

Composition

Process Refinement

November 2006

Contents

- Concurrency
- Multi-way synchronisation
- Rewriting
- Concurrency and choice

Concurrency

If P and Q are processes, and αP and αQ are sets of events, then we may write

$$P [\alpha P \parallel \alpha Q] Q$$

to denote the parallel composition of P and Q , in which P and Q are intended to participate in every event from sets αP and αQ , respectively.

In this composition, we call αP the alphabet of P .

Sharing events

In the parallel composition

$$P [\alpha P \parallel \alpha Q] Q$$

process Q forms part of P 's environment, sharing in every event from set αQ .

Q cannot perform any event in αP unless P performs the same event at the same time.

Processes P and Q are both involved in every event from the intersection of alphabets $\alpha P \cap \alpha Q$.

Step law: ||

$$P = \square e : A \bullet e \rightarrow P(e)$$

$$Q = \square e : B \bullet e \rightarrow Q(e)$$

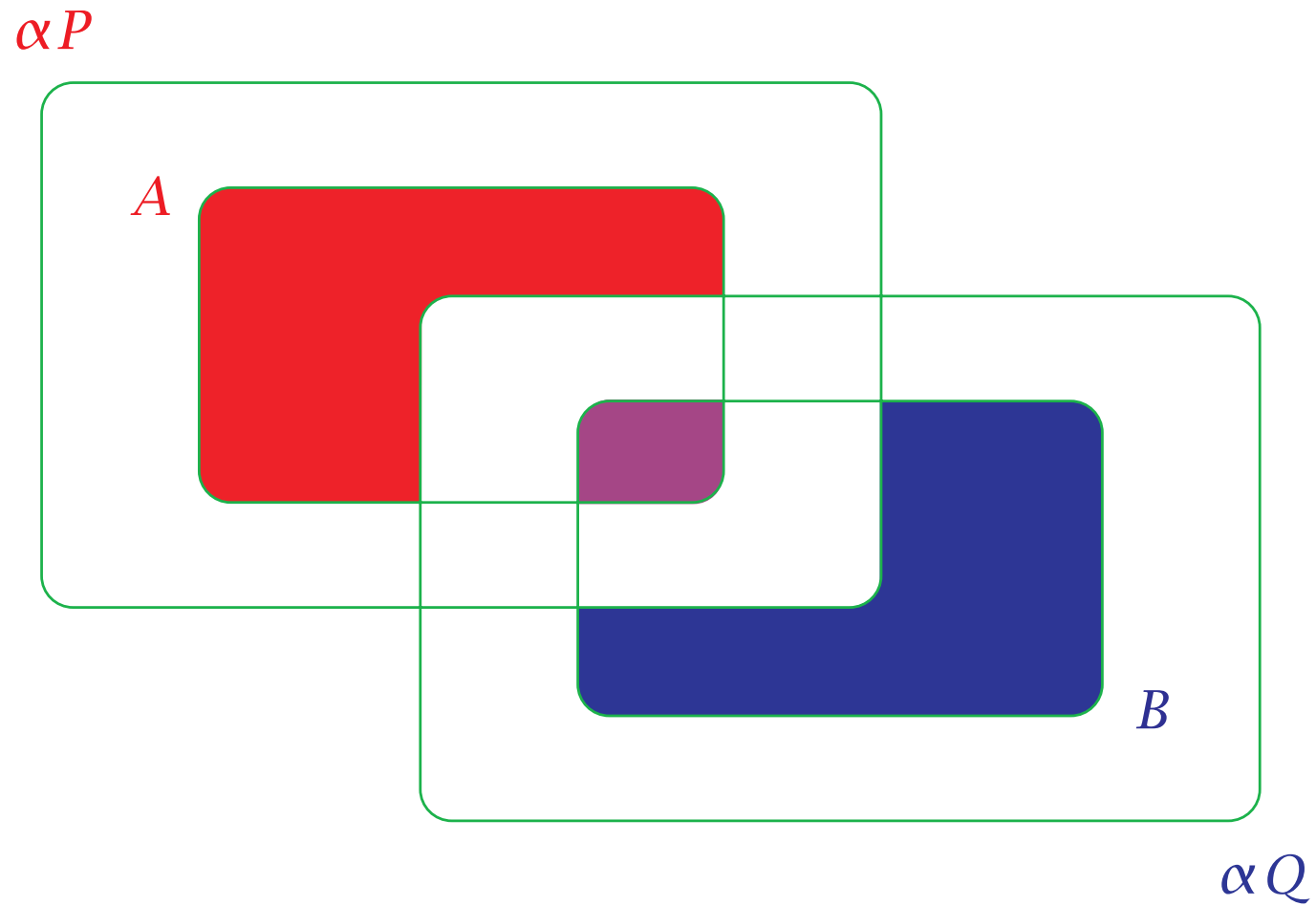
$$P [\alpha P \parallel \alpha Q] Q = \square e : (A \setminus \alpha Q) \bullet e \rightarrow (P(e) [\alpha P \parallel \alpha Q] Q)$$

