Concurrent Object-Oriented Programming

Bertrand Meyer, Sebastian Nanz

Lecture 11: 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
\[\text{<coin, coffee, coin, coffee>}\]

Many traces of interest are infinite, for example
\[\text{<coin, coffee, coin, coffee, ...>}\]

(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

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

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

\{<>\}
Basic CSP syntax

\[ P ::= \]

\[
\text{STOP} \quad | \quad \text{-- Does not engage in any events} \\
\text{a} \rightarrow \text{Q} \quad | \quad \text{-- Engages in a, then acts like Q} \\
\text{Q} \parallel \text{R} \quad | \quad \text{-- Internal choice} \\
\text{Q} \boxdot \text{R} \quad | \quad \text{-- External choice} \\
\text{Q} \parallel|\text{R}_E \quad | \quad \text{-- Concurrency (E: subset of alphabet)} \\
\text{Q} \parallel|\text{R} \Sigma \quad | \quad \text{-- Lock-step concurrency (same as Q} \parallel|\text{R} \Sigma \text{)} \\
\text{Q} \setminus \text{E} \quad | \quad \text{-- Hiding} \\
\mu\text{Q} \cdot f(\text{Q}) \quad \text{-- Recursion} 
\]
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
$$? x : E \rightarrow P(x)$$
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. \( (c \rightarrow P) \ || \ (d \rightarrow Q) = STOP \) \quad -- \text{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:
\[ CVM = (\text{coin} \rightarrow \text{coffee} \rightarrow \text{coin} \rightarrow \text{coffee} \rightarrow \text{STOP}) \]

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

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

\[ P ::= \]

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

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

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

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

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

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

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

\[ \mu Q \cdot f(Q) \quad \text{-- Recursion} \]
Recursion

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

This is an abbreviation for
\[ CLOCK = \mu P \cdot (\text{tick} \rightarrow P) \]

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

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

COPYBIT = (in.0 → out.0 → COPYBIT

\[\square\]

in.1 → out.1 → COPYBIT)
External choice

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

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

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

$\text{CHM1} = (\text{in1f} \rightarrow \text{out50rp} \rightarrow \text{out20rp} \rightarrow \text{out20rp} \rightarrow \text{out10rp})$

$\text{CHM2} = (\text{in1f} \rightarrow \text{out50rp} \rightarrow \text{out50rp})$

$\text{CHM} = \text{CHM1} \Box \text{CHM2}$
Lock-step concurrency

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

Then
\[
P \parallel Q = \exists x \rightarrow
\]
\[\begin{align*}
& (P' \parallel Q') & \text{if } x \in A \cap B \\
& \text{STOP} & \text{otherwise}
\end{align*}
\]

(to be generalized soon)
More examples

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

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

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

Consider

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

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

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

More generally:

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

\[
\begin{align*}
& P \setminus E \quad \text{if } a \in E \\
& a \rightarrow (P \setminus E) \quad \text{if } a \notin E
\end{align*}
\]
Hiding introduces internal non-determinism

Consider

\[ R = (a \to P) \boxtimes (b \to Q) \]

Then

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

\[ CH1F = (\text{in1f} \rightarrow \\
((\text{out20rp} \rightarrow \text{out20rp} \rightarrow \\
\text{out20rp} \rightarrow \text{out20rp} \rightarrow \text{CH1F}) \\
\Pi \\
(\text{out50rp} \rightarrow \text{out50rp} \rightarrow \text{CH1F}))) \]
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

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

Then

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

Lock-step concurrency:

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

Interleaving:

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

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

Then
\[ P \parallel Q = \exists x \rightarrow \]
\[ \begin{array}{ll}
(P' \parallel Q') & \text{if } x \in E \cap A \cap B \\
\text{STOP} & \text{otherwise}
\end{array} \]
Laws of non-deterministic internal choice

\( P \Pi P = P \)
\( P \Pi Q = Q \Pi P \)
\( P \Pi (Q \Pi R) = (P \Pi Q) \Pi R \)
\( x \rightarrow (P \Pi Q) = (x \rightarrow P) \Pi (x \rightarrow Q) \)

\( P \parallel (Q \Pi R) = (P \parallel Q) \Pi (P \parallel R) \)
\( (P \Pi Q) \parallel R = (P \parallel R) \Pi (Q \parallel R) \)

The recursion operator is not distributive; consider:

\( P = \mu X \cdot ((a \rightarrow X) \Pi (b \rightarrow X)) \)
\( Q = (\mu X \cdot (a \rightarrow X)) \Pi (\mu X \cdot (b \rightarrow X)) \)
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 \(\square\)

The conjectured property is

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

It does not hold, since

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

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

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

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

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

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

\[ P ::= \]

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

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

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

For example:

$P$ refines $P \perp Q$
The trace model is not enough

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

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 $es \in P(\Sigma)$ 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 $t$:

$$\text{traces} (P / t) = \{ u | t + u \in \text{traces} (p) \}$$
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:

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

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

\[ \{ \Sigma \} \]

\[ \{ \Sigma \} \]

\[ \{ \Sigma \} \]

\[ \{ \Sigma \} \]

\[ \{ \Sigma \} \]

\[ \{ \Sigma \} \]

\[ \{ \Sigma \} \]

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

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

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

\[
(\mu p.a \rightarrow p) \setminus a
\]
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