b + b \cdot a + c) \\ &\overset{\text{D1,2,4,5}}{=} \delta \cdot \partial_{\{a,b\}}(b) + \delta \cdot \partial_{\{a,b\}}(a) + c \\ &\overset{\text{A6,7}}{=} c \end{aligned}$$

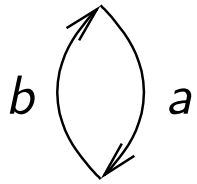
Example

Let $\gamma(a, b) \equiv c$ be the only communication between actions.

$$\begin{aligned} a \| b &\overset{\text{M1}}{=} a \between b + b \between a + a | b \\ &\overset{\text{LM2, CM5}}{=} a \cdot b + b \cdot a + c \end{aligned}$$

$$\begin{aligned} \partial_{\{a,b\}}(a \| b) &= \partial_{\{a,b\}}(a \cdot b + b \cdot a + c) \\ &\overset{\text{D1,2,4,5}}{=} \delta \cdot \partial_{\{a,b\}}(b) + \delta \cdot \partial_{\{a,b\}}(a) + c \\ &\overset{\text{A6,7}}{=} c \end{aligned}$$

Recursion



This process graph is captured by two recursive equations:

$$\begin{aligned} X &= a \cdot Y \\ Y &= b \cdot X \end{aligned}$$

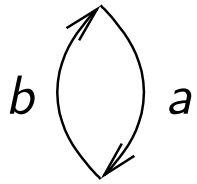
X and Y are recursion variables, representing the two states.

A recursive specification is a collection

$$\begin{aligned} X_1 &= t_1(X_1, \ldots, X_n) \\ &\vdots \\ X_n &= t_n(X_1, \ldots, X_n) \end{aligned}$$

where the $t_i(X_1, \ldots, X_n)$ are process terms over ACP with possible occurrences of the recursion variables $X_1, \ldots, X_n$.

Recursion



This process graph is captured by two recursive equations:

$$\begin{aligned} X &= a \cdot Y \\ Y &= b \cdot X \end{aligned}$$

X and Y are recursion variables, representing the two states.

A recursive specification is a collection

$$\begin{aligned} X_1 &= t_1(X_1, \ldots, X_n) \\ &\vdots \\ X_n &= t_n(X_1, \ldots, X_n) \end{aligned}$$

where the $t_i(X_1, \ldots, X_n)$ are process terms over ACP with possible occurrences of the recursion variables $X_1, \ldots, X_n$.

Solution of a Recursive Specification

Processes $p_1, \ldots, p_n$ are a solution modulo $\underline{\leftrightarrow}$ of

$$\{X_i = t_i(X_1, \ldots, X_n) \mid i \in \{1, \ldots, n\}\}$$

if $p_i \underline{\leftrightarrow} t_i(p_1, \ldots, p_n)$ for $i \in \{1, \ldots, n\}$.

If $p_1, \ldots, p_n$ and $q_1, \ldots, q_n$ are two solutions of the same recursive specification, then we want that $p_i \underline{\leftrightarrow} q_i$ for $i \in \{1, \ldots, n\}$.

A recursive specification with multiple solutions modulo $\underline{\leftrightarrow}$ is

$$X = X$$

In contrast, $X = a \cdot X$ has a unique solution modulo $\underline{\leftrightarrow}$.

Solution of a Recursive Specification

Processes $p_1, \ldots, p_n$ are a solution modulo $\underline{\leftrightarrow}$ of

$$\{X_i = t_i(X_1, \ldots, X_n) \mid i \in \{1, \ldots, n\}\}$$

if $p_i \underline{\leftrightarrow} t_i(p_1, \ldots, p_n)$ for $i \in \{1, \ldots, n\}$.

If $p_1, \ldots, p_n$ and $q_1, \ldots, q_n$ are two solutions of the same recursive specification, then we want that $p_i \underline{\leftrightarrow} q_i$ for $i \in \{1, \ldots, n\}$.

A recursive specification with multiple solutions modulo $\underline{\leftrightarrow}$ is

$$X = X$$

In contrast, $X = a \cdot X$ has a unique solution modulo $\underline{\leftrightarrow}$.

Guardedness

A recursive specification

$$\{X_i = t_i(X_1, \ldots, X_n) \mid i \in \{1, \ldots, n\}\}$$

is guarded if the right-hand sides of its recursive equations can be adapted to the form

$$a_1 \cdot s_1(X_1, \ldots, X_n) + \cdots + a_k \cdot s_k(X_1, \ldots, X_n) + b_1 + \cdots + b_\ell$$

using the axioms of ACP and the possibility to replace recursion variables by the right-hand sides of their recursive equations.

Guarded recursive specifications are exactly the ones that have a unique solution modulo $\underline{\leftrightarrow}$.

Transition Rules for Recursion

Given a *guarded* recursive specification E:

$$\{X_i = t_i(X_1, \ldots, X_n) \mid i \in \{1, \ldots, n\}\}$$

We introduce constants

$$\langle X_i | E \rangle \qquad (i \in \{1, \ldots, n\})$$

representing a solution of X_i in E.

$\langle X_i | E \rangle$ inherits the behaviour of $t_i(\langle X_1 | E \rangle, \ldots, \langle X_n | E \rangle)$:

$$\frac{t_i(\langle X_1 | E \rangle, \ldots, \langle X_n | E \rangle) \xrightarrow{v} \surd}{\langle X_i | E \rangle \xrightarrow{v} \surd} \qquad \frac{t_i(\langle X_1 | E \rangle, \ldots, \langle X_n | E \rangle) \xrightarrow{v} y}{\langle X_i | E \rangle \xrightarrow{v} y}$$

Transition Rules for Recursion

Given a *guarded* recursive specification E:

$$\{X_i = t_i(X_1, \ldots, X_n) \mid i \in \{1, \ldots, n\}\}$$

We introduce constants

$$\langle X_i | E \rangle \qquad (i \in \{1, \ldots, n\})$$

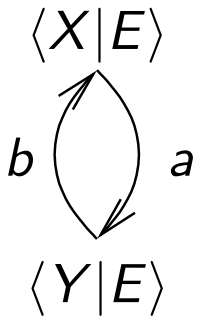
representing a solution of X_i in E.

$\langle X_i | E \rangle$ inherits the behaviour of $t_i(\langle X_1 | E \rangle, \ldots, \langle X_n | E \rangle)$:

$$\frac{t_i(\langle X_1 | E \rangle, \ldots, \langle X_n | E \rangle) \xrightarrow{v} \surd}{\langle X_i | E \rangle \xrightarrow{v} \surd} \qquad \frac{t_i(\langle X_1 | E \rangle, \ldots, \langle X_n | E \rangle) \xrightarrow{v} y}{\langle X_i | E \rangle \xrightarrow{v} y}$$

Example

Let E denote $\{X{=}a{\cdot}Y,\ Y{=}b{\cdot}X\}$. The process graph of $\langle X|E\rangle$ is:



We derive the transition $\langle X|E\rangle \xrightarrow{a} \langle Y|E\rangle$:

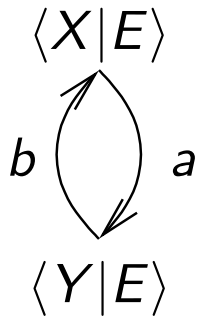
$$\frac{}{a \xrightarrow{a} \surd} \qquad \frac{}{v \xrightarrow{v} \surd}$$

$$\frac{}{a{\cdot}\langle Y|E\rangle \xrightarrow{a} \langle Y|E\rangle} \qquad \frac{x \xrightarrow{v} \surd}{x{\cdot}y \xrightarrow{v} y}$$

$$\frac{}{\langle X|E\rangle \xrightarrow{a} \langle Y|E\rangle} \qquad \frac{a{\cdot}\langle Y|E\rangle \xrightarrow{v} y}{\langle X|E\rangle \xrightarrow{v} y}$$

Example

Let E denote $\{X{=}a{\cdot}Y,\ Y{=}b{\cdot}X\}$. The process graph of $\langle X|E\rangle$ is:



We derive the transition $\langle X|E\rangle \xrightarrow{a} \langle Y|E\rangle$:

