Concepts of Concurrent Computation

Bertrand Meyer
Sebastian Nanz
Chris Poskitt

Lecture 10: An introduction to CSP
CSP: Origin

Communicating Sequential Processes: C.A.R. Hoare


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

1985 book, revised 2004

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 $\Sigma$ in the following.
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

\[
\{\langle\rangle, \\
\langle\text{coin, coffee}\rangle, \\
\langle\text{coin, tea}\rangle\}
\]

The special process \textit{STOP} has a trace set consisting of a single, empty trace:

\[
\{\langle\rangle\}
\]
Basic CSP syntax

\[ P ::= \]

\[ STOP \quad | \quad -- \text{Does not engage in any events} \]
\[ a \rightarrow Q \quad | \quad -- \text{Engages in } a, \text{ then acts like } Q \]
\[ Q \Pi R \quad | \quad -- \text{Internal choice} \]
\[ Q \Box R \quad | \quad -- \text{External choice} \]
\[ Q \parallel E R \quad | \quad -- \text{Concurrency (E: subset of alphabet)} \]
\[ Q \parallel R \quad | \quad -- \text{Lock-step concurrency (same as } Q \parallel R) \]
\[ Q \setminus E \quad | \quad -- \text{Hiding} \]
\[ \mu Q \cdot f(Q) \quad -- \text{Recursion} \]
Generalization of \( \rightarrow \) 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. \( P \parallel Q = Q \parallel P \)

2. \( (P \parallel (Q \parallel R)) = ((P \parallel Q) \parallel R) \)

3. \( P \parallel \text{STOP} = \text{STOP} \)

4. \((c \rightarrow P) \parallel (c \rightarrow Q) = (c \rightarrow (P \parallel Q)) \)

5. \((c \rightarrow P) \parallel (d \rightarrow Q) = \text{STOP} \quad \text{-- If } c \neq d \)

6. \((x: A \rightarrow P(x)) \parallel (y: B \rightarrow Q(y)) = \quad (z: (A \cap B) \rightarrow (P(z) \parallel Q(z))) \)
Processes engage in events

Example of basic notation:

\[ CVM = (\text{coin} \rightarrow \text{coffee} \rightarrow \text{coin} \rightarrow \text{coffee} \rightarrow \text{STOP}) \]

Right associativity: the above is an abbreviation for

\[ CVM = (\text{coin} \rightarrow (\text{coffee} \rightarrow (\text{coin} \rightarrow (\text{coffee} \rightarrow \text{STOP})))) \]

Trace set of \( CVM \): \{<\text{coin, coffee, coin, coffee}>\}

The events of a process are taken from its alphabet:

\[ \alpha(CVM) = \{\text{coin, coffee}\} \]

\text{STOP} can engage in no events
Traces

\(\text{traces (e } \rightarrow \text{ P)} = \{<e> + s \mid s \in \text{traces (P)}\}\)
Exercises: determine traces

\[ P ::= \]

\begin{align*}
& \text{STOP} & \quad \text{-- Does not engage in any events} \\
& a \rightarrow Q & \quad \text{-- Engages in } a \text{, then acts like } Q \\
& Q \sqcap R & \quad \text{-- Internal choice} \\
& Q \sqcup R & \quad \text{-- External choice} \\
& Q \parallel E R & \quad \text{-- Concurrency (} E \text{: subset of alphabet)} \\
& Q \parallel R & \quad \text{-- Lock-step concurrency (same as } Q \parallel R \text{)} \\
& Q \setminus E & \quad \text{-- Hiding} \\
& \mu Q \cdot f (Q) & \quad \text{-- Recursion}
\end{align*}
Recursion

CLOCK = (tick → CLOCK)

This is an abbreviation for

CLOCK = μP • (tick → 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

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:

\[ c? x: A \rightarrow d!x \]
A simple buffer

\[ \text{COPY} = c? \ x: \ A \rightarrow \ d!x \rightarrow \ \text{COPY} \]
External choice