□

$$\square e : (B \setminus \alpha P) \bullet e \rightarrow (P [\alpha P \parallel \alpha Q] Q(e))$$

□

$$\square e : (A \cap B) \bullet e \rightarrow (P(e) [\alpha P \parallel \alpha Q] Q(e))$$



The \parallel step law

Example

$$\begin{array}{l}
 a \rightarrow P \\
 [\{a, c, d\} \parallel \{b, c, d\}] \\
 b \rightarrow Q
 \end{array}
 =
 \begin{array}{l}
 a \rightarrow (P \\
 [\{a, c, d\} \parallel \{b, c, d\}] \\
 b \rightarrow Q)
 \end{array}$$

□

$$\begin{array}{l}
 b \rightarrow (a \rightarrow P \\
 [\{a, c, d\} \parallel \{b, c, d\}] \\
 Q)
 \end{array}$$

Independent events are shared only with the environment: they may be performed without reference to the other process.

Example

$$\begin{array}{l} a \rightarrow P \\ [\{a, c, d\} \parallel \{b, c, d\}] \\ c \rightarrow Q \end{array} = \begin{array}{l} a \rightarrow (P \\ [\{a, c, d\} \parallel \{b, c, d\}] \\ c \rightarrow Q) \end{array}$$

An event that is common to both alphabets cannot be performed without the cooperation of the other process.

Example

$$\begin{array}{l} c \rightarrow P \\ [\{a, c, d\} \parallel \{b, c, d\}] \\ c \rightarrow Q \end{array} = c \rightarrow (P \parallel Q)$$

When two processes in a parallel combination share in the same event, we see just one copy of that event occurring.

Example

$$\begin{array}{l} c \rightarrow P \\ [\{a, c, d\} \parallel \{b, c, d\}] \\ d \rightarrow Q \end{array} = \text{Stop}$$

If each process is insisting on a different common event, and neither can make progress independently, we have **deadlock**.

Example: boxes

Two people are loading boxes onto a truck. There is a red box, a blue box, and a green box.

Matthew is to load the red box, while Marina loads the blue box. The green box is particularly heavy, and both of them will be needed to lift it.

Matthew is happy to load the *red* and *green* boxes in either order:

Matthew = red → green → Stop

□

green → red → Stop

Marina insists that the *green* box is loaded before the *blue*:

Marina = green → blue → Stop

Matthew

[{*red, green*} || {*blue, green*}]

Marina

$$\begin{aligned} & \dots = \begin{array}{l} \textit{red} \rightarrow \textit{green} \rightarrow \textit{Stop} \\ \square \\ \textit{green} \rightarrow \textit{red} \rightarrow \textit{Stop} \end{array} \\ & \quad [\{\textit{red}, \textit{green}\} \parallel \{\textit{blue}, \textit{green}\}] \\ & \quad \textit{green} \rightarrow \textit{blue} \rightarrow \textit{Stop} \end{aligned}$$

$$\begin{aligned}
 & \text{red} \rightarrow (\text{green} \rightarrow \text{Stop} \\
 & \quad [\{ \text{red}, \text{green} \} \parallel \{ \text{blue}, \text{green} \}] \\
 & \quad \text{green} \rightarrow \text{blue} \rightarrow \text{Stop})
 \end{aligned}$$

$$\dots = \square$$

$$\begin{aligned}
 & \text{green} \rightarrow (\text{red} \rightarrow \text{Stop} \\
 & \quad [\{ \text{red}, \text{green} \} \parallel \{ \text{blue}, \text{green} \}] \\
 & \quad \text{blue} \rightarrow \text{Stop})
 \end{aligned}$$

$$\begin{aligned}
 &red \rightarrow green \rightarrow (Stop \\
 &\quad [\{red, green\} \parallel \{blue, green\}] \\
 &\quad blue \rightarrow Stop)
 \end{aligned}$$

□

$$\begin{aligned}
 \dots &= green \rightarrow (red \rightarrow (Stop \\
 &\quad [\{red, green\} \parallel \{blue, green\}] \\
 &\quad blue \rightarrow Stop)
 \end{aligned}$$

□

$$\begin{aligned}
 &blue \rightarrow (red \rightarrow Stop \\
 &\quad [\{red, green\} \parallel \{blue, green\}] \\
 &\quad Stop))
 \end{aligned}$$

red → *green* → *blue* → *Stop*

□

...