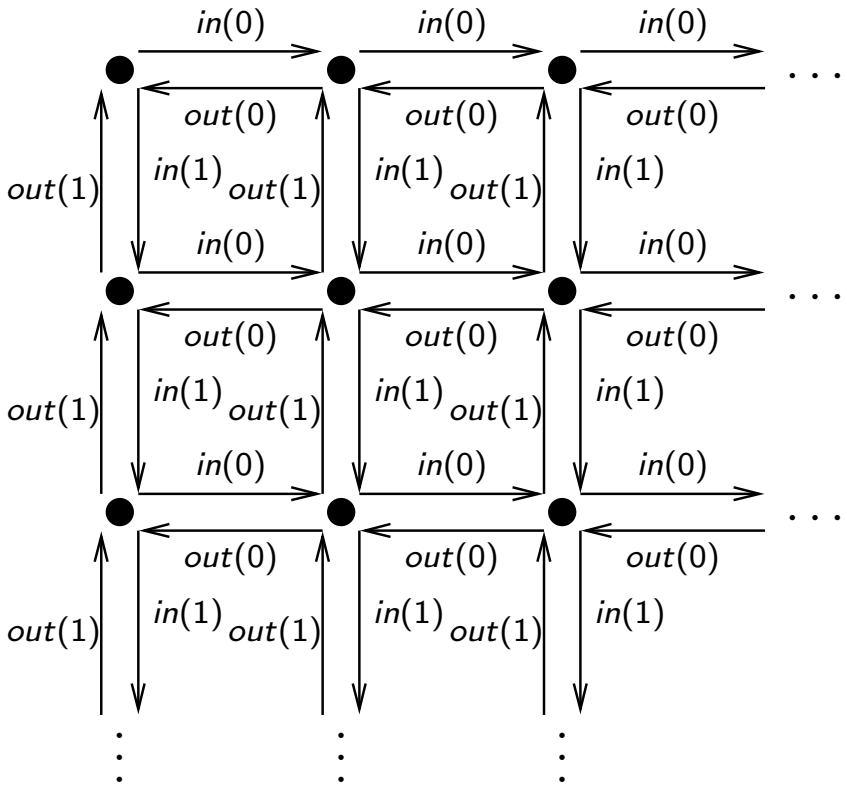
$$\frac{a \xrightarrow{a} \surd}{a{\cdot}\langle Y|E\rangle \xrightarrow{a} \langle Y|E\rangle}$$

$$\frac{a{\cdot}\langle Y|E\rangle \xrightarrow{a} \langle Y|E\rangle}{\langle X|E\rangle \xrightarrow{a} \langle Y|E\rangle}$$

$$\frac{}{v \xrightarrow{v} \surd} \qquad \frac{x \xrightarrow{v} \surd}{x{\cdot}y \xrightarrow{v} y} \qquad \frac{a{\cdot}\langle Y|E\rangle \xrightarrow{v} y}{\langle X|E\rangle \xrightarrow{v} y}$$

Example – Bag over $\{0, 1\}$

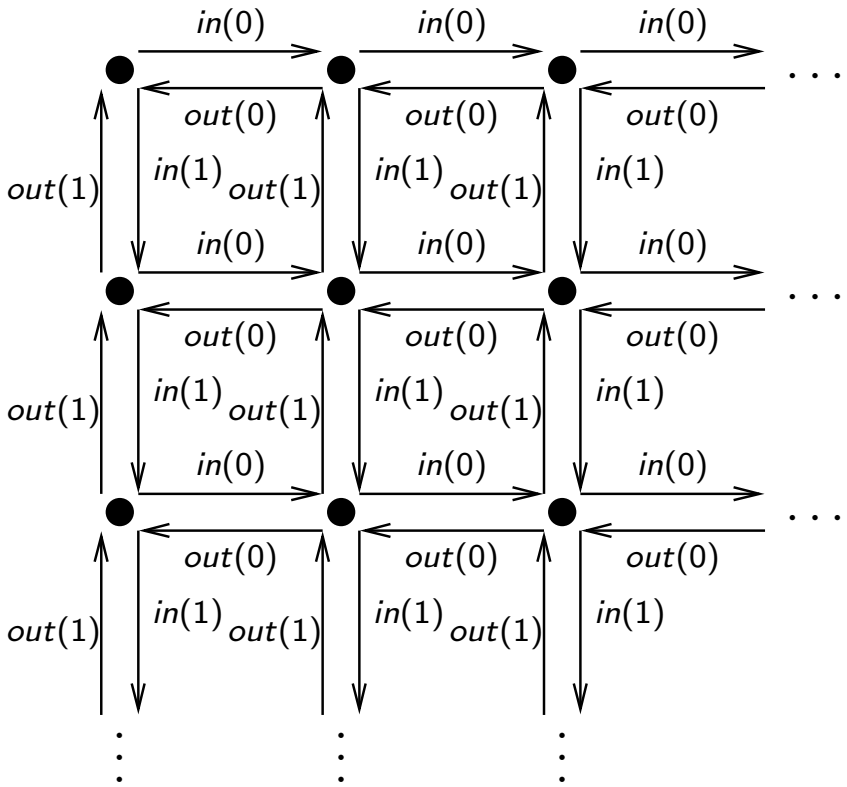
We can put elements 0 and 1 into a bag, and subsequently collect these elements from the bag in arbitrary order.



It is specified by $X = in(0) \cdot (X \| out(0)) + in(1) \cdot (X \| out(1))$.

Example – Bag over $\{0, 1\}$

We can put elements 0 and 1 into a bag, and subsequently collect these elements from the bag in arbitrary order.



It is specified by $X = in(0) \cdot (X \| out(0)) + in(1) \cdot (X \| out(1))$.

Axioms for Recursion

Given a *guarded* recursive specification E:

$$\{X_i = t_i(X_1, \ldots, X_n) \mid i \in \{1, \ldots, n\}\}$$

For $i \in \{1, \ldots, n\}$:

RDP $\quad \langle X_i | E \rangle = t_i(\langle X_1 | E \rangle, \ldots, \langle X_n | E \rangle)$

RSP $\quad$ If $y_j = t_j(y_1, \ldots, y_n)$ for all $j \in \{1, \ldots, n\}$, then $y_i = \langle X_i | E \rangle$

Soundness

The axioms for ACP with *guarded* recursion are sound modulo $\underline{\leftrightarrow}$:

$$s = t \;\Rightarrow\; s \mathrel{\underline{\leftrightarrow}} t$$

RSP is *not* sound for *unguarded* recursion.

For example, since $t = t$, RSP would yield

$$t \;=\; \langle X \,|\, X = X \rangle$$

for all process terms t.

Soundness

The axioms for ACP with *guarded* recursion are sound modulo $\underline{\leftrightarrow}$:

$$s = t \;\Rightarrow\; s \mathrel{\underline{\leftrightarrow}} t$$

RSP is *not* sound for *unguarded* recursion.

For example, since $t = t$, RSP would yield

$$t \;=\; \langle X \,|\, X = X \rangle$$

for all process terms t.

Example

$$\begin{aligned}
\langle Z \mid Z{=}a{\cdot}Z\rangle &\overset{\text{RDP}}{=} a{\cdot}\langle Z \mid Z{=}a{\cdot}Z\rangle \\
&\overset{\text{RDP}}{=} a{\cdot}(a{\cdot}\langle Z \mid Z{=}a{\cdot}Z\rangle) \\
&\overset{\text{A5}}{=} (a{\cdot}a){\cdot}\langle Z \mid Z{=}a{\cdot}Z\rangle
\end{aligned}$$

So by RSP,

$$\langle Z \mid Z{=}a{\cdot}Z\rangle \;=\; \langle X \mid X{=}(a{\cdot}a){\cdot}X\rangle$$

Regular Processes

A process p is regular if there are only *finitely* many processes p' such that $p \xrightarrow{a} \cdots \xrightarrow{b} p'$.

A recursive specification is linear if its recursive equations are of the form

$$X \;=\; a_1 \cdot X_1 + \cdots + a_k \cdot X_k + b_1 + \cdots + b_\ell$$

Each linear recursive specification gives rise to a regular process.

Vice versa, each regular process can be described by a linear recursive specification.

Ground-Completeness for Regular Processes

The axioms for ACP with *linear* recursion are ground-complete modulo $\underline{\leftrightarrow}$:

$$s \underline{\leftrightarrow} t \;\Rightarrow\; s = t$$

For *non-linear* recursion, the axioms are *not* ground-complete.

For example, one cannot derive

$$\langle Y \mid Y{=}a{\cdot}(Z{\cdot}b){+}b, Z{=}a{\cdot}(Y{\cdot}b){+}b\rangle \;=\; \langle Z \mid Y{=}a{\cdot}(Z{\cdot}b){+}b, Z{=}a{\cdot}(Y{\cdot}b){+}b\rangle$$

Non-linear recursion can be dealt with using the approximation induction principle, which states that if two processes are equal up to any finite depth, then the processes are equal.

Ground-Completeness for Regular Processes

The axioms for ACP with *linear* recursion are ground-complete modulo $\underline{\leftrightarrow}$:

$$s \underline{\leftrightarrow} t \;\Rightarrow\; s = t$$

For *non-linear* recursion, the axioms are *not* ground-complete.

For example, one cannot derive

$$\langle Y \mid Y{=}a{\cdot}(Z{\cdot}b){+}b, Z{=}a{\cdot}(Y{\cdot}b){+}b\rangle \;=\; \langle Z \mid Y{=}a{\cdot}(Z{\cdot}b){+}b, Z{=}a{\cdot}(Y{\cdot}b){+}b\rangle$$

Non-linear recursion can be dealt with using the approximation induction principle, which states that if two processes are equal up to any finite depth, then the processes are equal.