COPYBIT = (in.0 → out.0 → COPYBIT
    □
    in.1 → out.1 → COPYBIT)
External choice

\[ \text{COPY1} = \text{in? x: A} \rightarrow \text{out1!x} \rightarrow \text{COPY1} \]

\[ \text{COPY2} = \text{in? x: B} \rightarrow \text{out2!x} \rightarrow \text{COPY2} \]

\[ \text{COPY3} = \text{COPY1} \Box \text{COPY2} \]
Consider

\[ CHM_1 = (\text{in1f} \rightarrow \text{out50rp} \rightarrow \text{out20rp} \rightarrow \text{out20rp} \rightarrow \text{out10rp}) \]
\[ CHM_2 = (\text{in1f} \rightarrow \text{out50rp} \rightarrow \text{out50rp}) \]

\[ CHM = CHM_1 \Box CHM_2 \]
Lock-step concurrency

Consider

\[ P = \exists x: A \rightarrow P' \]
\[ Q = \exists x: B \rightarrow Q' \]

Then

\[ P \parallel Q = \exists x \rightarrow \]
\[ (P' \parallel Q') \quad \text{if } x \in A \cap B \]
\[ \text{STOP} \quad \text{otherwise} \]

(to be generalized soon)
More examples

\[ VMC = \]

\[ (in2f \rightarrow \]

\[ ((large \rightarrow VMC) \square \]

\[ (small \rightarrow out1f \rightarrow VMC) ) \]

\[ \square \]

\[ (in1f \rightarrow \]

\[ ((small \rightarrow VMC) \square \]

\[ (in1f \rightarrow large \rightarrow VMC) ) \]

\[ FOOLCUST = (in2f \rightarrow large \rightarrow FOOLCUST \square \]

\[ in1f \rightarrow large \rightarrow FOOLCUST) \]

\[ FV = FOOLCUST \parallel VMC = \]

\[ \mu P \bullet (in2f \rightarrow large \rightarrow P \square \quad in1f \rightarrow STOP) \]
Hiding

Consider

\[ P = a \rightarrow b \rightarrow Q \]

Assuming \( Q \) does not involve \( b \), then

\[ P \setminus \{b\} = a \rightarrow Q \]

More generally:

\[
(a \rightarrow P) \setminus E = \\
\begin{align*}
\text{if } a \in E & \quad P \setminus E \\
\text{if } a \notin E & \quad a \rightarrow (P \setminus E)
\end{align*}
\]
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

\[ CH1F = (in1f \rightarrow
\quad ((out20rp \rightarrow out20rp \rightarrow
\quad out20rp \rightarrow out20rp \rightarrow out20rp \rightarrow CH1F)) \]

\[ \Pi \]

\[ (out50rp \rightarrow out50rp \rightarrow CH1F)) \]
Non-deterministic internal choice: another application

\[
\text{TRANSMIT} (x) = \text{in} \, ? x \rightarrow \text{LOSSY} (x)
\]

\[
\text{LOSSY} (x) = \begin{array}{l}
\text{out} \, ! x \rightarrow \text{TRANSMIT} (x) \\
\Pi \text{out} \, ! x \rightarrow \text{LOSSY} (x) \\
\Pi \text{TRANSMIT} (x)
\end{array}
\]
The general concurrency operator

Consider

\[ P = ?x: A \rightarrow P' \]
\[ Q = ?x: B \rightarrow Q' \]

Then

\[ P \parallel Q = ?x \rightarrow \]
\[ \begin{align*}
E & : P' \parallel Q' & \text{if } x \in E \cap A \cap B \\
E & : P' \parallel Q & \text{if } x \in A-B-E \\
E & : P \parallel Q' & \text{if } x \in B-A-E \\
E & : (P' \parallel Q) \cap (P \parallel Q') & \text{if } x \in (A \cap B) - E
\end{align*} \]
Special cases of concurrency

Lock-step concurrency:

\[ P \parallel Q = P \parallel Q \]

Interleaving:

\[ P \parallel\parallel Q = P \parallel Q \]
Lock-step concurrency (reminder)

Consider

\[ P = \exists x: A \rightarrow P' \]
\[ Q = \exists x: B \rightarrow Q' \]

Then

\[ P \parallel Q = \exists x \rightarrow \]
- \( (P' \parallel Q') \) if \( x \in E \cap A \cap B \)
- STOP otherwise
Laws of non-deterministic internal choice

\[ P \sqcap P = P \]
\[ P \sqcap Q = Q \sqcap P \]
\[ P \sqcap (Q \sqcap R) = (P \sqcap Q) \sqcap R \]
\[ x \rightarrow (P \sqcap Q) = (x \rightarrow P) \sqcap (x \rightarrow Q) \]

\[ P \parallel (Q \sqcap R) = (P \parallel Q) \sqcap (P \parallel R) \]
\[ (P \sqcap Q) \parallel R = (P \parallel R) \sqcap (Q \parallel R) \]