=

green → (*red* → *blue* → *Stop*

□

blue → *red* → *Stop*)

Process alphabets

If a process is assigned the same alphabet wherever it appears, then we need define αP only once. If αP and αQ have been defined, then we may write $P \parallel Q$ in place of $P [\alpha P \parallel \alpha Q] Q$.

Example: boxes

If

$$\alpha_{\text{Matthew}} = \{\text{red}, \text{green}\}$$

$$\alpha_{\text{Marina}} = \{\text{blue}, \text{green}\}$$

then

$$\text{Matthew} \parallel \text{Marina} = \text{Matthew} [\{\text{red}, \text{green}\} \parallel \{\text{blue}, \text{green}\}] \text{Marina}$$

Constraint

In a parallel composition, a trace is possible only if each process would agree upon the order in which events appear.

$$\begin{aligned} \text{traces} [P \parallel Q] = \\ \{ tr : \text{seq Event} \mid tr = tr \upharpoonright (\alpha P \cup \alpha Q) \wedge \\ tr \upharpoonright \alpha P \in \text{traces} [P] \wedge tr \upharpoonright \alpha Q \in \text{traces} [Q] \} \end{aligned}$$

where

$$\langle \rangle \upharpoonright A = \langle \rangle$$

$$(\langle a \rangle \frown tr) \upharpoonright A = \text{if } a \in A \text{ then } \langle a \rangle \frown (tr \upharpoonright A) \text{ else } tr \upharpoonright A$$

Example: boxes

$traces \llbracket Matthew \rrbracket = \{\langle \rangle, \langle red \rangle, \langle green \rangle, \langle red, green \rangle, \langle green, red \rangle\}$

$traces \llbracket Marina \rrbracket = \{\langle \rangle, \langle green \rangle, \langle green, blue \rangle\}$

$\langle red, green, blue \rangle \notin traces \llbracket Matthew \parallel Marina \rrbracket$

Law

$$\textit{Skip} \parallel \textit{Skip} = \textit{Skip}$$

A parallel composition can terminate only when each of the processes involved is ready to do so.

Example: parallel breakfast

Juice = glass → juice → Skip
Cereal = bowl → cereal → milk → Skip
Coffee = cup → coffee → sugar → Skip

ParallelBreakfast = Juice || Coffee || Cereal

traces of parallel breakfast...

Multi-way synchronisation

The abstract events in CSP correspond to transactions, rather than point-to-point communications: there is no limit to the number of processes that may be involved.

An event that is shared between two processes P and Q may also be shared with a third process R : in the combination $P \parallel Q \parallel R$, the intersection $\alpha P \cap \alpha Q \cap \alpha R$ may be non-empty.

Example: boxes

$\alpha_{Alan} = \{red, blue, green\}$

$Alan = green \rightarrow red \rightarrow blue \rightarrow Stop$

$$\begin{aligned} \text{Matthew} &= (\text{Matthew} \\ &\parallel [\{\text{red, green}\} \parallel \{\text{blue, green}\}] \\ \text{Marina} &= \text{Marina}) \\ &\parallel [\{\text{red, green}\} \cup \{\text{blue, green}\} \parallel \{\text{red, blue, green}\}] \\ \text{Alan} &= \text{Alan} \end{aligned}$$

(*red* → *green* → *blue* → *Stop*

□

green → (*red* → *blue* → *Stop*

... =

□

blue → *red* → *Stop*))

[{ *red*, *blue*, *green* } || { *red*, *blue*, *green* }]

green → *red* → *blue* → *Stop*

$$\begin{aligned}
 & \dots = \text{green} \rightarrow \\
 & \quad ((\text{red} \rightarrow \text{blue} \rightarrow \text{Stop} \\
 & \quad \quad \square \\
 & \quad \text{blue} \rightarrow \text{red} \rightarrow \text{Stop}) \\
 & \quad [\{ \text{red}, \text{blue}, \text{green} \} \parallel \{ \text{red}, \text{blue}, \text{green} \}] \\
 & \quad \text{red} \rightarrow \text{blue} \rightarrow \text{Stop})
 \end{aligned}$$

$$\dots = \begin{array}{l} \textit{green} \rightarrow \textit{red} \rightarrow \\ \textit{blue} \rightarrow \textit{Stop} \\ [\{\textit{red}, \textit{blue}, \textit{green}\} \parallel \{\textit{red}, \textit{blue}, \textit{green}\}] \\ \textit{blue} \rightarrow \textit{Stop} \end{array}$$