Ground-Completeness for Regular Processes

The axioms for ACP with *linear* recursion are ground-complete modulo $\underline{\leftrightarrow}$:

$$s \underline{\leftrightarrow} t \;\Rightarrow\; s = t$$

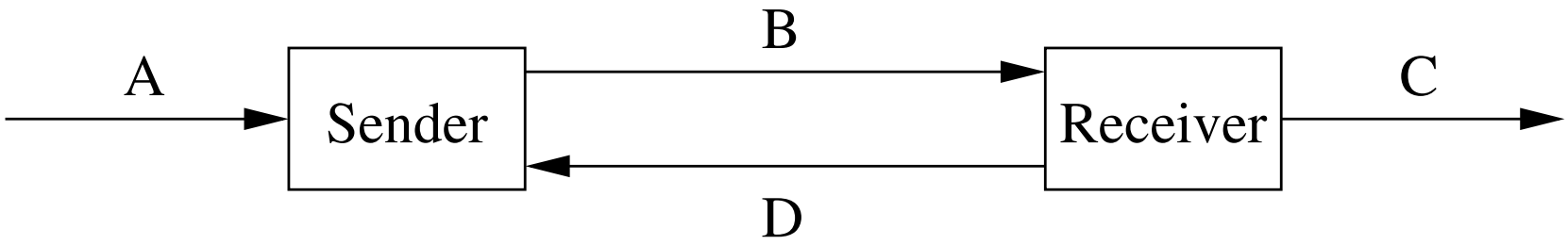
For *non-linear* recursion, the axioms are *not* ground-complete.

For example, one cannot derive

$$\langle Y \mid Y{=}a{\cdot}(Z{\cdot}b){+}b, Z{=}a{\cdot}(Y{\cdot}b){+}b\rangle \;=\; \langle Z \mid Y{=}a{\cdot}(Z{\cdot}b){+}b, Z{=}a{\cdot}(Y{\cdot}b){+}b\rangle$$

Non-linear recursion can be dealt with using the approximation induction principle, which states that if two processes are equal up to any finite depth, then the processes are equal.

Alternating Bit Protocol



Data elements are sent from *Sender* to *Receiver* via faulty channel B. *Sender* alternatingly attaches bit 0 or bit 1 to data elements.

If *Receiver* receives a datum, it sends the attached bit to *Sender* via faulty channel D, to acknowledge reception. If *Receiver* receives a corrupted message, then it resends the preceding acknowledgement.

Sender keeps sending a datum with attached bit b until it receives acknowledgement b. Then it starts sending the next datum with attached bit $1 - b$ until it receives acknowledgement $1 - b$, etc.

Correctness Criterion for the Alternating Bit Protocol

The data elements that are received by the *Sender* via channel A, are eventually sent into channel C by the *Receiver*, in the same order.

This assumes that a data element is not infinitely often corrupted. Namely, that would only happen if a channel were broken.

Specification of the Alternating Bit Protocol

Sender sending a datum with bit b attached:

$$
\begin{aligned}
S_b &= \sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot T_{db} \\
T_{db} &= (s_{\mathrm{B}}(d, b) + s_{\mathrm{B}}(\bot)) \cdot U_{db} \\
U_{db} &= r_{\mathrm{D}}(b) \cdot S_{1-b} \; + \; (r_{\mathrm{D}}(1-b) + r_{\mathrm{D}}(\bot)) \cdot T_{db}
\end{aligned}
$$

Receiver expecting a datum with b attached:

$$
\begin{aligned}
R_b &= \sum_{d \in \Delta} \{ r_{\mathrm{B}}(d, b) \cdot s_{\mathrm{C}}(d) \cdot Q_b \; + \; r_{\mathrm{B}}(d, 1-b) \cdot Q_{1-b} \} \\
&+ \; r_{\mathrm{B}}(\bot) \cdot Q_{1-b} \\
Q_b &= (s_{\mathrm{D}}(b) + s_{\mathrm{D}}(\bot)) \cdot R_{1-b}
\end{aligned}
$$

Specification of the Alternating Bit Protocol

Sender sending a datum with bit b attached:

$$
\begin{aligned}
S_b &= \sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot T_{db} \\
T_{db} &= (s_{\mathrm{B}}(d, b) + s_{\mathrm{B}}(\bot)) \cdot U_{db} \\
U_{db} &= r_{\mathrm{D}}(b) \cdot S_{1-b} \; + \; (r_{\mathrm{D}}(1-b) + r_{\mathrm{D}}(\bot)) \cdot T_{db}
\end{aligned}
$$

Receiver expecting a datum with b attached:

$$
\begin{aligned}
R_b &= \sum_{d \in \Delta} \{ r_{\mathrm{B}}(d, b) \cdot s_{\mathrm{C}}(d) \cdot Q_b \; + \; r_{\mathrm{B}}(d, 1-b) \cdot Q_{1-b} \} \\
&+ \; r_{\mathrm{B}}(\bot) \cdot Q_{1-b} \\
Q_b &= (s_{\mathrm{D}}(b) + s_{\mathrm{D}}(\bot)) \cdot R_{1-b}
\end{aligned}
$$

Specification of the Alternating Bit Protocol

A send and a read action of the same message ((d, b), b or $\perp$) over the same channel (B or D) communicate with each other.

The alternating bit protocol is specified by

$$\partial_H(R_0 \| S_0)$$

with H the set of send and read actions over channels B and D.

Specification of the Alternating Bit Protocol

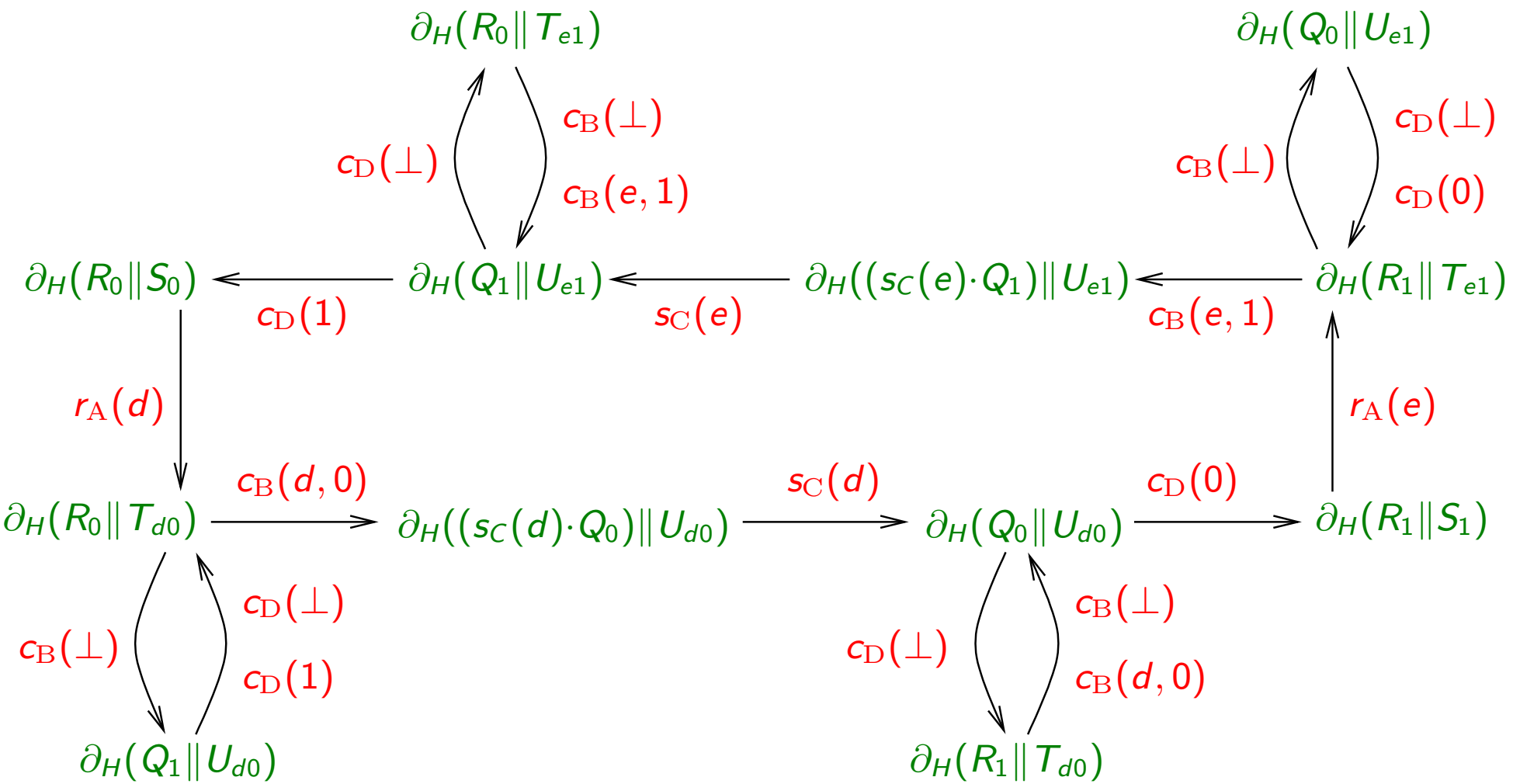
A send and a read action of the same message ((d, b), b or $\perp$) over the same channel (B or D) communicate with each other.

The alternating bit protocol is specified by

$$\partial_H(R_0 \| S_0)$$

with H the set of send and read actions over channels B and D.

