ETH Zürich
Chair of Software Engineering

Concepts of Concurrent Computation

Bertrand Meyer
Sebastian Nanz

Lecture 10: An introduction to CSP

CSP: Origin

Communicating Sequential Processes: C.A.R. Hoare

1978 paper, based in part on ideas of E.W. Dijkstra (guarded commands, 1978 paper and "A Discipline of Programming" book)

Revised with help of S. D. Brooks and A.W. Roscoe

1985 book, revised 2004

Complete reference: *The Theory and Practice of Concurrency*, A. W. Roscoe, Prentice Hall 1997 (2005) (used extensively in the present slides)

CSP purpose

Concurrency formalism

- Expresses many concurrent situations elegantly
- Influenced design of several concurrent programming languages, in particular Occam (Transputer)

Calculus

- Formally specified: laws
- Makes it possible to prove properties of systems

Traces

A trace is a sequence of events, for example

<coin, coffee, coin, coffee>

Many traces of interest are infinite, for example

<coin, coffee, coin, coffee, …>

(Can be defined formally, e.g by regular expressions, but such traces definition are not part of CSP; they are descriptions of CSP process properties.)

Events come from an *alphabet*. The alphabet of all possible events is written $\sum$ in the following.

Processes and their traces

A CSP process is characterized (although not necessarily defined fully) by the set of its traces. For example a process may have the trace set

{<>,
<coin, coffee>,
<coin, tea>}

The special process STOP has a trace set consisting of a single, empty trace:

{<>}

Basic CSP syntax

P ::=

STOP	| -- Does not engage in any events
$a \rightarrow Q$	| -- Engages in a, then acts like Q
$Q \sqcap R$	| -- Internal choice
$Q \,\square\, R$	| -- External choice
$Q \underset{E}{\parallel} R$	| -- Concurrency (E: subset of alphabet)
$Q \parallel R$	| -- Lock-step concurrency (same as $Q \underset{\Sigma}{\parallel} R$)
$Q \setminus E$	| -- Hiding
$\mu Q \bullet f(Q)$	-- Recursion

Generalization of → notation

Basic:

$$a \rightarrow P$$

Generalization:

$$x{:}\ E \rightarrow P\ (x)$$

Accepts any event from E, then executes $P\ (x)$ where x is that event

Also written

$$?\ x{:}\ E \rightarrow P\ (x)$$

Note that if E is empty then $x{:}\ E \rightarrow P\ (x)$ is $STOP$ for any P

Some laws of concurrency

1. $P \,||\, Q \;=\; Q \,||\, P$
2. $(P \,||\, (Q \,||\, R)) \;=\; ((P \,||\, Q) \,||\, R)$
3. $P \,||\, STOP = STOP$
4. $(c \rightarrow P) \,||\, (c \rightarrow Q) \;=\; (c \rightarrow (P \,||\, Q))$
5. $(c \rightarrow P) \,||\, (d \rightarrow Q) \;=\; STOP$ -- If $c \neq d$
6. $(x\!: A \rightarrow P\,(x)) \,||\, (y\!: B \rightarrow Q\,(y)) =$
   $(z\!: (A \cap B) \rightarrow (P\,(z) \,||\, Q\,(z))$

Basic notions

Processes engage in events

Example of basic notation:

$CVM = (coin \rightarrow coffee \rightarrow coin \rightarrow coffee \rightarrow STOP)$

Right associativity: the above is an abbreviation for

$CVM = (coin \rightarrow (coffee \rightarrow (coin \rightarrow (coffee \rightarrow STOP))))$

Trace set of CVM: {<coin, coffee, coin, coffee>}

The events of a process are taken from its alphabet:

$\alpha(CVM) = \{coin, coffee\}$

STOP can engage in no events

Traces

$$traces\,(e \rightarrow P) = \{\langle e\rangle + s \quad | \quad s \in traces\,(P)\}$$

Exercises: determine traces

P ::=

STOP	| -- Does not engage in any events
$a \rightarrow Q$	| -- Engages in a, then acts like Q
$Q \sqcap R$	| -- Internal choice
$Q \Box R$	| -- External choice
$Q \underset{E}{\|\|} R$	| -- Concurrency (E: subset of alphabet)
$Q \|\| R$	| -- Lock-step concurrency (same as $Q \underset{\Sigma}{\|\|} R$)
$Q \setminus E$	| -- Hiding
$\mu Q \bullet f(Q)$	-- Recursion

Recursion

$CLOCK = (tick \rightarrow CLOCK)$

This is an abbreviation for

$$CLOCK = \mu P \bullet (tick \rightarrow P)$$

A recursive definition is a fixpoint equation. The μ notation denotes the fixpoint

Accepting one of a set of events; channels

Basic notation:

$? \; x : A \; \rightarrow P\,(x)$

Accepts any event from A, then executes $P\,(x)$ where x is that event

Channel names

Example:

$? \; y : c.A \; \rightarrow d.y'$

(where $c.A$ denotes $\{c.x \mid x \in A\}$ and y' denotes y deprived of its initial channel name, e.g. $(c.a)' = a$)

More convenient notation for such cases involving channels:

$\boxed{c? \; x : A \; \rightarrow d!x}$

A simple buffer

$COPY = c?\, x : A \rightarrow d!x \rightarrow COPY$

External choice

$$
\begin{aligned}
COPYBIT = \; & (in.0 \rightarrow out.0 \rightarrow COPYBIT \\
& \;\square \\
& \;in.1 \rightarrow out.1 \rightarrow COPYBIT)
\end{aligned}
$$

External choice

COPY1 = in? x: A → out1!x → COPY1

COPY2 = in? x: B → out2!x → COPY2

COPY3 = COPY1 □ COPY2

External choice

Consider

CHM1 = (in1f → out50rp → out20rp → out20rp → out10rp)