= *green* → *red* → *blue* → *Stop*

Assigning alphabets

To ensure that the parallel operator is associative, we must be consistent in the way that we assign alphabets to processes. If S is defined by

$$S = P \parallel Q$$

and we have assigned alphabets to P , Q , and S , then we must take care that

$$\alpha S = \alpha P \cup \alpha Q$$

Example: boxes

The parallel combination $Matthew \parallel Marina \parallel Alan$ could also be written as

$$(Matthew \parallel Marina) [\alpha Matthew \cup \alpha Marina \parallel \alpha Alan] Alan$$

Alphabets are static

We must be consistent also in the way that we assign alphabets to processes constructed using the choice and sequencing operators. If S is defined by any of the following equations,

$$S = a \rightarrow P \qquad S = P \sqcap Q$$

$$S = P \sqcup Q \qquad S = P \circledast Q$$

and alphabets are assigned to S and P , then αS must be equal to αP (and similarly for Q).

Aside: α

α is not an operator.

To treat it as one, we would need to annotate our basic processes—and our recursive definitions—with a set expression.

For example, writing $Stop(A)$ rather than simply $Stop$.

Law

$$P \parallel (Q \sqcap R) = (P \parallel Q) \sqcap (P \parallel R)$$

$$(P \sqcap Q) \parallel R = (P \parallel R) \sqcap (Q \parallel R)$$

Rewriting

We may use the distributive and step laws to rewrite a parallel composition of processes into sequential form.

Before we do this, we may wish to give names to various stages of each of the component processes.

Example: vending machines

$$\mathit{Coins} = \mathit{coin} \rightarrow (\mathit{approve} \rightarrow \mathit{Coins} \sqcap \mathit{return} \rightarrow \mathit{Coins})$$

can be written as

$$\mathit{Coins} = \mathit{coin} \rightarrow \mathit{Approve} \sqcap \mathit{Return}$$

$$\mathit{Approve} = \mathit{approve} \rightarrow \mathit{Coins}$$

$$\mathit{Return} = \mathit{return} \rightarrow \mathit{Coins}$$

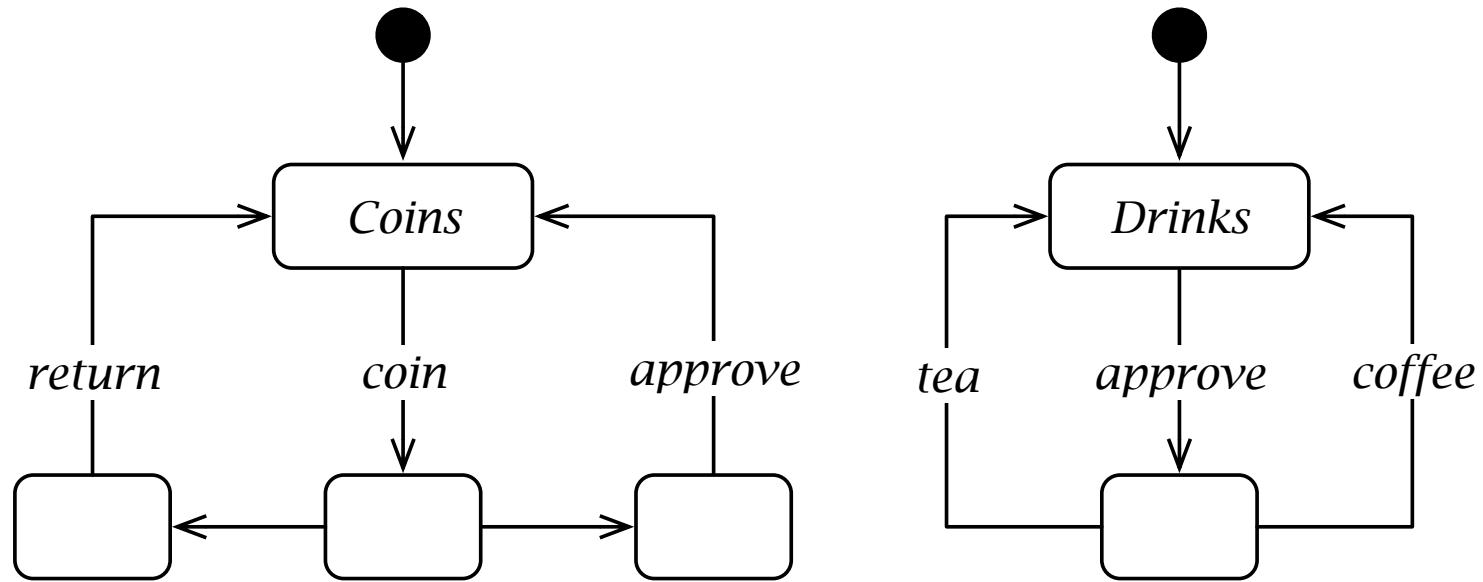
$$\textit{Drinks} = \textit{approve} \rightarrow (\textit{tea} \rightarrow \textit{Drinks} \sqcap \textit{coffee} \rightarrow \textit{Drinks})$$