Transition Graph of $\partial_H(R_0 \| S_0)$



Verification of the Alternating Bit Protocol

$$
\begin{aligned}
R_0 \| S_0 \;=\;& \textstyle\sum_{d' \in \Delta} \{ r_{\mathrm{B}}(d', 0) \cdot ((s_{\mathrm{C}}(d') \cdot Q_0) \| S_0) \;+\; r_{\mathrm{B}}(d', 1) \cdot (Q_1 \| S_0) \} \\
+\;& r_{\mathrm{B}}(\bot) \cdot (Q_1 \| S_0) \\
+\;& \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot (R_0 \| T_{d0})
\end{aligned}
$$

$$
\partial_H(R_0 \| S_0) \;=\; \sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot \partial_H(R_0 \| T_{d0})
$$

$$
\begin{aligned}
R_0 \| T_{d0} \;=\;& (s_{\mathrm{B}}(d, 0) + s_{\mathrm{B}}(\bot)) \cdot (R_0 \| U_{d0}) \\
+\;& \textstyle\sum_{d' \in \Delta} \{ r_{\mathrm{B}}(d', 0) \cdot ((s_{\mathrm{C}}(d') \cdot Q_0) \| T_{d0}) \;+\; r_{\mathrm{B}}(d', 1) \cdot (Q_1 \| T_{d0}) \} \\
+\;& r_{\mathrm{B}}(\bot) \cdot (Q_1 \| T_{d0}) \\
+\;& c_{\mathrm{B}}(d, 0) \cdot ((s_{\mathrm{C}}(d) \cdot Q_0) \| U_{d0}) \;+\; c_{\mathrm{B}}(\bot) \cdot (Q_1 \| U_{d0})
\end{aligned}
$$

$$
\partial_H(R_0 \| T_{d0}) \;=\; c_{\mathrm{B}}(d, 0) \cdot \partial_H((s_{\mathrm{C}}(d) \cdot Q_0) \| U_{d0}) \;+\; c_{\mathrm{B}}(\bot) \cdot \partial_H(Q_1 \| U_{d0})
$$

Verification of the Alternating Bit Protocol

$$
\begin{aligned}
R_0 \| S_0 \;=\;& \textstyle\sum_{d' \in \Delta} \{ r_{\mathrm{B}}(d', 0) \cdot ((s_{\mathrm{C}}(d') \cdot Q_0) \| S_0) \;+\; r_{\mathrm{B}}(d', 1) \cdot (Q_1 \| S_0) \} \\
+\;& r_{\mathrm{B}}(\bot) \cdot (Q_1 \| S_0) \\
+\;& \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot (R_0 \| T_{d0})
\end{aligned}
$$

$$
\partial_H(R_0 \| S_0) \;=\; \sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot \partial_H(R_0 \| T_{d0})
$$

$$
\begin{aligned}
R_0 \| T_{d0} \;=\;& (s_{\mathrm{B}}(d, 0) + s_{\mathrm{B}}(\bot)) \cdot (R_0 \| U_{d0}) \\
+\;& \textstyle\sum_{d' \in \Delta} \{ r_{\mathrm{B}}(d', 0) \cdot ((s_{\mathrm{C}}(d') \cdot Q_0) \| T_{d0}) \;+\; r_{\mathrm{B}}(d', 1) \cdot (Q_1 \| T_{d0}) \} \\
+\;& r_{\mathrm{B}}(\bot) \cdot (Q_1 \| T_{d0}) \\
+\;& c_{\mathrm{B}}(d, 0) \cdot ((s_{\mathrm{C}}(d) \cdot Q_0) \| U_{d0}) \;+\; c_{\mathrm{B}}(\bot) \cdot (Q_1 \| U_{d0})
\end{aligned}
$$

$$
\partial_H(R_0 \| T_{d0}) \;=\; c_{\mathrm{B}}(d, 0) \cdot \partial_H((s_{\mathrm{C}}(d) \cdot Q_0) \| U_{d0}) \;+\; c_{\mathrm{B}}(\bot) \cdot \partial_H(Q_1 \| U_{d0})
$$

Verification of the Alternating Bit Protocol

$$
\begin{aligned}
Q_1 \| U_{d0} &= r_{\mathrm{D}}(0)\cdot(Q_1 \| S_1) \\
&+ (r_{\mathrm{D}}(1) + r_{\mathrm{D}}(\bot))\cdot(Q_1 \| T_{d0}) \\
&+ (s_{\mathrm{D}}(1) + s_{\mathrm{D}}(\bot))\cdot(R_0 \| U_{d0}) \\
&+ (c_{\mathrm{D}}(1) + c_{\mathrm{D}}(\bot))\cdot(R_0 \| T_{d0})
\end{aligned}
$$

$$
\partial_H(Q_1 \| U_{d0}) = (c_{\mathrm{D}}(1) + c_{\mathrm{D}}(\bot))\cdot\partial_H(R_0 \| T_{d0})
$$

$$
\begin{aligned}
(s_{\mathrm{C}}(d)\cdot Q_0) \| U_{d0} &= r_{\mathrm{D}}(0)\cdot((s_{\mathrm{C}}(d)\cdot Q_0) \| S_1) \\
&+ (r_{\mathrm{D}}(1) + r_{\mathrm{D}}(\bot))\cdot((s_{\mathrm{C}}(d)\cdot Q_0) \| T_{d0}) \\
&+ s_{\mathrm{C}}(d)\cdot(Q_0 \| U_{d0})
\end{aligned}
$$

$$
\partial_H((s_{\mathrm{C}}(d)\cdot Q_0) \| U_{d0}) = s_{\mathrm{C}}(d)\cdot\partial_H(Q_0 \| U_{d0})
$$

Verification of the Alternating Bit Protocol

$$
\begin{aligned}
Q_1 \| U_{d0} &= r_{\mathrm{D}}(0)\cdot(Q_1 \| S_1) \\
&+ (r_{\mathrm{D}}(1) + r_{\mathrm{D}}(\bot))\cdot(Q_1 \| T_{d0}) \\
&+ (s_{\mathrm{D}}(1) + s_{\mathrm{D}}(\bot))\cdot(R_0 \| U_{d0}) \\
&+ (c_{\mathrm{D}}(1) + c_{\mathrm{D}}(\bot))\cdot(R_0 \| T_{d0})
\end{aligned}
$$

$$
\partial_H(Q_1 \| U_{d0}) = (c_{\mathrm{D}}(1) + c_{\mathrm{D}}(\bot))\cdot\partial_H(R_0 \| T_{d0})
$$

$$
\begin{aligned}
(s_{\mathrm{C}}(d)\cdot Q_0) \| U_{d0} &= r_{\mathrm{D}}(0)\cdot((s_{\mathrm{C}}(d)\cdot Q_0) \| S_1) \\
&+ (r_{\mathrm{D}}(1) + r_{\mathrm{D}}(\bot))\cdot((s_{\mathrm{C}}(d)\cdot Q_0) \| T_{d0}) \\
&+ s_{\mathrm{C}}(d)\cdot(Q_0 \| U_{d0})
\end{aligned}
$$

$$
\partial_H((s_{\mathrm{C}}(d)\cdot Q_0) \| U_{d0}) = s_{\mathrm{C}}(d)\cdot\partial_H(Q_0 \| U_{d0})
$$

Verification of the Alternating Bit Protocol

$$\begin{aligned} Q_0 \| U_{d0} &= (s_{\mathrm{D}}(0) + s_{\mathrm{D}}(\bot)) \cdot (R_1 \| U_{d0}) \\ &+ r_{\mathrm{D}}(0) \cdot (Q_0 \| S_1) + (r_{\mathrm{D}}(1) + r_{\mathrm{D}}(\bot)) \cdot (Q_0 \| T_{d0}) \\ &+ c_{\mathrm{D}}(0) \cdot (R_1 \| S_1) + c_{\mathrm{D}}(\bot) \cdot (R_1 \| T_{d0}) \end{aligned}$$

$$\partial_H(Q_0 \| U_{d0}) = c_{\mathrm{D}}(0) \cdot \partial_H(R_1 \| S_1) + c_{\mathrm{D}}(\bot) \cdot \partial_H(R_1 \| T_{d0})$$

$$\begin{aligned} R_1 \| T_{d0} &= \sum_{d' \in \Delta} \{ r_{\mathrm{B}}(d', 1) \cdot ((s_{\mathrm{C}}(d') \cdot Q_1) \| T_{d0}) + r_{\mathrm{B}}(d', 0) \cdot (Q_0 \| T_{d0}) \} \\ &+ r_{\mathrm{B}}(\bot) \cdot (Q_0 \| T_{d0}) \\ &+ (s_{\mathrm{B}}(d, 0) + s_{\mathrm{B}}(\bot)) \cdot (R_1 \| U_{d0}) \\ &+ (c_{\mathrm{B}}(d, 0) + c_{\mathrm{B}}(\bot)) \cdot (Q_0 \| U_{d0}) \end{aligned}$$

$$\partial_H(R_1 \| T_{d0}) = (c_{\mathrm{B}}(d, 0) + c_{\mathrm{B}}(\bot)) \cdot \partial_H(Q_0 \| U_{d0})$$