CHM2 = (in1f → out50rp → out50rp)

CHM = CHM1 □ CHM2

Lock-step concurrency

Consider

$P \quad = \quad ?x: A \rightarrow P'$

$Q \quad = \quad ?x: B \rightarrow Q'$

Then

$P \,||\, Q = ?\, x \rightarrow$

- $(P' \,||\, Q')$ if $x \in A \cap B$
- $STOP$ otherwise

(to be generalized soon)

More examples

VMC =

(in2f →

((large → VMC) □

(small → out1f → VMC))

□

(in1f →

((small→ VMC) □

(in1f → large → VMC))

FOOLCUST = (in2f → large → FOOLCUST □

in1f → large → FOOLCUST)

FV = FOOLCUST || VMC =

μP • (in2f →large → P □ in1f → STOP)

Hiding

Consider

$$P = a \rightarrow b \rightarrow Q$$

Assuming Q does not involve b, then

$$P \setminus \{b\} = a \rightarrow Q$$

More generally:

$$(a \rightarrow P) \setminus E =$$

- $P \setminus E$ if $a \in E$
- $a \rightarrow (P \setminus E)$ if $a \notin E$

Hiding introduces internal non-determinism

Consider

$$R = (a \rightarrow P) \;\Box\; (b \rightarrow Q)$$

Then

$$R \setminus \{a, b\} = P \sqcap Q$$

Internal non-deterministic choice

$CH1F = (in1f \rightarrow$
$((out20rp \rightarrow out20rp \rightarrow$
$out20rp \rightarrow out20rp \rightarrow out20rp \rightarrow CH1F)$
$\sqcap$
$(out50rp \rightarrow out50rp \rightarrow CH1F)))$

Non-deterministic internal choice: another application

TRANSMIT (x) = in?x → LOSSY (x)

LOSSY (x) =

out!x → TRANSMIT (x)

⊓ out!x → LOSSY (x)

⊓ TRANSMIT (x)

The general concurrency operator

Consider

$P \quad = \quad ?x{:}\ A \rightarrow P'$

$Q \quad = \quad ?x{:}\ B \rightarrow Q'$

Then

$P \underset{E}{||} Q = ?\, x \rightarrow$

- $P' \underset{E}{||} Q'$ if $x \in E \cap A \cap B$
- $P' \underset{E}{||} Q$ if $x \in A\text{-}B\text{-}E$
- $P \underset{E}{||} Q'$ if $x \in B\text{-}A\text{-}E$
- $(P' \underset{E}{||} Q) \sqcap (P \underset{E}{||} Q')$ if $x \in (A \cap B) - E$

Special cases of concurrency

Lock-step concurrency:

$$P \parallel Q \qquad = P \parallel_{\Sigma} Q$$

Interleaving:

$$P \,|||\, Q \qquad = P \parallel_{\varnothing} Q$$

Lock-step concurrency (reminder)

Consider

$P \quad = \quad ?x{:}\ A \rightarrow P'$

$Q \quad = \quad ?x{:}\ B \rightarrow Q'$

Then

$P \;||\; Q = ?\ x \rightarrow$

- $(P' \;||\; Q')$ if $x \in E \cap A \cap B$
- $STOP$ otherwise

Laws of non-deterministic internal choice

$P \sqcap P = P$

$P \sqcap Q = Q \sqcap P$

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

$x \rightarrow (P \sqcap Q) = (x \rightarrow P) \sqcap (x \rightarrow Q)$

$P \,||\, (Q \sqcap R) = (P \,||\, Q) \sqcap (P \,||\, R)$

$(P \sqcap Q) \,||\, R = (P \,||\, R) \sqcap (Q \,||\, R)$

The recursion operator is not distributive; consider:

$$P = \mu X \bullet ((a \rightarrow X) \sqcap (b \rightarrow X))$$

$$Q = (\mu X \bullet (a \rightarrow X)) \sqcap (\mu X \bullet (b \rightarrow X))$$

Note on external choice

From previous slide:

$$x \rightarrow (P \sqcap Q) = (x \rightarrow P) \sqcap (x \rightarrow Q)$$

The question was asked in class of whether a similar property also applies to external choice $\square$

The conjectured property is

$$x \rightarrow (P \square Q) = (x \rightarrow P) \square (x \rightarrow Q)$$

It does not hold, since

$$(x \rightarrow P) \square (x \rightarrow Q) = x \rightarrow (P \sqcap Q)$$

(As a consequence of rule on next page)

General property of external choice

$(?x: A \rightarrow P) \;\Box\; (?x: B \rightarrow Q) =$

$?x: A \cup B \rightarrow$

- P if $x \in A - B$
- Q if $x \in B - A$
- $P \sqcap Q$ if $x \in A \cap B$

Traces

$$traces\ (e \rightarrow P) = \{\langle e\rangle + s \ \mid\ s \in traces\ (P)\}$$

Exercise: determine traces

P ::=

STOP	| -- Does not engage in any events
$a \rightarrow Q$	| -- Engages in a, then acts like Q
$Q \sqcap R$	| -- Internal choice
$Q \,\square\, R$	| -- External choice
$Q \underset{E}{\vert\vert} R$	| -- Concurrency (E: subset of alphabet)
$Q \vert\vert R$	| -- Lock-step concurrency (same as $Q \underset{\Sigma}{\vert\vert} R$)
$Q \setminus E$	| -- Hiding
$\mu Q \bullet f(Q)$	-- Recursion

Refinement

Process Q *refines* (specifically, *trace-refines*) process P if

$$\text{traces}\ (Q) \subseteq \text{traces}\ (P)$$

For example:

$$P \quad \textit{refines} \quad P \sqcap Q$$

The trace model is not enough

The traces of and are the same:

$$traces\ (P \,\Box\, Q) = traces\ (P) \cup traces\ (Q)$$

$$traces\ (P \sqcap Q) = traces\ (P) \cup traces\ (Q)$$

But the processes can behave differently if for example:

$$P = a \rightarrow b \rightarrow STOP$$

$$Q = b \rightarrow a \rightarrow STOP$$

Traces define what a process may do, not what it may refuse to do

Refusals

For a process P and a trace t of P:

- An event set es ∈ P (∑) is a *refusal set* if P can forever refuse all events in es
- Refusals (P) is the set of P's refusal sets
- Convention: keep only maximal refusal sets

  (if X is a refusal set and Y ⊆ X, then Y is a refusal set)

This also leads to a notion of "failure":

- Failures (P, t) is Refusals (P / t)

where P/t is P *after* t :

traces (P / t) = {u | t + u ∈ traces (P))

Comparing failures

Compare

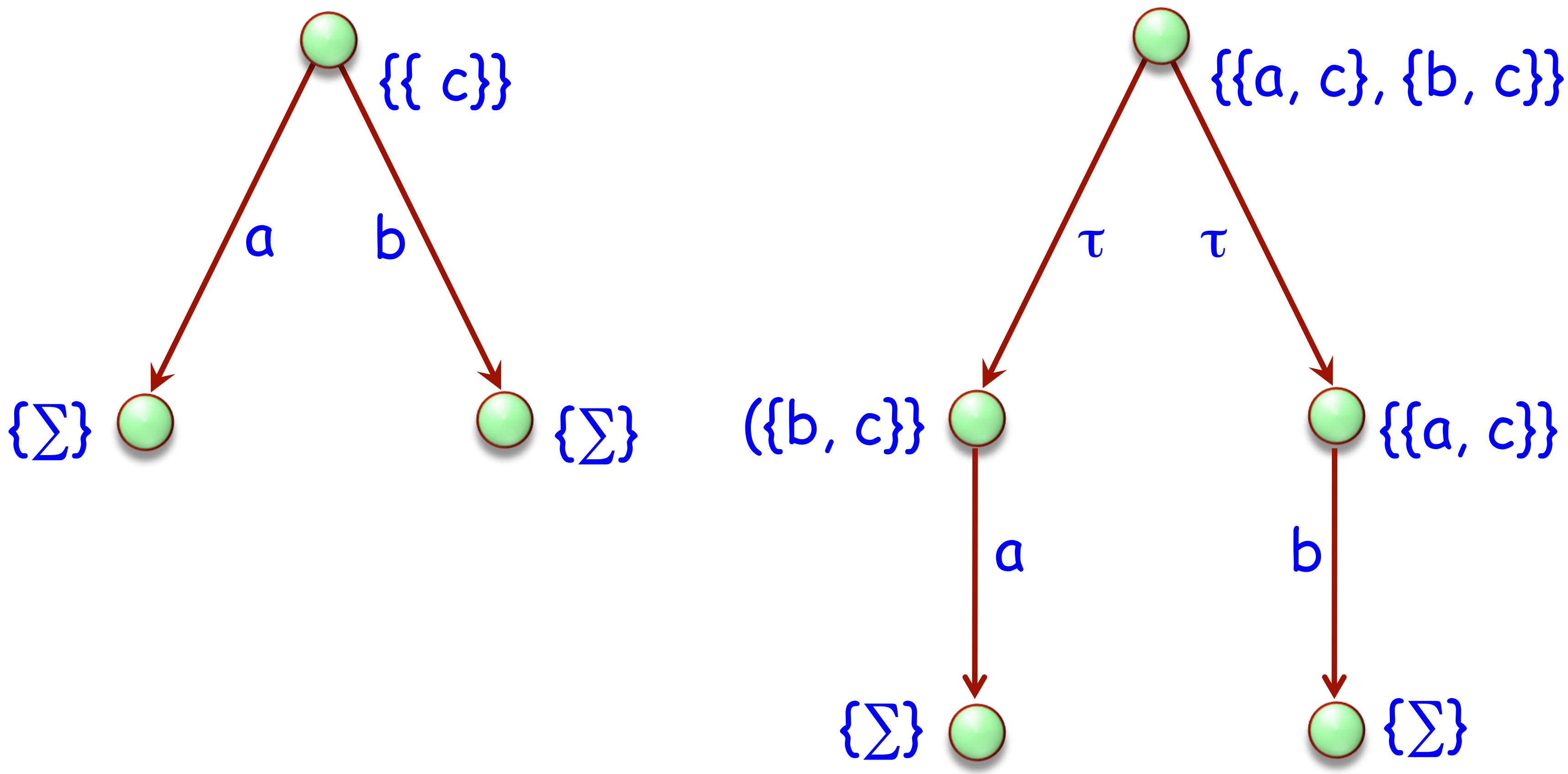
- $P = a \rightarrow STOP \;\Box\; b \rightarrow STOP$
- $Q = a \rightarrow STOP \;\sqcap\; b \rightarrow STOP$

Same traces, but:

- Refusals (P) = $\emptyset$
- Refusals (Q) = $\{\{a\}, \{b\}\}$

Refusal sets (from labeled transition diagram)

$\sum = \{ a, b, c\}$



$a \rightarrow STOP \;\Box\; b \rightarrow STOP$

$a \rightarrow STOP \sqcap b \rightarrow STOP$

A more complete notion of refinement

Process Q *failures-refines* process P if both

traces (Q) ⊆ traces (P)
failures (Q) ⊆ failures (P)

Makes it possible to distinguish between □ and ⊓

Divergence

A process diverges if it is not refusing all events but not communicating with the environment

This happens if a process can engage in an infinite sequence of τ transitions

An example of diverging process:

$$(\mu\ p.a \rightarrow p) \setminus a$$

The divergence model (Brookes, Roscoe)

CSP semantics is often expressed through a failures set

A failure is of the form

$$[s, X]$$

where s is a trace (sequence of events) and X a finite set of events

A failure set must satisfy the following properties:

- $[\langle\rangle, \varnothing] \in F$
- $[s + t, \varnothing] \in F \quad \Rightarrow \quad [s, \varnothing] \in F$
- $[s, X] \in F \wedge \; Y \subseteq X \quad \Rightarrow \quad [s, Y] \in F$
- $[s, X] \in F \wedge [s + \langle c\rangle, \varnothing] \notin F \quad \Rightarrow \quad [s, X \cup \{c\}] \in F$

Basic CSP syntax

$P ::=$

$STOP$	|	-- Does not engage in any events
$a \rightarrow Q$	|	-- Engages in a, then acts like Q
$Q \sqcap R$	|	-- Internal choice
$Q \square R$	|	-- External choice
$Q \underset{E}{\|\|} R$	|	-- Concurrency (E: subset of alphabet)
$Q \|\| R$	|	-- Lock-step concurrency (same as $Q \underset{\Sigma}{\|\|} R$)
$Q \setminus E$	|	-- Hiding
$\mu Q \bullet f(Q)$		-- Recursion

CSP laws in the divergence model (1/2)

$$
\begin{aligned}
P \Box P &\equiv_M P \\
P \Box Q &\equiv_M Q \Box P \\
P \Box (Q \Box R) &\equiv_M (P \Box Q) \Box R \\
P \Box (Q \sqcap R) &\equiv_M (P \Box Q) \sqcap (P \Box R) \\
P \sqcap (Q \Box R) &\equiv_M (P \sqcap Q) \Box (P \sqcap R) \\
P \Box \mathrm{STOP} &\equiv_M P \\
(a \to (P \sqcap Q)) &\equiv_M (a \to P) \sqcap (a \to Q) \\
(a \to P) \Box (a \to Q) &\equiv_M (a \to P) \sqcap (a \to Q) \\
P \sqcap P &\equiv_M P \\
P \sqcap Q &\equiv_M Q \sqcap P \\
P \sqcap (Q \sqcap R) &\equiv_M (P \sqcap Q) \sqcap R \\
P \parallel Q &\equiv_M Q \parallel P \\
P \parallel (Q \parallel R) &\equiv_M (P \parallel Q) \parallel R \\
P \parallel (Q \sqcap R) &\equiv_M (P \parallel Q) \sqcap (P \parallel R) \\
(a \to P) \parallel (b \to Q) &\equiv_M \mathrm{STOP} \qquad \text{if } a \neq b \\
&\equiv_M (a \to (P \parallel Q)) \quad \text{if } a = b \\
P \parallel \mathrm{STOP} &\equiv_M \mathrm{STOP}
\end{aligned}
$$

(From: Brooks & Roscoe 85)

CSP laws (2/2)

$$
\begin{aligned}
P|||Q &\equiv_M Q|||P \\
(P|||Q)|||R &\equiv_M P|||(Q|||R) \\
P|||(Q \sqcap R) &\equiv_M (P|||Q) \sqcap (P|||R) \\
(a \to P)|||(b \to Q) &\equiv_M (a \to (P|||(b \to Q))) \Box (b \to ((a \to P)|||Q)) \\
P; (Q; R) &\equiv_M (P; Q); R \\
\mathrm{STOP}|||Q &\equiv_M Q \\
\mathrm{SKIP}; Q &\equiv_M Q \\
\mathrm{STOP}; Q &\equiv_M \mathrm{STOP} \\
P; (Q \sqcap R) &\equiv_M (P; Q) \sqcap (P; R) \\
(P \sqcap Q); R &\equiv_M (P; R) \sqcap (Q; R) \\
(a \to P); Q &\equiv_M (a \to P; Q) \qquad \text{if } a \neq \checkmark
\end{aligned}
$$

$$
\begin{aligned}
(P\backslash a)\backslash b &\equiv_M (P\backslash b)\backslash a \\
(P\backslash a)\backslash a &\equiv_M P\backslash a \\
(a \to P)\backslash b &\equiv_M (a \to P\backslash b) \qquad \text{if } a \neq b \\
&\equiv_M P\backslash b \qquad \text{if } a = b \\
(P \sqcap Q)\backslash a &\equiv_M (P\backslash a) \sqcap (Q\backslash a)
\end{aligned}
$$

Some extensions

Non-timed:

- The ✔ event (not in Σ): successful termination
- Skip : successfully terminates
- Sequential composition: P ; Q
- $\perp$: diverging process

Timed:

- $P \overset{t}{\lightning} Q$: interrupt
- $P \overset{t}{\triangleright} Q$: timeout
- $a \overset{!}{\rightarrow} P$: communicate immediately
- WAIT t: same as $STOP \overset{t}{\triangleright} SKIP$

Example (Ouaknine)

$V1 = coin.in \rightarrow$

$$((coke \rightarrow V1) \,\Box\, (fanta \rightarrow V1)) \stackrel{60}{\triangleright} (coin.out \xrightarrow{!} V1)$$

Some laws no longer hold

$P \mid\mid STOP = STOP$ if $P \neq \bot$

$\bot \mid\mid STOP = \bot$

$(a \rightarrow P) \setminus b \quad = \quad a \rightarrow (P \setminus b) \quad$ if $a \neq b$

$(a \rightarrow P) \setminus a \quad = \quad P \setminus a$

CSP: Summary

A calculus based on mathematical laws

Provides a general model of computation based on communication

Serves both as specification of concurrent systems and as a guide to implementation

One of the most influential models for concurrency work

CSP: Origin

Communicating Sequential Processes: C.A.R. Hoare

1978 paper, based in part on ideas of E.W. Dijkstra (guarded commands, 1978 paper and "A Discipline of Programming" book)

Revised with help of S. D. Brooks and A.W. Roscoe

1985 book, revised 2004

Complete reference: *The Theory and Practice of Concurrency*, A. W. Roscoe, Prentice Hall 1997 (2005) (used extensively in the present slides)

CSP purpose

Concurrency formalism

- Expresses many concurrent situations elegantly
- Influenced design of several concurrent programming languages, in particular Occam (Transputer)

Calculus

- Formally specified: laws
- Makes it possible to prove properties of systems

Traces

A trace is a sequence of events, for example

<coin, coffee, coin, coffee>

Many traces of interest are infinite, for example

<coin, coffee, coin, coffee, …>

(Can be defined formally, e.g by regular expressions, but such traces definition are not part of CSP; they are descriptions of CSP process properties.)