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:

\[ x \rightarrow (P \land Q) = (x \rightarrow P) \land (x \rightarrow Q) \]

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

The conjectured property is

\[ x \rightarrow (P \Box Q) = (x \rightarrow P) \Box (x \rightarrow Q) \]

It does not hold, since

\[ (x \rightarrow P) \Box (x \rightarrow Q) = x \rightarrow (P \land Q) \]

(As a consequence of rule on next page)
General property of external choice

$(\exists x: A \to P) \Box (\exists x: B \to Q) =$

$\exists x: A \cup B \to$

- $P$ if $x \in A - B$
- $Q$ if $x \in B - A$
- $P \land Q$ if $x \in A \cap B$
Traces

\[
\text{traces } (e \rightarrow P) = \{<e> + s \mid s \in \text{traces } (P)\}
\]
Exercise: determine traces

\[ P ::= \]

- **STOP** | -- Does not engage in any events
- **a \rightarrow Q** | -- Engages in \( a \), then acts like \( Q \)
- **Q \sqcup R** | -- Internal choice
- **Q \boxdot R** | -- External choice
- **Q \parallel E R** | -- Concurrency (\( E \): subset of alphabet)
- **Q \parallel R** | -- Lock-step concurrency (same as \( Q \parallel R \))
- **Q \setminus E** | -- Hiding
- **\mu Q \cdot f(Q)** | -- Recursion
Refinement

Process $Q$ \textit{refines} (specifically, \textit{trace-refines}) process $P$ if

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

For example:

$$P \text{ \textit{refines} } P \sqcap Q$$
The trace model is not enough

The traces of $P$ and $Q$ are the same:
\[
\text{traces (}P \square Q\text{)} = \text{traces (}P\text{)} \cup \text{traces (}Q\text{)}
\]
\[
\text{traces (}P \sqcap Q\text{)} = \text{traces (}P\text{)} \cup \text{traces (}Q\text{)}
\]

But the processes can behave differently if for example:
\[
P = a \rightarrow b \rightarrow \text{STOP}
\]
\[
Q = b \rightarrow a \rightarrow \text{STOP}
\]

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 $P$ after $t$:

\[
\text{traces (} P / t \text{)} = \{ u \mid t + u \in \text{traces (} P \text{)} \} 
\]
Comparing failures

Compare

- \( P = a \to \text{STOP} \ \Box \ b \to \text{STOP} \)
- \( Q = a \to \text{STOP} \ \sqcap \ b \to \text{STOP} \)

Same traces, but:

- \( \text{Refusals (P)} = \emptyset \)
- \( \text{Refusals (Q)} = \{\{a\}, \{b\}\} \)
Refusal sets (from labeled transition diagram)

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

$$\{ \sum \} \xrightarrow{a} \{ \Sigma \} \xrightarrow{b} \{ \Sigma \}$$

$$\{ \Sigma \} \xrightarrow{\tau} \{ \{ c \} \}$$

$$\{ \{ a, c \}, \{ b, c \} \} \xrightarrow{\tau} \{ \{ a, c \}, \{ b, c \} \}$$

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

$$a \rightarrow \text{STOP} \quad \square \quad b \rightarrow \text{STOP}$$

$$a \rightarrow \text{STOP} \quad \pi \quad b \rightarrow \text{STOP}$$
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 $\Box$
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 \( \tau \) 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:

- \([<> , \emptyset] \in F\]
- \([s + t , \emptyset] \in F \Rightarrow [s , \emptyset] \in F\]
- \([s , X] \in F \land Y \subseteq X \Rightarrow [s , Y] \in F\]
- \([s , X] \in F \land [s + <c>, \emptyset] \notin F \Rightarrow [s , X \cup \{c\}] \in F\]
Basic CSP syntax

\[ P ::= \]

\[ \text{STOP} \mid -- \text{Does not engage in any events} \]

\[ a \rightarrow Q \mid -- \text{Engages in } a, \text{ then acts like } Q \]

\[ Q \sqcap R \mid -- \text{Internal choice} \]

\[ Q \square R \mid -- \text{External choice} \]

