Concurrent Object-Oriented Programming

Bertrand Meyer, Sebastian Nanz

Lecture 9: 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 $\sum$ 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

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

The special process \textit{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 \( Q \)

\( Q \parallel R \) \( | \) -- Internal choice

\( Q \parallel R \) \( | \) -- External choice

\( Q \parallel R \) \( | \) -- 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
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. \( P \parallel Q = Q \parallel P \)
2. \( (P \parallel (Q \parallel R)) = ((P \parallel Q) \parallel R) \)
3. \( P \parallel \text{STOP} = \text{STOP} \)
4. \( (c \to P) \parallel (c \to Q) = (c \to (P \parallel Q)) \)
5. \( (c \to P) \parallel (d \to Q) = \text{STOP} \quad \text{-- If } c \neq d \)
6. \( (x: A \to P (x)) \parallel (y: B \to Q (y)) = (z: (A \cap B) \to (P (z) \parallel Q (z))) \)
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

\[ \text{traces (e → P)} = \{<e> + s \mid s \in \text{traces (P)}\} \]
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 \text{, 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} \sqcup_{E} \text{R} \quad | \quad \text{-- Concurrency (E: subset of alphabet)}
\text{Q} \sqcup_{\Sigma} \text{R} \quad | \quad \text{-- Lock-step concurrency (same as } \text{Q} \sqcup \text{R) }
\text{Q} \setminus \text{E} \quad | \quad \text{-- Hiding}
\mu Q \cdot f (Q) \quad \text{-- Recursion}
Recursion

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

This is an abbreviation for

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

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

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

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

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

\[ \text{COPY3} = \text{COPY1 □ COPY2} \]
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
\[ P || Q = \exists x \rightarrow \]
\[ \begin{align*}
&P || Q = \exists x \rightarrow \\
&\quad (P' || Q') \quad \text{if } x \in A \cap B \\
&\quad \text{STOP} \quad \text{otherwise}
\end{align*} \]

(to be generalized soon)
More examples

\[ VMC = \]

\[ (in2f \rightarrow \]

\[ ((\text{large} \rightarrow VMC) \square \]

\[ \square \]

\[ (\text{small} \rightarrow \text{out1f} \rightarrow VMC)) \]

\[ \square \]

\[ (in1f \rightarrow \]

\[ ((\text{small} \rightarrow VMC) \square \]

\[ \square \]

\[ (in1f \rightarrow \text{large} \rightarrow VMC)) \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]

\[ \square \]
Hiding

Consider

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

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

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

More generally:

\[(a \rightarrow P) \setminus E = \]

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

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

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

Consider
\[ P = \text{?x: } A \rightarrow P' \]
\[ Q = \text{?x: } B \rightarrow Q' \]

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

Lock-step concurrency:

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

Interleaving:

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

Consider

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

Then

\[ P || Q = ?x \rightarrow \]
\[ (P' || Q') \quad \text{if} \quad x \in E \cap A \cap B \]
\[ \text{STOP} \quad \text{otherwise} \]
Laws of non-deterministic internal choice

\[ P \sqcup P = P \]
\[ P \sqcup Q = Q \sqcup P \]
\[ P \sqcup (Q \sqcup R) = (P \sqcup Q) \sqcup R \]
\[ x \to (P \sqcup Q) = (x \to P) \sqcup (x \to Q) \]

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

The recursion operator is not distributive; consider:

\[ P = \mu X \bullet ((a \to X) \sqcup (b \to X)) \]
\[ Q = (\mu X \bullet (a \to X)) \sqcup (\mu X \bullet (b \to X)) \]
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

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

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

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

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

\[ P ::= \]

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

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

\[ \text{traces (Q)} \subseteq \text{traces (P)} \]

For example:

\[ P \quad \text{refines} \quad P \sqcap Q \]
The trace model is not enough

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

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 \textit{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 $P$ \textit{after} $t$:  

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

Compare

- $P = a \rightarrow \text{STOP} \, \Box \, b \rightarrow \text{STOP}$
- $Q = a \rightarrow \text{STOP} \, \Pi \, b \rightarrow \text{STOP}$

Same traces, but:

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

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

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

\[ \sum = \{\} \]

\[ \sum = \{\} \]

\[ \sum = \{\} \]

\[ \sum = \{\} \]

\[ \sum = \{\} \]
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

\[ P ::= \]

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

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

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

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

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

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

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

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

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

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

\[
\begin{align*}
P || Q & \equiv_M Q || P \\
(P || Q) || R & \equiv_M P || (Q || R) \\
P || (Q \cap R) & \equiv_M (P || Q) \cap (P || R) \\
(a \rightarrow P) || (b \rightarrow Q) & \equiv_M (a \rightarrow (P || (b \rightarrow Q))) \quad \Box (b \rightarrow ((a \rightarrow P) || Q)) \\
P ; (Q ; R) & \equiv_M (P ; Q) ; R \\
STOP || Q & \equiv_M Q \\
SKIP ; Q & \equiv_M Q \\
STOP ; Q & \equiv_M STOP \\
P ; (Q \cap R) & \equiv_M (P ; Q) \cap (P ; R) \\
(P \cap Q) ; R & \equiv_M (P ; R) \cap (Q ; R) \\
(a \rightarrow P) ; Q & \equiv_M (a \rightarrow P ; Q) \quad \text{if } a \neq \sqrt{} \\
(P \setminus a) \setminus b & \equiv_M (P \setminus b) \setminus a \\
(P \setminus a) \setminus a & \equiv_M P \setminus a \\
(a \rightarrow P) \setminus b & \equiv_M (a \rightarrow P \setminus b) \quad \text{if } a \neq b \\
& \equiv_M P \setminus b \quad \text{if } a = b \\
(P \cap Q) \setminus a & \equiv_M (P \setminus a) \cap (Q \setminus a)
\end{align*}
\]
Some extensions

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

Timed:
- \( P \xrightarrow{\dagger} Q \): interrupt
- \( P \xrightarrow{\dagger} Q \): timeout
- \( a \xrightarrow{\dagger} P \): communicate immediately
- \( \text{WAIT } \dagger: \text{ same as } \text{STOP } \xrightarrow{\dagger} \text{SKIP} \)
Example (Ouaknine)

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

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

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