can be written as

$$\textit{Drinks} = \textit{approve} \rightarrow \textit{Ready}$$

$$\textit{Ready} = \textit{tea} \rightarrow \textit{Drinks} \sqcap$$

$$\textit{coffee} \rightarrow \textit{Drinks}$$



Vending machine components

Example: Vending machines

$$CVM = Coins \parallel Drinks$$

We can use the step law for \parallel , together with an application of the \parallel -distributive law, to rewrite this process in sequential form.

Coins || Drinks =

coin → ((*Approve || Drinks*) □ (*Return || Drinks*))

Approve || Drinks =

approve → (*Coins || Ready*)

Return || Drinks =

return → (*Coins || Drinks*)

Coins || Ready =

coin → ((*Approve || Ready*) □ (*Return || Ready*)) □

tea → (*Coins || Drinks*) □

coffee → (*Coins || Drinks*)

Approve || Ready =

tea → (*Approve || Drinks*) □

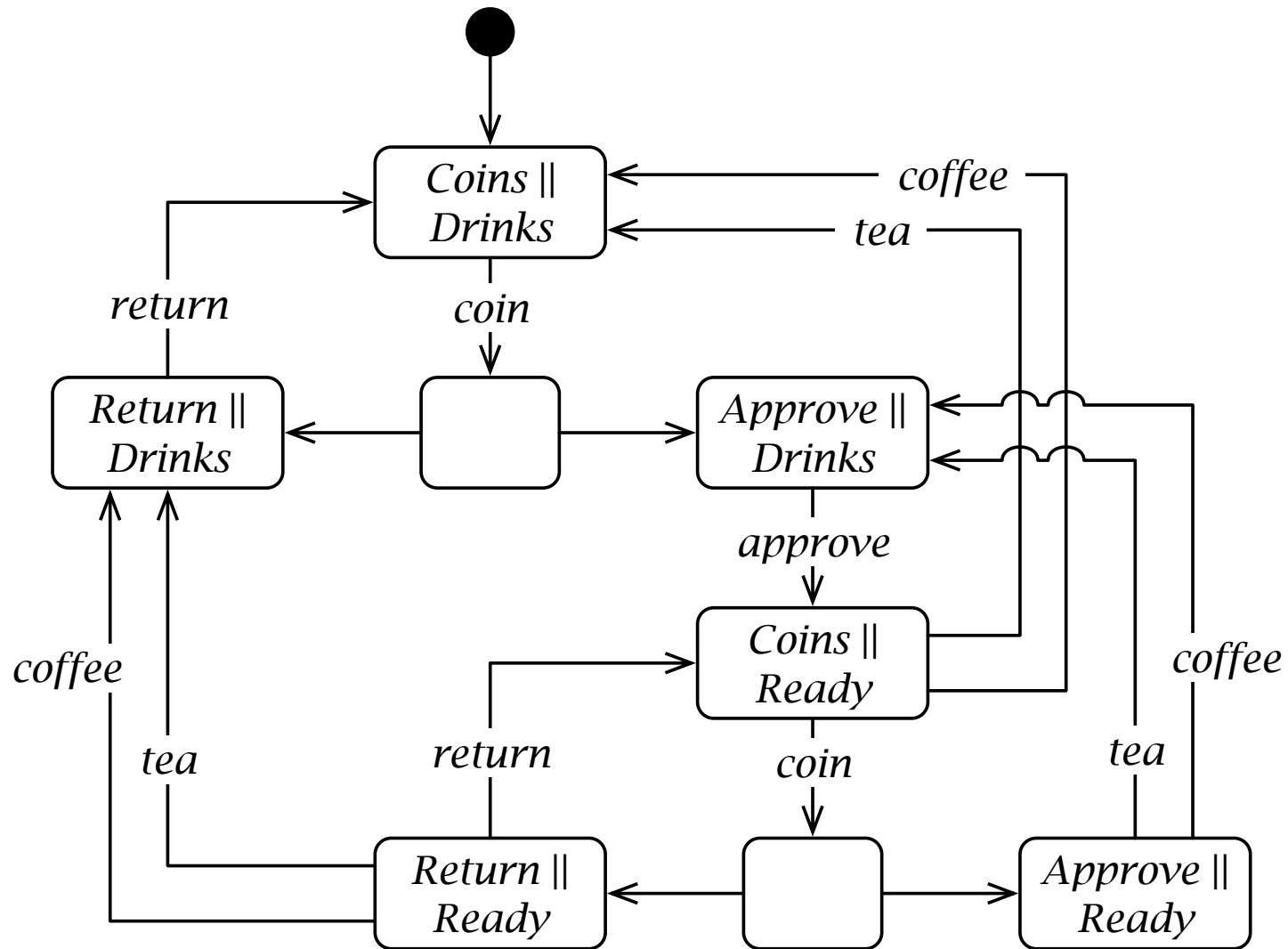
coffee → (*Approve || Drinks*)