\[ Q \parallel_{E} R \mid -- \text{Concurrency (} E \text{: subset of alphabet)} \]

\[ Q \parallel R \mid -- \text{Lock-step concurrency (same as } Q \parallel R \text{)} \]

\[ Q \setminus E \mid -- \text{Hiding} \]

\[ \mu Q \cdot f(Q) \mid -- \text{Recursion} \]
CSP laws in the divergence model (1/2)

\[ \begin{align*}
P \lozenge P & \equiv_M P \\
P \lozenge Q & \equiv_M Q \lozenge P \\
\begin{array}{c}
P \lozenge (Q \lozenge R) \\
\equiv_M (P \lozenge Q) \lozenge R
\end{array} \\
\begin{array}{c}
P \lozenge (Q \cap R) \\
\equiv_M (P \lozenge Q) \cap (P \lozenge R)
\end{array} \\
\begin{array}{c}
P \cap (Q \lozenge R) \\
\equiv_M (P \cap Q) \lozenge (P \cap R)
\end{array} \\
\begin{array}{c}
P \lozenge \text{STOP} \\
\equiv_M P
\end{array} \\
\begin{array}{c}
(a \rightarrow (P \cap Q)) \\
\equiv_M (a \rightarrow P) \cap (a \rightarrow Q)
\end{array} \\
\begin{array}{c}
(a \rightarrow P) \lozenge (a \rightarrow Q) \\
\equiv_M (a \rightarrow P) \cap (a \rightarrow Q)
\end{array} \\
\begin{array}{c}
P \cap P \\
\equiv_M P
\end{array} \\
\begin{array}{c}
P \cap Q \\
\equiv_M Q \cap P
\end{array} \\
\begin{array}{c}
P \cap (Q \cap R) \\
\equiv_M (P \cap Q) \cap R
\end{array} \\
\begin{array}{c}
P \parallel Q \\
\equiv_M Q \parallel P
\end{array} \\
\begin{array}{c}
P \parallel (Q \parallel R) \\
\equiv_M (P \parallel Q) \parallel R
\end{array} \\
\begin{array}{c}
P \parallel (Q \cap R) \\
\equiv_M (P \parallel Q) \cap (P \parallel R)
\end{array} \\
\begin{array}{c}
(a \rightarrow P) \parallel (b \rightarrow Q) \\
\equiv_M \text{STOP} \quad \text{if } a \neq b
\end{array} \\
\begin{array}{c}
\equiv_M (a \rightarrow (P \parallel Q)) \quad \text{if } a = b
\end{array} \\
P \parallel \text{STOP} \equiv_M \text{STOP}
\end{align*} \]

(From: Brooks & Roscoe 85)
CSP laws (2/2)

\[ P || Q \equiv_M Q || P \]
\[ (P || Q) || R \equiv_M P || (Q || R) \]
\[ P || (Q \cap R) \equiv_M (P || Q) \cap (P || R) \]
\[ (a \rightarrow P) || (b \rightarrow Q) \equiv_M (a \rightarrow (P || (b \rightarrow Q))) \Box (b \rightarrow ((a \rightarrow P) || Q)) \]
\[ P; (Q; R) \equiv_M (P; Q); R \]
\[ STOP || Q \equiv_M Q \]
\[ SKIP; Q \equiv_M Q \]
\[ STOP; Q \equiv_M STOP \]
\[ P; (Q \cap R) \equiv_M (P; Q) \cap (P; R) \]
\[ (P \cap Q); R \equiv_M (P; R) \cap (Q; R) \]
\[ (a \rightarrow P); Q \equiv_M (a \rightarrow P; Q) \quad \text{if } a \neq \checkmark \]
\[ (P \backslash a) \backslash b \equiv_M (P \backslash b) \backslash a \]
\[ (P \backslash a) \backslash a \equiv_M P \backslash a \]
\[ (a \rightarrow P) \backslash b \equiv_M (a \rightarrow P \backslash b) \quad \text{if } a \neq b \]
\[ \equiv_M P \backslash b \quad \text{if } a = b \]
\[ (P \cap Q) \backslash a \equiv_M (P \backslash a) \cap (Q \backslash a) \]
Some extensions

Non-timed:

- The ✓ event (not in \( \Sigma \)): successful termination
- Skip: successfully terminates
- Sequential composition: \( P ; Q \)
- \( \bot \): diverging process