Verification of the Alternating Bit Protocol

$$
\begin{aligned}
Q_0 \| U_{d0} &= (s_{\mathrm{D}}(0) + s_{\mathrm{D}}(\bot)) \cdot (R_1 \| U_{d0}) \\
&+ r_{\mathrm{D}}(0) \cdot (Q_0 \| S_1) + (r_{\mathrm{D}}(1) + r_{\mathrm{D}}(\bot)) \cdot (Q_0 \| T_{d0}) \\
&+ c_{\mathrm{D}}(0) \cdot (R_1 \| S_1) + c_{\mathrm{D}}(\bot) \cdot (R_1 \| T_{d0})
\end{aligned}
$$

$$
\partial_H(Q_0 \| U_{d0}) = c_{\mathrm{D}}(0) \cdot \partial_H(R_1 \| S_1) + c_{\mathrm{D}}(\bot) \cdot \partial_H(R_1 \| T_{d0})
$$

$$
\begin{aligned}
R_1 \| T_{d0} &= \sum_{d' \in \Delta} \{ r_{\mathrm{B}}(d', 1) \cdot ((s_{\mathrm{C}}(d') \cdot Q_1) \| T_{d0}) + r_{\mathrm{B}}(d', 0) \cdot (Q_0 \| T_{d0}) \} \\
&+ r_{\mathrm{B}}(\bot) \cdot (Q_0 \| T_{d0}) \\
&+ (s_{\mathrm{B}}(d, 0) + s_{\mathrm{B}}(\bot)) \cdot (R_1 \| U_{d0}) \\
&+ (c_{\mathrm{B}}(d, 0) + c_{\mathrm{B}}(\bot)) \cdot (Q_0 \| U_{d0})
\end{aligned}
$$

$$
\partial_H(R_1 \| T_{d0}) = (c_{\mathrm{B}}(d, 0) + c_{\mathrm{B}}(\bot)) \cdot \partial_H(Q_0 \| U_{d0})
$$

Verification of the Alternating Bit Protocol

Likewise we can derive the six symmetric counter-parts of these equations (with 0's and 1's interchanged).

Concluding, the state space of $\partial_H(R_0 \| S_0)$ can be reconstructed on an equational level.

But how to get rid of the communication actions?

Verification of the Alternating Bit Protocol

Likewise we can derive the six symmetric counter-parts of these equations (with 0's and 1's interchanged).

Concluding, the state space of $\partial_H(R_0 \| S_0)$ can be reconstructed on an equational level.

But how to get rid of the communication actions?

Abstraction

τ represents an internal action:

$$\frac{}{\tau \xrightarrow{\tau} \surd}$$

The abstraction operator τ_I, with $I \subseteq A$, renames all actions from I in its argument to τ:

$$\frac{x \xrightarrow{v} \surd \quad (v \notin I)}{\tau_I(x) \xrightarrow{v} \surd} \qquad \frac{x \xrightarrow{v} x' \quad (v \notin I)}{\tau_I(x) \xrightarrow{v} \tau_I(x')}$$

$$\frac{x \xrightarrow{v} \surd \quad (v \in I)}{\tau_I(x) \xrightarrow{\tau} \surd} \qquad \frac{x \xrightarrow{v} x' \quad (v \in I)}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')}$$

Abstraction

τ represents an internal action:

$$\frac{}{\tau \xrightarrow{\tau} \surd}$$

The abstraction operator τ_I, with $I \subseteq A$, renames all actions from I in its argument to τ:

$$\frac{x \xrightarrow{v} \surd \quad (v \notin I)}{\tau_I(x) \xrightarrow{v} \surd} \qquad \frac{x \xrightarrow{v} x' \quad (v \notin I)}{\tau_I(x) \xrightarrow{v} \tau_I(x')}$$

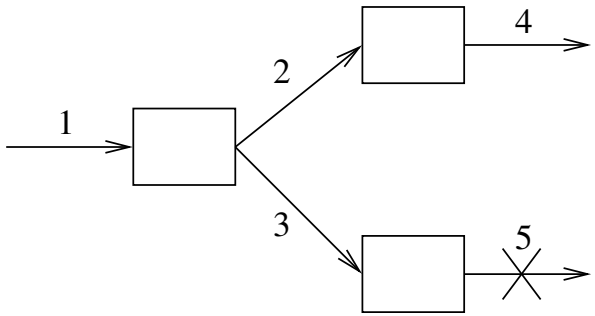
$$\frac{x \xrightarrow{v} \surd \quad (v \in I)}{\tau_I(x) \xrightarrow{\tau} \surd} \qquad \frac{x \xrightarrow{v} x' \quad (v \in I)}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')}$$

Branching Bisimulation Equivalence

Not all τ-transitions are silent.

Example: A malfunctioning channel.

A datum received via channel 1 is sent into channel 2 or 3.
A datum communicated via channel 2 it is sent into channel 4.
A datum communicated via channel 3 gets stuck, because the subsequent channel 5 is broken.



The system gets into a deadlock if a datum is transferred via channel 3. This deadlock should not disappear when abstracting away from the communication actions via channels 2 and 3.

Branching Bisimulation Equivalence

$a + \tau \cdot \delta$ and a are *not* equivalent.

$\partial_{\{b\}}(a + \tau \cdot b)$ and $\partial_{\{b\}}(a + b)$ are *not* equivalent.

$a + \tau \cdot b$ and $a + b$ are *not* equivalent.

A correct answer to the question

which τ-transitions are silent?

turns out to be

those τ-transitions that do not lose possible behaviours!

Example: $a + \tau \cdot (a + b)$ and $a + b$ are equivalent: after executing the τ in the first process term it is still possible to execute a.

Branching Bisimulation Equivalence

$a + \tau \cdot \delta$ and a are *not* equivalent.

$\partial_{\{b\}}(a + \tau \cdot b)$ and $\partial_{\{b\}}(a + b)$ are *not* equivalent.

$a + \tau \cdot b$ and $a + b$ are *not* equivalent.

A correct answer to the question

which τ-transitions are silent?

turns out to be

those τ-transitions that do not lose possible behaviours!

Example: $a + \tau \cdot (a + b)$ and $a + b$ are equivalent: after executing the τ in the first process term it is still possible to execute a.

Branching Bisimulation Equivalence

$a + \tau \cdot \delta$ and a are *not* equivalent.

$\partial_{\{b\}}(a + \tau \cdot b)$ and $\partial_{\{b\}}(a + b)$ are *not* equivalent.

$a + \tau \cdot b$ and $a + b$ are *not* equivalent.

A correct answer to the question

which τ-transitions are silent?

turns out to be

those τ-transitions that do not lose possible behaviours!

Example: $a + \tau \cdot (a + b)$ and $a + b$ are equivalent: after executing the τ in the first process term it is still possible to execute a.

Branching Bisimulation Equivalence

$a + \tau \cdot \delta$ and a are *not* equivalent.

$\partial_{\{b\}}(a + \tau \cdot b)$ and $\partial_{\{b\}}(a + b)$ are *not* equivalent.

$a + \tau \cdot b$ and $a + b$ are *not* equivalent.

A correct answer to the question

which τ-transitions are silent?

turns out to be

those τ-transitions that do not lose possible behaviours!

Example: $a + \tau \cdot (a + b)$ and $a + b$ are equivalent: after executing the τ in the first process term it is still possible to execute a.

Branching Bisimulation Equivalence

$a + \tau \cdot \delta$ and a are *not* equivalent.

$\partial_{\{b\}}(a + \tau \cdot b)$ and $\partial_{\{b\}}(a + b)$ are *not* equivalent.

$a + \tau \cdot b$ and $a + b$ are *not* equivalent.

A correct answer to the question

which τ-transitions are silent?

turns out to be

those τ-transitions that do not lose possible behaviours!

Example: $a + \tau \cdot (a + b)$ and $a + b$ are equivalent: after executing the τ in the first process term it is still possible to execute a.

Branching Bisimulation Equivalence

Let $\surd \downarrow$. A branching bisimulation is a binary relation $\mathcal{B}$ on states such that:

1. if $p \mathcal{B} q$ and $p \xrightarrow{a} p'$, then
   - either $a \equiv \tau$ and $p' \mathcal{B} q$
   - or $q \xrightarrow{\tau} \cdots \xrightarrow{\tau} q_0$ such that $p \mathcal{B} q_0$ and $q_0 \xrightarrow{a} q'$ with $p' \mathcal{B} q'$
2. if $p \mathcal{B} q$ and $q \xrightarrow{a} q'$, then
   - either $a \equiv \tau$ and $p \mathcal{B} q'$
   - or $p \xrightarrow{\tau} \cdots \xrightarrow{\tau} p_0$ such that $p_0 \mathcal{B} q$ and $p_0 \xrightarrow{a} p'$ with $p' \mathcal{B} q'$
3. if $p \mathcal{B} q$ and $p \downarrow$, then $q \xrightarrow{\tau} \cdots \xrightarrow{\tau} q_0$ with $p \mathcal{B} q_0$ and $q_0 \downarrow$
4. if $p \mathcal{B} q$ and $q \downarrow$, then $p \xrightarrow{\tau} \cdots \xrightarrow{\tau} p_0$ with $p_0 \mathcal{B} q$ and $p_0 \downarrow$

p and q are branching bisimilar, denoted $p \leftrightarrow_b q$, if there is a branching bisimulation relation $\mathcal{B}$ such that $p \mathcal{B} q$.