Return || *Ready* =

tea → (*Return* || *Drinks*) □

coffee → (*Return* || *Drinks*) □

return → (*Coins* || *Ready*)



A sequential view of the parallel combination

Example: vending machines

$AC = \textit{coin} \rightarrow \textit{coffee} \rightarrow \textit{Skip}$

□

$\textit{tea} \rightarrow \textit{Skip}$

$$\begin{aligned} TOVM = & \textit{coin} \rightarrow \\ & (\textit{coffee} \rightarrow \textit{Skip} \\ & \quad \square \\ & \textit{tea} \rightarrow \textit{Skip}) \end{aligned}$$

$$AC \parallel TOVM$$

$$\dots = AC \parallel (\textit{coffee} \rightarrow \textit{Skip} \sqcap \textit{tea} \rightarrow \textit{Skip})$$

$$\dots = (AC \parallel \textit{coffee} \rightarrow \textit{Skip}) \sqcap (AC \parallel \textit{tea} \rightarrow \textit{Skip})$$

$\dots = \textit{coin} \rightarrow (\textit{coffee} \rightarrow \textit{Skip}$

\sqcap

$\textit{Stop})$

Concurrency and choice

If a choice is internal to a particular component, it will be internal to any parallel combination that the component is placed in.

- a parallel combination of two external choices presents an external choice to the environment;
- a parallel combination of external and internal choices behaves as an internal choice;
- a parallel combination of two internal choices behaves as an internal choice that may deadlock.

Example: Agreement

$\alpha_{\text{Helpful}} = \{\text{stayIn}, \text{goOut}\}$

$\text{Helpful} = \text{stayIn} \rightarrow \text{Skip} \sqcap \text{goOut} \rightarrow \text{Skip}$

$\alpha_{\text{Awkward}} = \{\text{stayIn}, \text{goOut}\}$

$\text{Awkward} = \text{stayIn} \rightarrow \text{Skip} \sqcap \text{goOut} \rightarrow \text{Skip}$

Helpful || *Helpful* = *Helpful*

Helpful || Awkward = Awkward

$$\textit{Awkward} \parallel \textit{Awkward} = \textit{Awkward} \sqcap \textit{Stop}$$

Exercise: customer and machines

$$ND = coin \rightarrow (coffee \rightarrow ND$$
$$\sqcap$$
$$tea \rightarrow ND)$$

Which of the following machines is safe for the normal drinker ND to use?