Timed:

- \( P \overset{\dagger}{\not\leq} Q \): interrupt
- \( P \overset{\dagger}{\implies} Q \): timeout
- \( a \overset{!}{\rightarrow} P \): communicate immediately
- \textit{WAIT} \( \dagger \): same as STOP \( \dagger SL \) SKIP
Example (Ouaknine)

\[ V1 = \text{coin.in} \rightarrow ((\text{coke} \rightarrow V1) \Box (\text{fanta} \rightarrow V1)) \uparrow (\text{coin.out} \rightarrow V1) \]
Some laws no longer hold

\[ P \parallel \text{STOP} = \text{STOP} \text{ if } P \neq \bot \]
\[ \bot \parallel \text{STOP} = \bot \]

\[(a \rightarrow P) \setminus b = a \rightarrow (P \setminus b) \text{ if } a \neq b\]
\[(a \rightarrow P) \setminus a = P \setminus a\]
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
CSP: Origin

Communicating Sequential Processes: C.A.R. Hoare


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

1985 book, revised 2004

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 \textit{alphabet}. The alphabet of all possible events is written $\Sigma$ 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

\[
\{<>,
<\text{coin, coffee}>,
<\text{coin, tea}>\}
\]

The special process \texttt{STOP} has a trace set consisting of a single, empty trace:

\[
\{<\text{}>\}
\]
Basic CSP syntax

\[ P ::= \]

\[ \text{STOP} \quad | \quad \text{-- Does not engage in any events} \]

\[ a \rightarrow Q \quad | \quad \text{-- Engages in } a, \text{ then acts like } Q \]

\[ Q \sqcap R \quad | \quad \text{-- Internal choice} \]

\[ Q \square R \quad | \quad \text{-- External choice} \]

\[ Q \parallel R \quad | \quad \text{-- Concurrency (} E \text{: subset of alphabet)} \]

\[ Q \parallel R \quad | \quad \text{-- Lock-step concurrency (same as } Q \parallel R) \]

\[ Q \setminus E \quad | \quad \text{-- Hiding} \]

\[ \mu Q \cdot f (Q) \quad | \quad \text{-- Recursion} \]
Generalization of \( \rightarrow \) 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. $P \parallel Q = Q \parallel P$

2. $(P \parallel (Q \parallel R)) = ((P \parallel Q) \parallel R)$

3. $P \parallel \text{STOP} = \text{STOP}$

4. $(c \rightarrow P) \parallel (c \rightarrow Q) = (c \rightarrow (P \parallel Q))$

5. $(c \rightarrow P) \parallel (d \rightarrow Q) = \text{STOP}$
   -- If $c \neq d$

6. $(x: A \rightarrow P(x)) \parallel (y: B \rightarrow Q(y)) =$
   $(z: (A \cap B) \rightarrow (P(z) \parallel Q(z)))$
Basic notions

Processes engage in events

Example of basic notation:

\[ CVM = (\text{coin} \rightarrow \text{coffee} \rightarrow \text{coin} \rightarrow \text{coffee} \rightarrow \text{STOP}) \]

Right associativity: the above is an abbreviation for

\[ CVM = (\text{coin} \rightarrow (\text{coffee} \rightarrow (\text{coin} \rightarrow (\text{coffee} \rightarrow \text{STOP})))) \]

Trace set of \( CVM \): \( \{<\text{coin, coffee, coin, coffee}>\} \)

The events of a process are taken from its alphabet:

\[ \alpha(CVM) = \{\text{coin, coffee}\} \]

STOP can engage in no events
Traces

\[ \text{traces (e } \rightarrow \text{ P)} = \{ \langle e \rangle + s \mid s \in \text{traces (P)} \} \]
Exercises: determine traces

\[ P ::= \]

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

\[ \text{CLOCK} = (\text{tick} \rightarrow \text{CLOCK}) \]

This is an abbreviation for

\[ \text{CLOCK} = \mu P \bullet (\text{tick} \rightarrow P) \]

A recursive definition is a fixpoint equation. The \( \mu \) 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

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:

\[ c? x: A \rightarrow d!x \]
A simple buffer

\[\text{COPY} = c? \ x: \ A \rightarrow \ d!x \rightarrow \ \text{COPY}\]
External choice

\[
\text{COPYBIT} = (\text{in.0} \rightarrow \text{out.0} \rightarrow \text{COPYBIT} \\
\quad \text{□} \\
\quad \text{in.1} \rightarrow \text{out.1} \rightarrow \text{COPYBIT})
\]
External choice