Branching Bisimulation Equivalence

Let $\surd \downarrow$. A branching bisimulation is a binary relation $\mathcal{B}$ on states such that:

1. if $p \mathcal{B} q$ and $p \xrightarrow{a} p'$, then
   - either $a \equiv \tau$ and $p' \mathcal{B} q$
   - or $q \xrightarrow{\tau} \cdots \xrightarrow{\tau} q_0$ such that $p \mathcal{B} q_0$ and $q_0 \xrightarrow{a} q'$ with $p' \mathcal{B} q'$
2. if $p \mathcal{B} q$ and $q \xrightarrow{a} q'$, then
   - either $a \equiv \tau$ and $p \mathcal{B} q'$
   - or $p \xrightarrow{\tau} \cdots \xrightarrow{\tau} p_0$ such that $p_0 \mathcal{B} q$ and $p_0 \xrightarrow{a} p'$ with $p' \mathcal{B} q'$
3. if $p \mathcal{B} q$ and $p \downarrow$, then $q \xrightarrow{\tau} \cdots \xrightarrow{\tau} q_0$ with $p \mathcal{B} q_0$ and $q_0 \downarrow$
4. if $p \mathcal{B} q$ and $q \downarrow$, then $p \xrightarrow{\tau} \cdots \xrightarrow{\tau} p_0$ with $p_0 \mathcal{B} q$ and $p_0 \downarrow$

p and q are branching bisimilar, denoted $p \leftrightarrow_b q$, if there is a branching bisimulation relation $\mathcal{B}$ such that $p \mathcal{B} q$.

Branching Bisimulation Equivalence

Let $\surd \downarrow$. A branching bisimulation is a binary relation $\mathcal{B}$ on states such that:

1. if $p \mathcal{B} q$ and $p \xrightarrow{a} p'$, then
   - either $a \equiv \tau$ and $p' \mathcal{B} q$
   - or $q \xrightarrow{\tau} \cdots \xrightarrow{\tau} q_0$ such that $p \mathcal{B} q_0$ and $q_0 \xrightarrow{a} q'$ with $p' \mathcal{B} q'$
2. if $p \mathcal{B} q$ and $q \xrightarrow{a} q'$, then
   - either $a \equiv \tau$ and $p \mathcal{B} q'$
   - or $p \xrightarrow{\tau} \cdots \xrightarrow{\tau} p_0$ such that $p_0 \mathcal{B} q$ and $p_0 \xrightarrow{a} p'$ with $p' \mathcal{B} q'$
3. if $p \mathcal{B} q$ and $p \downarrow$, then $q \xrightarrow{\tau} \cdots \xrightarrow{\tau} q_0$ with $p \mathcal{B} q_0$ and $q_0 \downarrow$
4. if $p \mathcal{B} q$ and $q \downarrow$, then $p \xrightarrow{\tau} \cdots \xrightarrow{\tau} p_0$ with $p_0 \mathcal{B} q$ and $p_0 \downarrow$

p and q are branching bisimilar, denoted $p \leftrightarrow_b q$, if there is a branching bisimulation relation $\mathcal{B}$ such that $p \mathcal{B} q$.

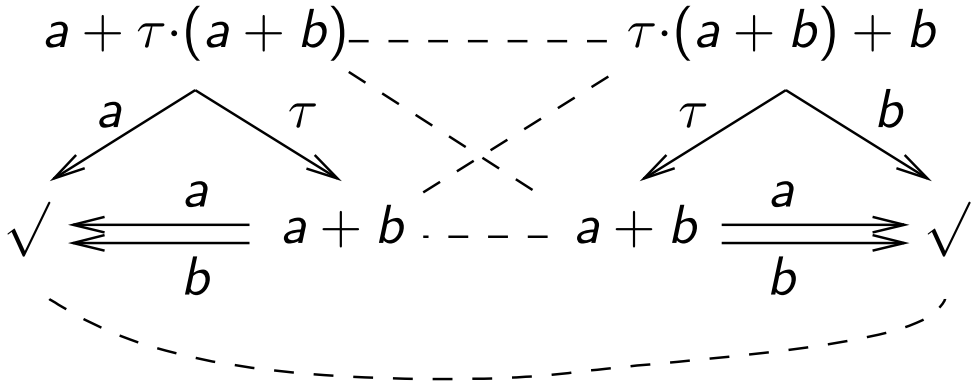
Branching Bisimulation Equivalence

Let $\surd \downarrow$. A branching bisimulation is a binary relation $\mathcal{B}$ on states such that:

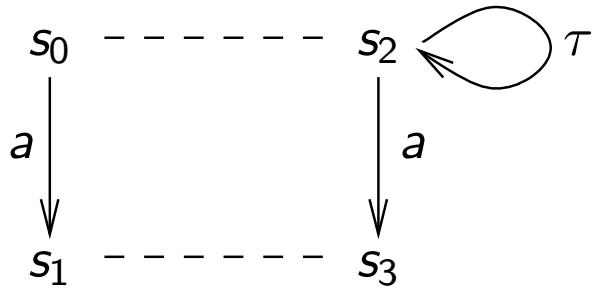
1. if $p \mathcal{B} q$ and $p \xrightarrow{a} p'$, then
   - either $a \equiv \tau$ and $p' \mathcal{B} q$
   - or $q \xrightarrow{\tau} \cdots \xrightarrow{\tau} q_0$ such that $p \mathcal{B} q_0$ and $q_0 \xrightarrow{a} q'$ with $p' \mathcal{B} q'$
2. if $p \mathcal{B} q$ and $q \xrightarrow{a} q'$, then
   - either $a \equiv \tau$ and $p \mathcal{B} q'$
   - or $p \xrightarrow{\tau} \cdots \xrightarrow{\tau} p_0$ such that $p_0 \mathcal{B} q$ and $p_0 \xrightarrow{a} p'$ with $p' \mathcal{B} q'$
3. if $p \mathcal{B} q$ and $p \downarrow$, then $q \xrightarrow{\tau} \cdots \xrightarrow{\tau} q_0$ with $p \mathcal{B} q_0$ and $q_0 \downarrow$
4. if $p \mathcal{B} q$ and $q \downarrow$, then $p \xrightarrow{\tau} \cdots \xrightarrow{\tau} p_0$ with $p_0 \mathcal{B} q$ and $p_0 \downarrow$

p and q are branching bisimilar, denoted $p \underline{\leftrightarrow}_b q$, if there is a branching bisimulation relation $\mathcal{B}$ such that $p \mathcal{B} q$.

Examples



$\underline{\leftrightarrow}_b$ satisfies a notion of fairness:



Characterizing Logic

Branching bisimilarity has an expressice characterizing logic.

Rocco De Nicola, Frits Vaandrager
Three Logics for Branching Bisimulation
JACM 42(2):458-487, 1995

Rooted Branching Bisimulation Equivalence

$a + \tau \cdot b$ and $a + b$ are *not* equivalent.

$\tau \cdot b$ and b are *not* equivalent.

A rooted branching bisimulation is a binary relation $\mathcal{B}$ on states such that:

1. if $p \mathcal{B} q$ and $p \xrightarrow{a} p'$, then $q \xrightarrow{a} q'$ with $p' \leftrightarrow_b q'$
2. if $p \mathcal{B} q$ and $q \xrightarrow{a} q'$, then $p \xrightarrow{a} p'$ with $p' \leftrightarrow_b q'$

p and q are rooted branching bisimilar, denoted $p \leftrightarrow_{rb} q$, if there is a rooted branching bisimulation relation $\mathcal{B}$ such that $p \mathcal{B} q$.

$\leftrightarrow_{rb}$ does constitute a congruence for ACP_τ.

Rooted Branching Bisimulation Equivalence

$a + \tau \cdot b$ and $a + b$ are *not* equivalent.

$\tau \cdot b$ and b are *not* equivalent.

A rooted branching bisimulation is a binary relation $\mathcal{B}$ on states such that:

1. if $p \mathcal{B} q$ and $p \xrightarrow{a} p'$, then $q \xrightarrow{a} q'$ with $p' \leftrightarrow_b q'$
2. if $p \mathcal{B} q$ and $q \xrightarrow{a} q'$, then $p \xrightarrow{a} p'$ with $p' \leftrightarrow_b q'$

p and q are rooted branching bisimilar, denoted $p \leftrightarrow_{rb} q$, if there is a rooted branching bisimulation relation $\mathcal{B}$ such that $p \mathcal{B} q$.

$\leftrightarrow_{rb}$ does constitute a congruence for ACP_τ.

Rooted Branching Bisimulation Equivalence

$a + \tau \cdot b$ and $a + b$ are *not* equivalent.

$\tau \cdot b$ and b are *not* equivalent.

A rooted branching bisimulation is a binary relation $\mathcal{B}$ on states such that:

1. if $p \mathcal{B} q$ and $p \xrightarrow{a} p'$, then $q \xrightarrow{a} q'$ with $p' \leftrightarrow_b q'$
2. if $p \mathcal{B} q$ and $q \xrightarrow{a} q'$, then $p \xrightarrow{a} p'$ with $p' \leftrightarrow_b q'$

p and q are rooted branching bisimilar, denoted $p \leftrightarrow_{rb} q$, if there is a rooted branching bisimulation relation $\mathcal{B}$ such that $p \mathcal{B} q$.