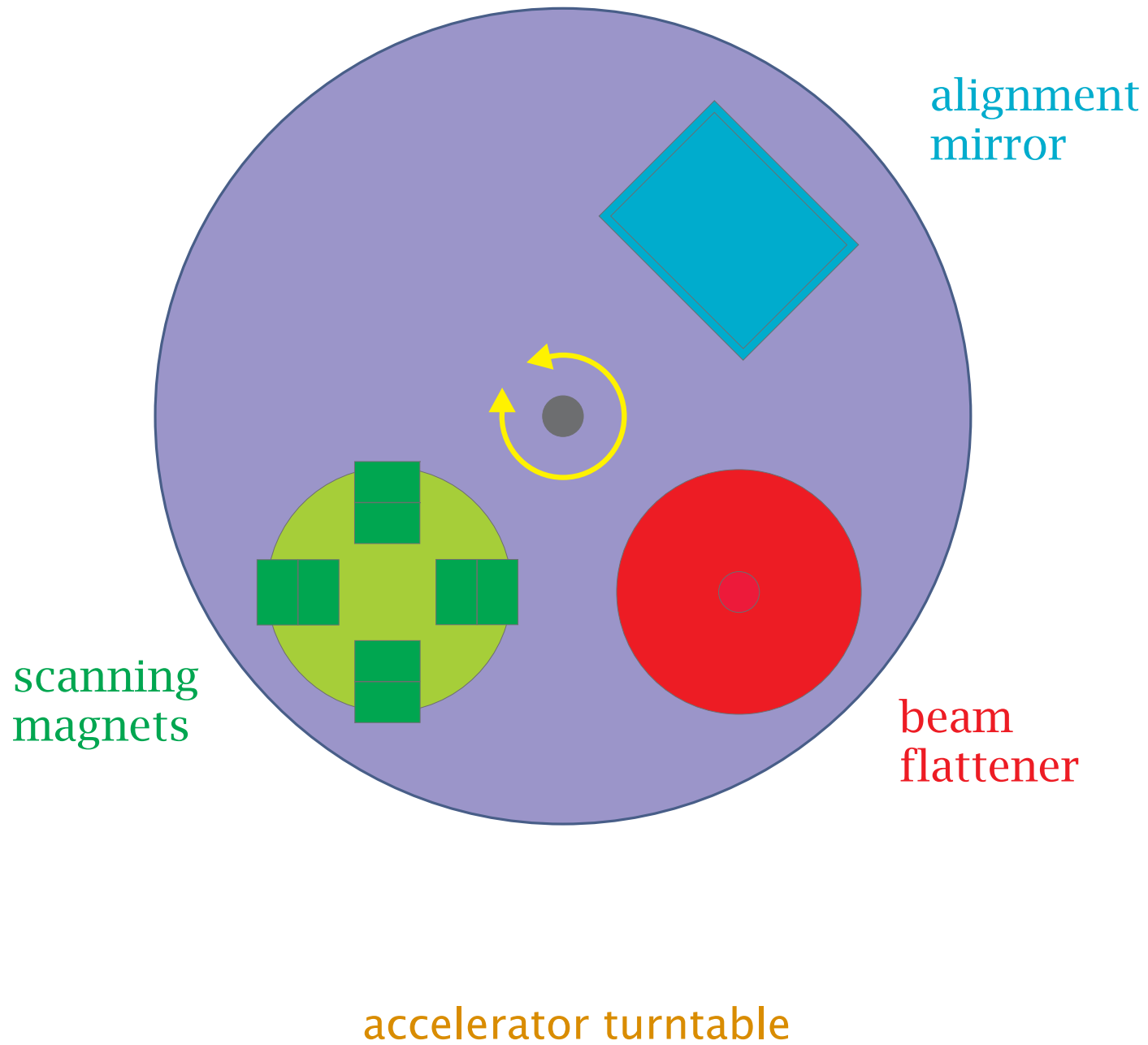
1. CM : the coffee machine
2. VM : the vending machine
3. NVM : the nondeterministic vending machine

Solution: customer and machines

1. *CM*: no
2. *VM*: yes
3. *NVM*: no

Exercise: Medical accelerator

- computer-controlled machine used in radiation therapy
- two types of beam can be created:
 - low beam (low current, directed by scanning magnets)
 - high beam (high current, diffused by beam flattener)



Events

External:

- *high* — select high beam
- *low* — select low beam
- *treat* — fire beam
- *reset* — abort treatment

Internal:

- *hb* — beam current goes high
- *lb* — beam current goes low
- *hs* — flattener (or shield) moves into the high position
- *ls* — flattener (or shield) moves into the low position

Question

Triggering the high (photon) beam without the beam flattener may cause severe bone and tissue damage to the patient.

What kind of traces must we avoid?

Answer

Any trace that has a *treat* event for which the most recent beam event was *hb* and the most recent shield event was *ls*.

For example,

$\langle \dots, hb, ls, treat \rangle$

Process description

$$\text{System} = (\text{SetLow} \wp \text{Ready})$$

$$\text{Ready} = (\text{high} \rightarrow \text{High}) \sqcap (\text{low} \rightarrow \text{Low})$$

$$\text{High} = (\text{HighTreatment} \triangle \text{reset} \rightarrow \text{Skip}) \wp \text{Ready}$$

$$\text{HighTreatment} = \text{SetHigh} \wp \text{treat} \rightarrow \text{SetLow}$$

$$\text{SetHigh} = (\text{hs} \rightarrow \text{Skip} \parallel \text{hb} \rightarrow \text{Skip})$$

$$\text{SetLow} = (\text{ls} \rightarrow \text{Skip} \parallel \text{lb} \rightarrow \text{Skip})$$

$$\text{Low} = (\text{LowTreatment} \triangle \text{reset} \rightarrow \text{Skip}) \wp \text{Ready}$$

$$\text{LowTreatment} = \text{treat} \rightarrow \text{Skip}$$

Question

Is this a safe design?

Answer

No. The system can engage in the following sequence of events:

$\langle ls, lb, high, hb, reset, low, treat \rangle$

Question

How could we fix it?