COPY1 = in? x: A \rightarrow out1!x \rightarrow COPY1

COPY2 = in? x: B \rightarrow out2!x \rightarrow COPY2

COPY3 = COPY1 \boxdot COPY2
Consider

$CHM1 = (in1f \rightarrow out50rp \rightarrow out20rp \rightarrow out20rp \rightarrow out10rp)$

$CHM2 = (in1f \rightarrow out50rp \rightarrow out50rp)$

$CHM = CHM1 \boxdot CHM2$
Lock-step concurrency

Consider

\[ P = \exists x: A \rightarrow P' \]
\[ Q = \exists x: B \rightarrow Q' \]

Then

\[ P || Q = \exists x: A \cap B \rightarrow (P' || Q') \]

Note that \( P || Q \) is STOP for any event \( x \notin A \cap B \)

(to be generalized soon)
More examples

\[ \text{VMC} = \]
\[ (\text{in2f} \rightarrow \]
\[ ((\text{large} \rightarrow \text{VMC}) \Box \]
\[ (\text{small} \rightarrow \text{out1f} \rightarrow \text{VMC})) \]
\[ \Box \]
\[ (\text{in1f} \rightarrow \]
\[ ((\text{small} \rightarrow \text{VMC}) \Box \]
\[ (\text{in1f} \rightarrow \text{large} \rightarrow \text{VMC})) \]

\[ \text{FOOLCUST} = (\text{in2f} \rightarrow \text{large} \rightarrow \text{FOOLCUST} \ Box \]
\[ \text{in1f} \rightarrow \text{large} \rightarrow \text{FOOLCUST}) \]

\[ \text{FV} = \text{FOOLCUST} \parallel \text{VMC} = \]
\[ \mu P \bullet (\text{in2f} \rightarrow \text{large} \rightarrow \text{FV} \ Box \text{in1f} \rightarrow \text{STOP}) \]
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

\[ R \setminus \{a, b\} = P \cap Q \]
Internal non-deterministic choice

\[ CH1F = (\text{in}1f \rightarrow \) \\
\quad ((\text{out}20\text{rp} \rightarrow \text{out}20\text{rp} \rightarrow \\
\quad \text{out}20\text{rp} \rightarrow \text{out}20\text{rp} \rightarrow \text{out}20\text{rp} \rightarrow CH1F) \\
\quad \prod \\
\quad (\text{out}50\text{rp} \rightarrow \text{out}50\text{rp} \rightarrow CH1F)) \]
Non-deterministic internal choice: another application

\[
\text{TRANSMIT} (x) = \text{in}?x \rightarrow \text{LOSSY} (x)
\]

\[
\text{LOSSY} (x) = \begin{align*}
& \text{out}!x \rightarrow \text{TRANSMIT} (x) \\
\& \prod & \text{out}!x \rightarrow \text{LOSSY} (x) \\
\& \prod & \text{TRANSMIT} (x)
\end{align*}
\]
The general concurrency operator

Consider

\[ P = \ ?x: A \rightarrow P' \]
\[ Q = \ ?x: B \rightarrow Q' \]

Then

\[ P \parallel E Q = \ ?x \rightarrow \]
\[ \begin{align*}
& P' \parallel E Q' \quad \text{if } x \in E \cap A \cap B \\
& P' \parallel E Q \quad \text{if } x \in A-B-E \\
& P \parallel E Q' \quad \text{if } x \in B-A-E \\
& (P' \parallel E Q) \cap (P \parallel E Q') \quad \text{if } x \in (A \cap B) - E
\end{align*} \]
Special cases of concurrency

Lock-step concurrency:

\[ P \parallel Q = P \parallel Q \]

Interleaving:

\[ P \parallel\parallel Q = P \parallel Q \]
Lock-step concurrency (reminder)

Consider

\[ P = ?x: A \rightarrow P' \]
\[ Q = ?x: B \rightarrow Q' \]

Then

\[ P \parallel Q = ?x: A \cap B \rightarrow (P' \parallel Q') \]

Note that \( P \parallel Q \) is STOP for any event \( x \notin A \cap B \)