$\leftrightarrow_{rb}$ does constitute a congruence for ACP_τ.

Rooted Branching Bisimulation Equivalence

$a + \tau \cdot b$ and $a + b$ are *not* equivalent.

$\tau \cdot b$ and b are *not* equivalent.

A rooted branching bisimulation is a binary relation $\mathcal{B}$ on states such that:

1. if $p \mathcal{B} q$ and $p \xrightarrow{a} p'$, then $q \xrightarrow{a} q'$ with $p' \underline{\leftrightarrow}_b q'$
2. if $p \mathcal{B} q$ and $q \xrightarrow{a} q'$, then $p \xrightarrow{a} p'$ with $p' \underline{\leftrightarrow}_b q'$

p and q are rooted branching bisimilar, denoted $p \underline{\leftrightarrow}_{rb} q$, if there is a rooted branching bisimulation relation $\mathcal{B}$ such that $p \mathcal{B} q$.

$\underline{\leftrightarrow}_{rb}$ does constitute a congruence for ACP_τ.

Axioms for Abstraction

B1	$v \cdot \tau$	$=$	v	
B2	$v \cdot (\tau \cdot (x + y) + x)$	$=$	$v \cdot (x + y)$	
TI1	$\tau_I(v)$	$=$	v	$(v \notin I)$
TI2	$\tau_I(v)$	$=$	τ	$(v \in I)$
TI3	$\tau_I(\delta)$	$=$	δ	
TI4	$\tau_I(x + y)$	$=$	$\tau_I(x) + \tau_I(y)$	
TI5	$\tau_I(x \cdot y)$	$=$	$\tau_I(x) \cdot \tau_I(y)$	

Guarded Linear Recursion

All process terms $\tau \cdot s$ are a solution of $X = \tau \cdot X$, as $\tau \cdot s \leftrightarrows_{rb} \tau \cdot (\tau \cdot s)$.

Hence, $X = \tau \cdot X$ is unguarded.

A *linear* recursive specification E is guarded if there does not exist an infinite sequence of τ-transitions

$$\langle X|E\rangle \xrightarrow{\tau} \langle X'|E\rangle \xrightarrow{\tau} \langle X''|E\rangle \xrightarrow{\tau} \cdots$$

Guarded linear recursive specifications are exactly the ones that have a unique solution modulo $\leftrightarrows_{rb}$.

Guarded Linear Recursion

All process terms $\tau \cdot s$ are a solution of $X = \tau \cdot X$, as $\tau \cdot s \leftrightarrow_{rb} \tau \cdot (\tau \cdot s)$.

Hence, $X = \tau \cdot X$ is unguarded.

A *linear* recursive specification E is guarded if there does not exist an infinite sequence of τ-transitions

$$\langle X|E\rangle \xrightarrow{\tau} \langle X'|E\rangle \xrightarrow{\tau} \langle X''|E\rangle \xrightarrow{\tau} \cdots$$

Guarded linear recursive specifications are exactly the ones that have a unique solution modulo $\leftrightarrow_{rb}$.

Cluster Fair Abstraction Rule

Let E be a guarded linear recursive specification.

Recursion variables X and Y in E are in the same cluster for $I \subseteq A$ if $\langle X|E\rangle \xrightarrow{b_1} \cdots \xrightarrow{b_m} \langle Y|E\rangle$ and $\langle Y|E\rangle \xrightarrow{c_1} \cdots \xrightarrow{c_n} \langle X|E\rangle$ for some $b_1, \ldots, b_m, c_1, \ldots, c_n \in I \cup \{\tau\}$.

a or $a \cdot X$ is an exit for cluster C if:

1. a or $a \cdot X$ is a summand in the right-hand side of the recursive equation for a recursion variable in C, and
2. in the case of $a \cdot X$, $a \notin I \cup \{\tau\}$ or $X \notin C$.

CFAR If X is in a cluster for I with exits $\{v_1 \cdot Y_1, \ldots, v_m \cdot Y_m, w_1, \ldots, w_n\}$, then

$$v \cdot \tau_I(\langle X|E\rangle) = v \cdot \tau_I(v_1 \cdot \langle Y_1|E\rangle + \cdots + v_m \cdot \langle Y_m|E\rangle + w_1 + \cdots + w_n)$$

Cluster Fair Abstraction Rule

Let E be a guarded linear recursive specification.

Recursion variables X and Y in E are in the same cluster for $I \subseteq A$ if $\langle X|E\rangle \xrightarrow{b_1} \cdots \xrightarrow{b_m} \langle Y|E\rangle$ and $\langle Y|E\rangle \xrightarrow{c_1} \cdots \xrightarrow{c_n} \langle X|E\rangle$ for some $b_1, \ldots, b_m, c_1, \ldots, c_n \in I \cup \{\tau\}$.

a or $a \cdot X$ is an exit for cluster C if:

1. a or $a \cdot X$ is a summand in the right-hand side of the recursive equation for a recursion variable in C, and
2. in the case of $a \cdot X$, $a \notin I \cup \{\tau\}$ or $X \notin C$.

CFAR If X is in a cluster for I with exits $\{v_1 \cdot Y_1, \ldots, v_m \cdot Y_m, w_1, \ldots, w_n\}$, then

$$v \cdot \tau_I(\langle X|E\rangle) = v \cdot \tau_I(v_1 \cdot \langle Y_1|E\rangle + \cdots + v_m \cdot \langle Y_m|E\rangle + w_1 + \cdots + w_n)$$

Cluster Fair Abstraction Rule

Let E be a guarded linear recursive specification.

Recursion variables X and Y in E are in the same cluster for $I \subseteq A$ if $\langle X|E\rangle \xrightarrow{b_1} \cdots \xrightarrow{b_m} \langle Y|E\rangle$ and $\langle Y|E\rangle \xrightarrow{c_1} \cdots \xrightarrow{c_n} \langle X|E\rangle$ for some $b_1, \ldots, b_m, c_1, \ldots, c_n \in I \cup \{\tau\}$.

a or $a{\cdot}X$ is an exit for cluster C if:

1. a or $a{\cdot}X$ is a summand in the right-hand side of the recursive equation for a recursion variable in C, and
2. in the case of $a{\cdot}X$, $a \notin I \cup \{\tau\}$ or $X \notin C$.

CFAR If X is in a cluster for I with exits $\{v_1{\cdot}Y_1, \ldots, v_m{\cdot}Y_m, w_1, \ldots, w_n\}$, then

$$v \cdot \tau_I(\langle X|E\rangle) = v \cdot \tau_I(v_1{\cdot}\langle Y_1|E\rangle + \cdots + v_m{\cdot}\langle Y_m|E\rangle + w_1 + \cdots + w_n)$$

Example

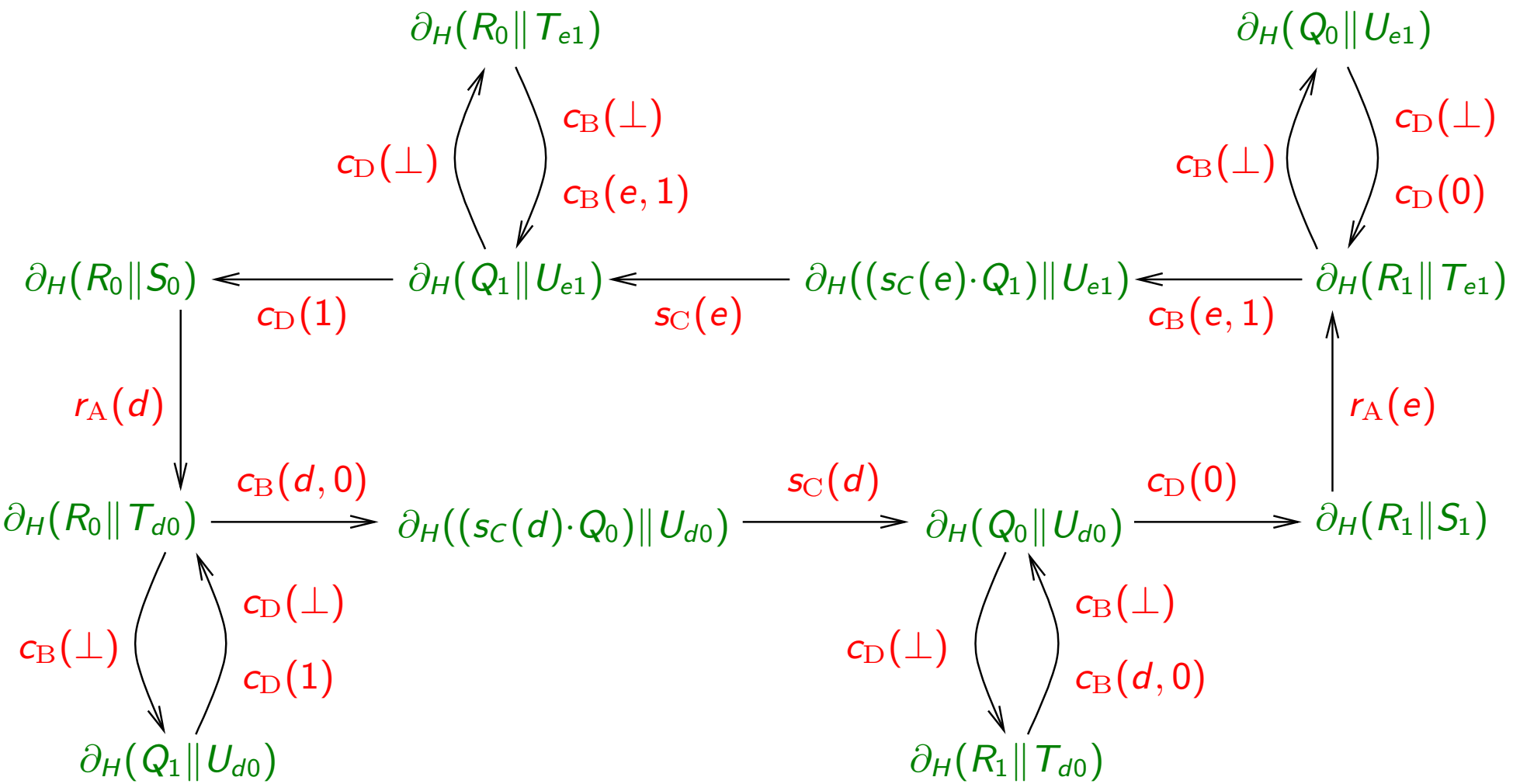
$$a \cdot \tau_{\{c\}}(\langle X | X = c \cdot Y + b_1, Y = c \cdot X + b_2 \rangle) \;=\; a \cdot (b_1 + b_2)$$