Events come from an *alphabet*. The alphabet of all possible events is written $\sum$ in the following.

Processes and their traces

A CSP process is characterized (although not necessarily defined fully) by the set of its traces. For example a process may have the trace set

{<>,
<coin, coffee>,
<coin, tea>}

The special process STOP has a trace set consisting of a single, empty trace:

{<>}

Basic CSP syntax

P ::=

STOP	| -- Does not engage in any events
$a \rightarrow Q$	| -- Engages in a, then acts like Q
$Q \sqcap R$	| -- Internal choice
$Q \,\square\, R$	| -- External choice
$Q \underset{E}{\parallel} R$	| -- Concurrency (E: subset of alphabet)
$Q \parallel R$	| -- Lock-step concurrency (same as $Q \underset{\Sigma}{\parallel} R$)
$Q \setminus E$	| -- Hiding
$\mu Q \bullet f(Q)$	-- Recursion

Generalization of → notation

Basic:

$a \rightarrow P$

Generalization:

$x: E \rightarrow P(x)$

Accepts any event from E, then executes $P(x)$ where x is that event

Also written

$?\,x: E \rightarrow P(x)$

Note that if E is empty then $x: E \rightarrow P(x)$ is $STOP$ for any P

Some laws of concurrency

1. $P \mid\mid Q = Q \mid\mid P$
2. $(P \mid\mid (Q \mid\mid R)) = ((P \mid\mid Q) \mid\mid R)$

3. $P \mid\mid STOP = STOP$

4. $(c \rightarrow P) \mid\mid (c \rightarrow Q) = (c \rightarrow (P \mid\mid Q))$
5. $(c \rightarrow P) \mid\mid (d \rightarrow Q) = STOP$ -- If $c \neq d$

6. $(x{:}\ A \rightarrow P\ (x)) \mid\mid (y{:}\ B \rightarrow Q\ (y)) =$

   $(z{:}\ (A \cap B) \rightarrow (P\ (z) \mid\mid Q\ (z))$

Basic notions

Processes engage in events

Example of basic notation:

$CVM = (coin \rightarrow coffee \rightarrow coin \rightarrow coffee \rightarrow STOP)$

Right associativity: the above is an abbreviation for

$CVM = (coin \rightarrow (coffee \rightarrow (coin \rightarrow (coffee \rightarrow STOP))))$

Trace set of CVM: {<coin, coffee, coin, coffee>}

The events of a process are taken from its alphabet:

$\alpha(CVM) = \{coin, coffee\}$

STOP can engage in no events

Traces

$$traces\,(e \rightarrow P) = \{\langle e\rangle + s \quad | \quad s \in traces\,(P)\}$$

Exercises: determine traces

$P ::=$

$STOP$	| -- Does not engage in any events
$a \rightarrow Q$	| -- Engages in a, then acts like Q
$Q \sqcap R$	| -- Internal choice
$Q \Box R$	| -- External choice
$Q \mathbin{\|\|_E} R$	| -- Concurrency (E: subset of alphabet)
$Q \mathbin{\|\|} R$	| -- Lock-step concurrency (same as $Q \mathbin{\|\|_\Sigma} R$)
$Q \setminus E$	| -- Hiding
$\mu Q \bullet f(Q)$	-- Recursion

Recursion

$CLOCK = (tick \to CLOCK)$

This is an abbreviation for

$$CLOCK = \mu P \bullet (tick \to P)$$

A recursive definition is a fixpoint equation. The μ notation denotes the fixpoint

Accepting one of a set of events; channels

Basic notation:

$$? \, x: A \rightarrow P(x)$$

Accepts any event from A, then executes $P(x)$ where x is that event

Channel names

Example:

$$? \, y: c.A \rightarrow d.y'$$

(where $c.A$ denotes $\{c.x \mid x \in A\}$ and y' denotes y deprived of its initial channel name, e.g. $(c.a)' = a$)

More convenient notation for such cases involving channels:

$$\boxed{c? \, x: A \rightarrow d!x}$$

A simple buffer

$COPY = c?\ x{:}\ A \rightarrow d!x \rightarrow COPY$

External choice

$$COPYBIT = (in.0 \rightarrow out.0 \rightarrow COPYBIT \;\Box\; in.1 \rightarrow out.1 \rightarrow COPYBIT)$$

External choice

COPY1 = in? x: A → out1!x → COPY1

COPY2 = in? x: B → out2!x → COPY2

COPY3 = COPY1 □ COPY2

External choice

Consider

CHM1 = (in1f → out50rp → out20rp → out20rp → out10rp)
CHM2 = (in1f → out50rp → out50rp)

CHM = CHM1 □ CHM2

Lock-step concurrency

Consider

$$P \quad = \quad ?x{:}\ A \rightarrow P'$$

$$Q \quad = \quad ?x{:}\ B \rightarrow Q'$$

Then

$$P \mid\mid Q = ?\ x{:}\ A \cap B \rightarrow (P' \mid\mid Q')$$

Note that $P \mid\mid Q$ is $STOP$ for any event $x \notin A \cap B$

(to be generalized soon)