(to be generalized soon)
Laws of non-deterministic internal choice

\[ P \land P = P \]
\[ P \land Q = Q \land P \]
\[ P \land (Q \land R) = (P \land Q) \land R \]
\[ x \rightarrow (P \land Q) = (x \rightarrow P) \land (x \rightarrow Q) \]

\[ P \parallel (Q \land R) = (P \parallel Q) \land (P \parallel R) \]
\[ (P \land Q) \parallel R = (P \parallel R) \land (Q \parallel R) \]

The recursion operator is not distributive; consider:

\[ P = \mu X \bullet ((a \rightarrow X) \land (b \rightarrow X)) \]
\[ Q = (\mu X \bullet (a \rightarrow X)) \land (\mu X \bullet (b \rightarrow X)) \]
Note on external choice

From previous slide:
\[ x \to (P \land Q) = (x \to P) \land (x \to Q) \]

The question was asked in class of whether a similar property also applies to external choice \(\Box\)

The conjectured property is
\[ x \to (P \Box Q) = (x \to P) \Box (x \to Q) \]

It does not hold, since
\[ (x \to P) \Box (x \to Q) = x \to (P \land Q) \]
(As a consequence of rule on next page)
General property of external choice

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

$?x: A \cup B \to$

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

\[ \text{traces (e \rightarrow P)} = \{<e> + s \mid s \in \text{traces (P)}\} \]
Exercise: determine traces

\[ P ::= \]

- **STOP** | -- Does not engage in any events
- \( a \to Q \) | -- Engages in \( a \), then acts like \( Q \)
- \( Q \parallel R \) | -- Internal choice
- \( Q \square R \) | -- External choice
- \( Q \parallel\ E \parallel R \) | -- Concurrency (\( E \): subset of alphabet)
- \( Q \parallel\ R \) | -- Lock-step concurrency (same as \( Q \parallel\ R \) \( \sum \))
- \( Q \setminus E \) | -- Hiding
- \( \mu Q \cdot f (Q) \) | -- Recursion
Refinement

Process $Q \text{ refines}$ (specifically, trace-refines) process $P$ if

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

For example:

$$P \text{ refines } P \cong Q$$
The trace model is not enough

The traces of and are the same:

\[ \text{traces (} P \square Q) = \text{traces (} P) \cup \text{traces (} Q) \]
\[ \text{traces (} P \cap Q) = \text{traces (} P) \cup \text{traces (} Q) \]

But the processes can behave differently if for example:

\[ P = a \to b \to \text{STOP} \]
\[ Q = b \to a \to \text{STOP} \]

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 $es \in P(\sum)$ is a \textit{refusal set} if $P$ can forever refuse all events in $es$.
- $\text{Refusals}(P)$ 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”:

- $\text{Failures}(P, t)$ is $\text{Refusals}(P/t)$

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

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

Compare

- \( P = a \rightarrow \text{STOP} \quad \Box \quad b \rightarrow \text{STOP} \)
- \( Q = a \rightarrow \text{STOP} \quad \Pi \quad b \rightarrow \text{STOP} \)

Same traces, but:

- \( \text{Refusals (P)} = \emptyset \)
- \( \text{Refusals (Q)} = \{\{a\}, \{b\}\} \)
Refusal sets (from labeled transition diagram)

\[ \sum = \{ a, b, c \} \]

\[ \{\Sigma\} \rightarrow \{\Sigma\} \]

\[ \{\Sigma\} \rightarrow \{\Sigma\} \]

\[ \{\Sigma\} \rightarrow \{\Sigma\} \]

\[ a \rightarrow \text{STOP} \quad \square \quad b \rightarrow \text{STOP} \]

\[ a \rightarrow \text{STOP} \quad \pi \quad b \rightarrow \text{STOP} \]
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 $\tau$ 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:

- $[\langle\rangle, \emptyset] \in F$
- $[s + t, \emptyset] \in F \Rightarrow [s, \emptyset] \in F$
- $[s, X] \in F \land Y \subseteq X \Rightarrow [s, Y] \in F$
- $[s, X] \in F \land [s + \langle c \rangle, \emptyset] \notin F \Rightarrow [s, X \cup \{c\}] \in F$
Basic CSP syntax

\( P ::= \)

\( \text{STOP} \mid \text{-- Does not engage in any events} \)

\( a \rightarrow Q \mid \text{-- Engages in } a, \text{ then acts like } Q \)