Soundness and Ground-Completeness

The axioms for ACP_τ with guarded linear recursion are sound and ground-complete modulo $\underline{\leftrightarrow}_{rb}$:

$$s \underline{\leftrightarrow}_{rb} t \;\Leftrightarrow\; s = t$$

Back to the Alternating Bit Protocol



Verification of the Alternating Bit Protocol

Let I be the set of communication actions over channels B and D.

After application of τ_I to $\partial_H(R_0 \| S_0)$, the loops of communication actions can be removed by CFAR.

For example, $\partial_H(R_0 \| T_{d0})$ and $\partial_H(Q_1 \| U_{d0})$ form a cluster for I with exit $c_{\mathrm{B}}(d,0) \cdot \partial_H((s_C(d) \cdot Q_0) \| U_{d0})$, so

$$\begin{aligned} r_{\mathrm{A}}(d) \cdot \tau_I(\partial_H(R_0 \| T_{d0})) &= r_{\mathrm{A}}(d) \cdot (\tau_I(c_{\mathrm{B}}(d,0) \cdot \partial_H((s_C(d) \cdot Q_0) \| U_{d0}))) \\ &= r_{\mathrm{A}}(d) \cdot (\tau \cdot \tau_I(\partial_H((s_C(d) \cdot Q_0) \| U_{d0}))) \\ &= r_{\mathrm{A}}(d) \cdot \tau_I(\partial_H((s_C(d) \cdot Q_0) \| U_{d0})) \end{aligned}$$

Verification of the Alternating Bit Protocol

Let I be the set of communication actions over channels B and D.

After application of τ_I to $\partial_H(R_0 \| S_0)$, the loops of communication actions can be removed by CFAR.

For example, $\partial_H(R_0 \| T_{d0})$ and $\partial_H(Q_1 \| U_{d0})$ form a cluster for I with exit $c_{\mathrm{B}}(d,0) \cdot \partial_H((s_C(d) \cdot Q_0) \| U_{d0})$, so

$$\begin{aligned} r_{\mathrm{A}}(d) \cdot \tau_I(\partial_H(R_0 \| T_{d0})) &= r_{\mathrm{A}}(d) \cdot (\tau_I(c_{\mathrm{B}}(d,0) \cdot \partial_H((s_C(d) \cdot Q_0) \| U_{d0}))) \\ &= r_{\mathrm{A}}(d) \cdot (\tau \cdot \tau_I(\partial_H((s_C(d) \cdot Q_0) \| U_{d0}))) \\ &= r_{\mathrm{A}}(d) \cdot \tau_I(\partial_H((s_C(d) \cdot Q_0) \| U_{d0})) \end{aligned}$$

Verification of the Alternating Bit Protocol

Likewise we derive

$$
\begin{aligned}
s_{\mathrm{C}}(d)\cdot\tau_I(\partial_H(Q_0\|U_{d0})) &= s_{\mathrm{C}}(d)\cdot\tau_I(\partial_H(R_1\|S_1))\\
r_{\mathrm{A}}(e)\cdot\tau_I(\partial_H(R_1\|T_{e1})) &= r_{\mathrm{A}}(e)\cdot\tau_I(\partial_H((s_C(e)\cdot Q_1)\|U_{e1}))\\
s_{\mathrm{C}}(e)\cdot\tau_I(\partial_H(Q_1\|U_{e1})) &= s_{\mathrm{C}}(e)\cdot\tau_I(\partial_H(R_0\|S_0))
\end{aligned}
$$

Verification of the Alternating Bit Protocol

$$\begin{aligned}
\tau_I(\partial_H(R_0 \| S_0)) &= \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot \tau_I(\partial_H(R_0 \| T_{d0})) \\
&= \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot \tau_I(\partial_H((s_C(d) \cdot Q_0) \| U_{d0})) \\
&= \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot (s_{\mathrm{C}}(d) \cdot \tau_I(\partial_H(Q_0 \| U_{d0}))) \\
&= \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot (s_{\mathrm{C}}(d) \cdot \tau_I(\partial_H(R_1 \| S_1)))
\end{aligned}$$

Likewise we derive

$$\tau_I(\partial_H(R_1 \| S_1)) = \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot (s_{\mathrm{C}}(d) \cdot \tau_I(\partial_H(R_0 \| S_0)))$$

Concluding, using RSP it follows that

$$\tau_I(\partial_H(R_0 \| S_0)) = \sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot (s_{\mathrm{C}}(d) \cdot \tau_I(\partial_H(R_0 \| S_0)))$$

Verification of the Alternating Bit Protocol

$$\begin{aligned}
\tau_I(\partial_H(R_0 \| S_0)) &= \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot \tau_I(\partial_H(R_0 \| T_{d0})) \\
&= \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot \tau_I(\partial_H((s_C(d) \cdot Q_0) \| U_{d0})) \\
&= \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot (s_{\mathrm{C}}(d) \cdot \tau_I(\partial_H(Q_0 \| U_{d0}))) \\
&= \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot (s_{\mathrm{C}}(d) \cdot \tau_I(\partial_H(R_1 \| S_1)))
\end{aligned}$$

Likewise we derive

$$\tau_I(\partial_H(R_1 \| S_1)) = \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot (s_{\mathrm{C}}(d) \cdot \tau_I(\partial_H(R_0 \| S_0)))$$

Concluding, using RSP it follows that

$$\tau_I(\partial_H(R_0 \| S_0)) = \sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot (s_{\mathrm{C}}(d) \cdot \tau_I(\partial_H(R_0 \| S_0)))$$

Verification of the Alternating Bit Protocol

$$
\begin{aligned}
\tau_I(\partial_H(R_0 \| S_0)) &= \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot \tau_I(\partial_H(R_0 \| T_{d0})) \\
&= \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot \tau_I(\partial_H((s_C(d) \cdot Q_0) \| U_{d0})) \\
&= \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot (s_{\mathrm{C}}(d) \cdot \tau_I(\partial_H(Q_0 \| U_{d0}))) \\
&= \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot (s_{\mathrm{C}}(d) \cdot \tau_I(\partial_H(R_1 \| S_1)))
\end{aligned}
$$

Likewise we derive

$$
\tau_I(\partial_H(R_1 \| S_1)) = \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot (s_{\mathrm{C}}(d) \cdot \tau_I(\partial_H(R_0 \| S_0)))
$$

Concluding, using RSP it follows that

$$
\tau_I(\partial_H(R_0 \| S_0)) = \sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot (s_{\mathrm{C}}(d) \cdot \tau_I(\partial_H(R_0 \| S_0)))
$$

Summary

Concurrent systems are generally specified in the form

$$\tau_I(\partial_H(X_1 \| \cdots \| X_k))$$

where H contains all send and receive actions over internal channels, and I all communication actions over these channels.

The recursion variables $X_1, \ldots, X_k$ are in turn specified using:

- actions, alternative and sequential composition
- recursive equations

The underlying network topology is captured by means of the communication function γ.

Abstract Data Types

Concurrent systems often require intricate data structures.

Data can also be specified algebraically, as abstract data types.

Example: Natural numbers with successor *succ* and addition *plus*:

$$\begin{aligned} plus(n, 0) &= n \\ plus(n, succ(m)) &= succ(plus(n, m)) \end{aligned}$$

μCRL

The specification language μCRL is based on process algebra and abstract data types.

Its toolset supports model checking as well as theorem proving.

It has been used for verifications of numerous real-life protocols and distributed algorithms.