Answer

We could sequentialise the two setup processes:

$$\textit{SetHigh} = \textit{hs} \rightarrow \textit{hb} \rightarrow \textit{Skip}$$

$$\textit{SetLow} = \textit{lb} \rightarrow \textit{ls} \rightarrow \textit{Skip}$$

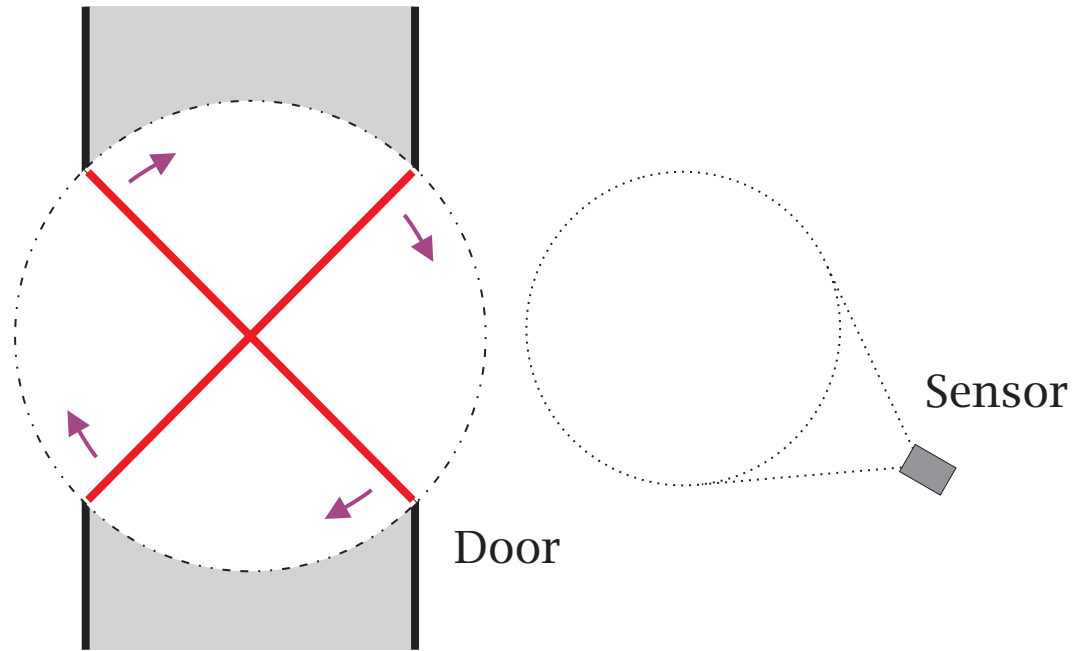
We could also use *SetLow* to guard the *LowTreatment* process.

Exercise: revolving door

A revolving door system consists of the door itself, which is divided into four quadrants, and a sensor. The sensor is not placed to detect the presence of a person within the circumference of the door; instead, it is placed to detect the approach of a potential door user.

We may construct a model using the following events:

- *start*: the door is told to start rotating;
- *rotate*: the door rotates one quarter-turn;
- *enter*: a person enters the door;
- *leave*: a person leaves the door;
- *detect*: the sensor detects a person.



A revolving door system

System = Sensor || Door || Person

Sensor = detect → start → Sensor

Door = start → (Rotate △ Door)

Rotate =

rotate → rotate →

rotate → rotate → Stop

Person = PersonOutside

PersonOutside =

detect → *PersonDetected*

□

rotate → *PersonOutside*

PersonDetected =

enter → *PersonInside*

□

rotate → *PersonDetected*

PersonInside =

leave → *PersonOutside*

□

rotate → *rotate* → *PersonInside*

Question

Are there any problems with this design?

Answer

Yes

Summary

- Concurrency
- Multi-way synchronisation
- Rewriting
- Concurrency and choice

Index

- 2 Contents
- 3 **Concurrency**
- 4 **Sharing events**
- 5 **Step law: parallel**
- 7 **Example**
- 8 **Example**
- 9 **Example**
- 10 **Example**
- 11 **Example: boxes**
- 18 **Process alphabets**
- 19 **Example: boxes**

- 20 **Constraint**
- 21 **Example: boxes**
- 22 **Law**
- 23 **Example: parallel breakfast**
- 24 **Multi-way synchronisation**
- 25 **Example: boxes**
- 31 **Assigning alphabets**
- 32 **Example: boxes**
- 33 **Alphabets are static**
- 34 **aside: alpha**
- 35 **Law**
- 36 **Rewriting**
- 37 **Example: vending machines**

40 Example: Vending machines

45 Example: vending machines

51 Concurrency and choice

52 Example: Agreement

56 Exercise: customer and machines

57 Solution: customer and machines

58 Exercise: Medical accelerator

60 Events

61 Question

62 Answer

63 Process description

64 Question

65 Answer

66 Question

67 Answer

68 Exercise: revolving door

72 Question

73 Answer

74 Summary

75 Index