\( Q \sqcap R \mid \text{-- Internal choice} \)

\( Q \square R \mid \text{-- External choice} \)

\( Q \parallel R \mid \text{-- Concurrency (} E: \text{ subset of alphabet)} \)

\( Q \parallel R \mid \text{-- Lock-step concurrency (same as } Q \parallel R \noalign{\text{}} \Sigma) \)

\( Q \setminus E \mid \text{-- Hiding} \)

\( \mu Q \cdot f(Q) \mid \text{-- Recursion} \)
CSP laws in the divergence model (1/2)

\[
\begin{align*}
   P \square P & \equiv_M P \\
   P \square Q & \equiv_M Q \square P \\
   P \square (Q \square R) & \equiv_M (P \square Q) \square R \\
   P \square (Q \cap R) & \equiv_M (P \square Q) \cap (P \square R) \\
   P \cap (Q \square R) & \equiv_M (P \cap Q) \square (P \cap R) \\
   P \square \text{STOP} & \equiv_M P \\
   (a \to (P \cap Q)) & \equiv_M (a \to P) \cap (a \to Q) \\
   (a \to P) \square (a \to Q) & \equiv_M (a \to P) \cap (a \to Q) \\
   P \cap P & \equiv_M P \\
   P \cap Q & \equiv_M Q \cap P \\
   P \cap (Q \cap R) & \equiv_M (P \cap Q) \cap R \\
   P \| Q & \equiv_M Q \| P \\
   P \| (Q \| R) & \equiv_M (P \| Q) \| R \\
   P \| (Q \cap R) & \equiv_M (P \| Q) \cap (P \| R) \\
   (a \to P) \| (b \to Q) & \equiv_M \text{STOP} \quad \text{if } a \neq b \\
   & \equiv_M (a \to (P \| Q)) \quad \text{if } a = b \\
   P \| \text{STOP} & \equiv_M \text{STOP}
\end{align*}
\]

(From: Brooks & Roscoe 85)
CSP laws (2/2)

\[
P || Q \equiv_M Q || P \\
(P || Q) || R \equiv_M P || (Q || R) \\
P || (Q \cap R) \equiv_M (P || Q) \cap (P || R) \\
(a \rightarrow P) || (b \rightarrow Q) \equiv_M (a \rightarrow (P || (b \rightarrow Q))) \sqcap (b \rightarrow ((a \rightarrow P) || Q)) \\
P; (Q; R) \equiv_M (P; Q); R \\
STOP || Q \equiv_M Q \\
SKIP; Q \equiv_M Q \\
STOP; Q \equiv_M STOP \\
P; (Q \cap R) \equiv_M (P; Q) \cap (P; R) \\
(P \cap Q); R \equiv_M (P; R) \cap (Q; R) \\
(a \rightarrow P); Q \equiv_M (a \rightarrow P; Q) \quad \text{if } a \neq \sqrt{1} \\
(P \setminus a) \setminus b \equiv_M (P \setminus b) \setminus a \\
(P \setminus a) \setminus a \equiv_M P \setminus a \\
(a \rightarrow P) \setminus b \equiv_M (a \rightarrow P \setminus b) \quad \text{if } a \neq b \\
\equiv_M P \setminus b \quad \text{if } a = b \\
(P \cap Q) \setminus a \equiv_M (P \setminus a) \cap (Q \setminus a)
Some extensions

Non-timed:

- The ✓ event (not in Σ): successful termination
- Skip : successfully terminates
- Sequential composition: P ; Q
- ⊥ : diverging process

Timed:

- P ⊥ Q: interrupt
- P ⊳ Q: timeout
- a !→ P: communicate immediately
- WAIT ⊳: same as STOP ⊳ SKIP
Example (Ouaknine)

\[ V1 = \text{coin.in} \rightarrow \]

\[((\text{coke} \rightarrow V1) \Box (\text{fanta} \rightarrow V1)) \uparrow (\text{coin.out} \rightarrow V1)\]
Some laws no longer hold

\[ P || \text{STOP} = \text{STOP} \text{ if } P \neq \bot \]
\[ \bot || \text{STOP} = \bot \]

\[ (a \rightarrow P) \setminus b = a \rightarrow (P \setminus b) \text{ if } a \neq b \]
\[ (a \rightarrow P) \setminus a = P \setminus a \]
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