More examples

$$
\begin{aligned}
VMC = {} & \\
& (in2f \rightarrow \\
& \qquad ((large \rightarrow VMC) \;\Box \\
& \qquad (small \rightarrow out1f \rightarrow VMC)) \\
& \Box \\
& (in1f \rightarrow \\
& \qquad ((small \rightarrow VMC) \;\Box \\
& \qquad (in1f \rightarrow large \rightarrow VMC))
\end{aligned}
$$

$$
\begin{aligned}
FOOLCUST = (& in2f \rightarrow large \rightarrow FOOLCUST \;\Box \\
& in1f \rightarrow large \rightarrow FOOLCUST)
\end{aligned}
$$

$$
\begin{aligned}
FV = FOOLCUST \parallel VMC = {} & \\
& \mu P \bullet (in2f \rightarrow large \rightarrow FV \;\Box\; in1f \rightarrow STOP)
\end{aligned}
$$

Hiding

Consider

$$P = a \rightarrow b \rightarrow Q$$

Assuming Q does not involve b, then

$$P \setminus \{b\} = a \rightarrow Q$$

More generally:

$(a \rightarrow P) \setminus E =$

- $P \setminus E$ if $a \in E$
- $a \rightarrow (P \setminus E)$ if $a \notin E$

Hiding introduces internal non-determinism

Consider

$$R \quad = \quad (a \rightarrow P) \;\Box\; (b \rightarrow Q)$$

Then

$$R \setminus \{a, b\} \quad = \quad P \sqcap Q$$

Internal non-deterministic choice

$CH1F = (in1f \rightarrow$

$((out20rp \rightarrow out20rp \rightarrow$

$out20rp \rightarrow out20rp \rightarrow out20rp \rightarrow CH1F)$

$\sqcap$

$(out50rp \rightarrow out50rp \rightarrow CH1F)))$

Non-deterministic internal choice: another application

$$
\begin{aligned}
TRANSMIT(x) &= in?x \rightarrow LOSSY(x) \\[1em]
LOSSY(x) &= \\
&\qquad out!x \rightarrow TRANSMIT(x) \\
&\sqcap \quad out!x \rightarrow LOSSY(x) \\
&\sqcap \quad TRANSMIT(x)
\end{aligned}
$$

The general concurrency operator

Consider

$P \;=\; ?x{:}\,A \rightarrow P'$

$Q \;=\; ?x{:}\,B \rightarrow Q'$

Then

$P \underset{E}{||} Q = ?\,x \rightarrow$

- $P' \underset{E}{||} Q'$ if $x \in E \cap A \cap B$
- $P' \underset{E}{||} Q$ if $x \in A\text{-}B\text{-}E$
- $P \underset{E}{||} Q'$ if $x \in B\text{-}A\text{-}E$
- $(P' \underset{E}{||} Q) \sqcap (P \underset{E}{||} Q')$ if $x \in (A \cap B) - E$

Special cases of concurrency

Lock-step concurrency:

$$P \mid\mid Q \quad = P \underset{\Sigma}{\mid\mid} Q$$

Interleaving:

$$P \mid\mid\mid Q \quad = P \underset{\varnothing}{\mid\mid} Q$$

Lock-step concurrency (reminder)

Consider

$P \quad = \quad ?x\colon A \rightarrow P'$

$Q \quad = \quad ?x\colon B \rightarrow Q'$

Then

$P \mid\mid Q = ?\, x\colon A \cap B \rightarrow (P' \mid\mid Q')$

Note that $P \mid\mid Q$ is $STOP$ for any event $x \notin A \cap B$

(to be generalized soon)

Laws of non-deterministic internal choice

$P \sqcap P = P$

$P \sqcap Q = Q \sqcap P$

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

$x \rightarrow (P \sqcap Q) = (x \rightarrow P) \sqcap (x \rightarrow Q)$

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

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

The recursion operator is not distributive; consider:

$$P = \mu X \bullet ((a \rightarrow X) \sqcap (b \rightarrow X))$$

$$Q = (\mu X \bullet (a \rightarrow X)) \sqcap (\mu X \bullet (b \rightarrow X))$$

Note on external choice

From previous slide:

$$x \rightarrow (P \sqcap Q) = (x \rightarrow P) \sqcap (x \rightarrow Q)$$

The question was asked in class of whether a similar property also applies to external choice $\square$

The conjectured property is

$$x \rightarrow (P \square Q) = (x \rightarrow P) \square (x \rightarrow Q)$$

It does not hold, since

$$(x \rightarrow P) \square (x \rightarrow Q) = x \rightarrow (P \sqcap Q)$$

(As a consequence of rule on next page)

General property of external choice

$(?x: A \rightarrow P) \Box (?x: B \rightarrow Q) =$

$?x: A \cup B \rightarrow$

- P if $x \in A\text{-}B$
- Q if $x \in B\text{-}A$
- $P \sqcap Q$ if $x \in A \cap B$

Traces

$$traces\ (e \rightarrow P) = \{\langle e\rangle + s \quad | \quad s \in traces\ (P)\}$$

Exercise: determine traces

$P ::=$

$STOP$	| -- Does not engage in any events
$a \rightarrow Q$	| -- Engages in a, then acts like Q
$Q \sqcap R$	| -- Internal choice
$Q \Box R$	| -- External choice
$Q \underset{E}{\vert\vert} R$	| -- Concurrency (E: subset of alphabet)
$Q \vert\vert R$	| -- Lock-step concurrency (same as $Q \underset{\Sigma}{\vert\vert} R$)
$Q \setminus E$	| -- Hiding
$\mu Q \bullet f(Q)$	-- Recursion

Refinement

Process Q *refines* (specifically, *trace-refines*) process P if

$$\text{traces}(Q) \subseteq \text{traces}(P)$$

For example:

P *refines* P ⊓ Q

The trace model is not enough

The traces of and are the same:

$$traces\ (P \ \square\ Q) = traces\ (P) \cup traces\ (Q)$$

$$traces\ (P \sqcap Q) = traces\ (P) \cup traces\ (Q)$$

But the processes can behave differently if for example:

$$P = a \rightarrow b \rightarrow STOP$$

$$Q = b \rightarrow a \rightarrow STOP$$

Traces define what a process may do, not what it may refuse to do

Refusals

For a process P and a trace t of P:

- An event set $es \in P(\Sigma)$ is a *refusal set* if P can forever refuse all events in es
- Refusals (P) is the set of P's refusal sets
- Convention: keep only maximal refusal sets

  (if X is a refusal set and $Y \subseteq X$, then Y is a refusal set)

This also leads to a notion of "failure":

- Failures (P, t) is Refusals (P / t)

where P/t is $\boxed{\text{P } \textit{after} \text{ t}}$:

$$traces(P / t) = \{u \mid t + u \in traces(P)\}$$

Comparing failures

Compare

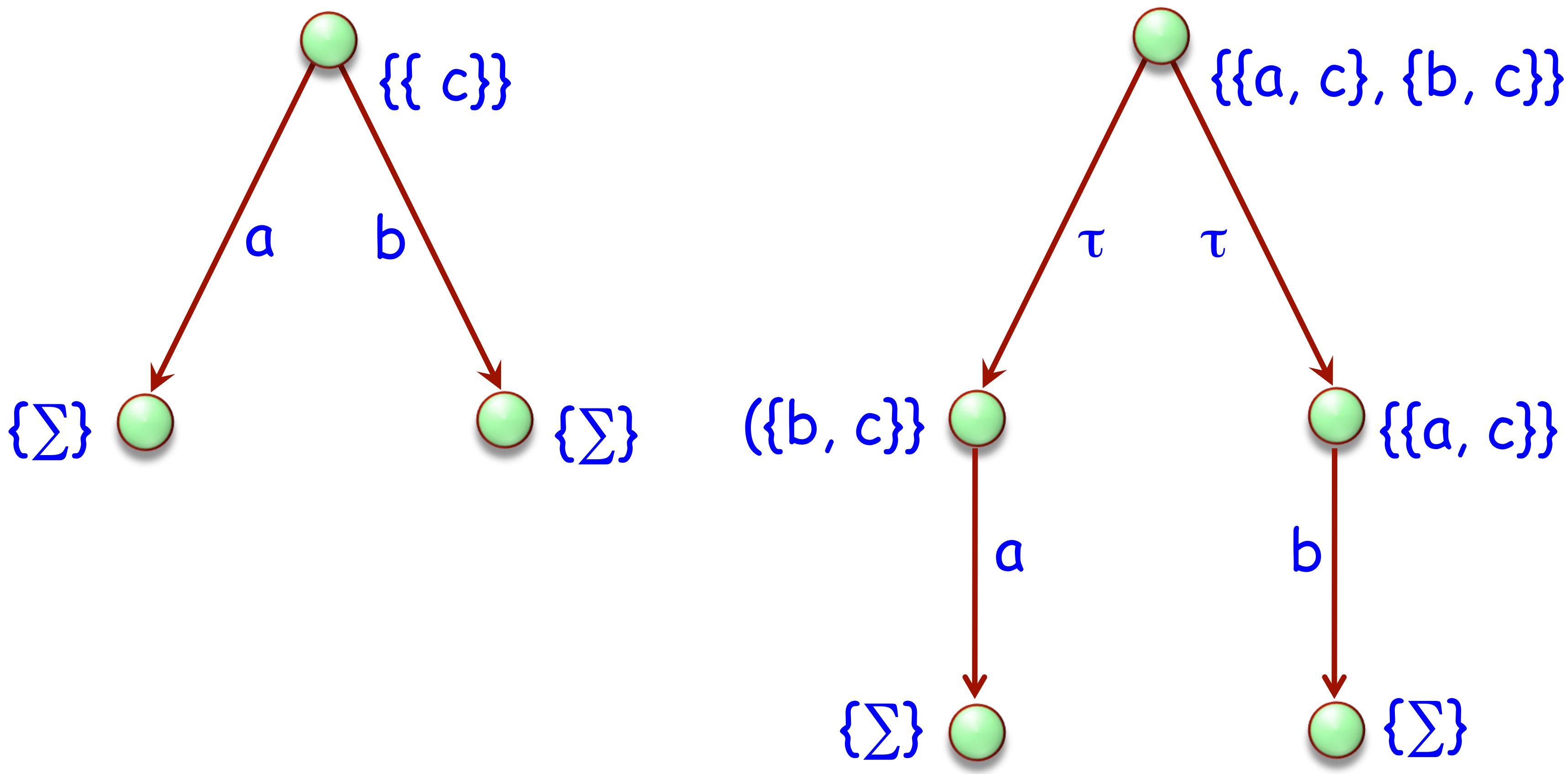
- $P = a \rightarrow STOP \;\Box\; b \rightarrow STOP$
- $Q = a \rightarrow STOP \;\sqcap\; b \rightarrow STOP$

Same traces, but:

- Refusals (P) = ∅
- Refusals (Q) = {{a}, {b}}

Refusal sets (from labeled transition diagram)

$\Sigma = \{ a, b, c\}$



$a \rightarrow STOP \quad \Box \quad b \rightarrow STOP$

$a \rightarrow STOP \sqcap b \rightarrow STOP$

A more complete notion of refinement

Process Q *failures-refines* process P if both

$$\text{traces}(Q) \subseteq \text{traces}(P)$$
$$\text{failures}(Q) \subseteq \text{failures}(P)$$

Makes it possible to distinguish between $\square$ and $\sqcap$

Divergence

A process diverges if it is not refusing all events but not communicating with the environment

This happens if a process can engage in an infinite sequence of τ transitions

An example of diverging process:

$$(\mu\, p.a \rightarrow p) \setminus a$$

The divergence model (Brookes, Roscoe)

CSP semantics is often expressed through a failures set

A failure is of the form

$$[s, X]$$

where s is a trace (sequence of events) and X a finite set of events

A failure set must satisfy the following properties:

- $[\langle\rangle, \varnothing] \in F$
- $[s + t, \varnothing] \in F \;\Rightarrow\; [s, \varnothing] \in F$
- $[s, X] \in F \land Y \subseteq X \;\Rightarrow\; [s, Y] \in F$
- $[s, X] \in F \land [s + \langle c\rangle, \varnothing] \notin F \;\Rightarrow\; [s, X \cup \{c\}] \in F$

Basic CSP syntax

$P ::=$

$STOP$	|	-- Does not engage in any events
$a \rightarrow Q$	|	-- Engages in a, then acts like Q
$Q \sqcap R$	|	-- Internal choice
$Q \square R$	|	-- External choice
$Q \parallel_E R$	|	-- Concurrency (E: subset of alphabet)
$Q \parallel R$	|	-- Lock-step concurrency (same as $Q \parallel_\Sigma R$)
$Q \setminus E$	|	-- Hiding
$\mu Q \bullet f(Q)$		-- Recursion

CSP laws in the divergence model (1/2)

$$
\begin{aligned}
P \Box P &\equiv_M P \\
P \Box Q &\equiv_M Q \Box P \\
P \Box (Q \Box R) &\equiv_M (P \Box Q) \Box R \\
P \Box (Q \sqcap R) &\equiv_M (P \Box Q) \sqcap (P \Box R) \\
P \sqcap (Q \Box R) &\equiv_M (P \sqcap Q) \Box (P \sqcap R) \\
P \Box \mathrm{STOP} &\equiv_M P \\
(a \to (P \sqcap Q)) &\equiv_M (a \to P) \sqcap (a \to Q) \\
(a \to P) \Box (a \to Q) &\equiv_M (a \to P) \sqcap (a \to Q) \\
P \sqcap P &\equiv_M P \\
P \sqcap Q &\equiv_M Q \sqcap P \\
P \sqcap (Q \sqcap R) &\equiv_M (P \sqcap Q) \sqcap R \\
P \parallel Q &\equiv_M Q \parallel P \\
P \parallel (Q \parallel R) &\equiv_M (P \parallel Q) \parallel R \\
P \parallel (Q \sqcap R) &\equiv_M (P \parallel Q) \sqcap (P \parallel R) \\
(a \to P) \parallel (b \to Q) &\equiv_M \mathrm{STOP} \qquad \text{if } a \neq b \\
&\equiv_M (a \to (P \parallel Q)) \quad \text{if } a = b \\
P \parallel \mathrm{STOP} &\equiv_M \mathrm{STOP}
\end{aligned}
$$

(From: Brooks & Roscoe 85)

CSP laws (2/2)

$$
\begin{aligned}
P \,|||\, Q &\equiv_M Q \,|||\, P \\
(P \,|||\, Q) \,|||\, R &\equiv_M P \,|||\, (Q \,|||\, R) \\
P \,|||\, (Q \sqcap R) &\equiv_M (P \,|||\, Q) \sqcap (P \,|||\, R) \\
(a \rightarrow P) \,|||\, (b \rightarrow Q) &\equiv_M (a \rightarrow (P \,|||\, (b \rightarrow Q))) \,\Box\, (b \rightarrow ((a \rightarrow P) \,|||\, Q)) \\
P; (Q; R) &\equiv_M (P; Q); R \\
\mathrm{STOP} \,|||\, Q &\equiv_M Q \\
\mathrm{SKIP}; Q &\equiv_M Q \\
\mathrm{STOP}; Q &\equiv_M \mathrm{STOP} \\
P; (Q \sqcap R) &\equiv_M (P; Q) \sqcap (P; R) \\
(P \sqcap Q); R &\equiv_M (P; R) \sqcap (Q; R) \\
(a \rightarrow P); Q &\equiv_M (a \rightarrow P; Q) \qquad \text{if } a \neq \surd
\end{aligned}
$$

$$
\begin{aligned}
(P \backslash a) \backslash b &\equiv_M (P \backslash b) \backslash a \\
(P \backslash a) \backslash a &\equiv_M P \backslash a \\
(a \rightarrow P) \backslash b &\equiv_M (a \rightarrow P \backslash b) \qquad \text{if } a \neq b \\
&\equiv_M P \backslash b \qquad\qquad\quad \text{if } a = b \\
(P \sqcap Q) \backslash a &\equiv_M (P \backslash a) \sqcap (Q \backslash a)
\end{aligned}
$$

Some extensions

Non-timed:

- The ✔ event (not in $\sum$): successful termination
- Skip : successfully terminates
- Sequential composition: P ; Q
- $\perp$: diverging process

Timed:

- $P \overset{t}{\lightning} Q$: interrupt
- $P \overset{t}{\triangleright} Q$: timeout
- $a \overset{!}{\rightarrow} P$: communicate immediately
- WAIT t: same as $STOP \overset{t}{\triangleright} SKIP$

Example (Ouaknine)

$V1 = coin.in \rightarrow$

$((coke \rightarrow V1) \;\square\; (fanta \rightarrow V1)) \overset{60}{\triangleright} (coin.out \xrightarrow{!} V1)$

Some laws no longer hold

$P \,||\, STOP = STOP$ if $P \neq \perp$

$\perp \,||\, STOP = \perp$

$(a \to P) \setminus b \;=\; a \to (P \setminus b)$ if $a \neq b$

$(a \to P) \setminus a \;=\; P \setminus a$

CSP: Summary

A calculus based on mathematical laws

Provides a general model of computation based on communication

Serves both as specification of concurrent systems and as a guide to implementation

One of the most influential models for concurrency work