Concurrent Object-Oriented Programming

Prof. Dr. Bertrand Meyer

Lecture 11: An introduction to 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

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

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

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

\[ P ::= \]

- \textit{STOP} \hspace{1em} | \hspace{1em} -- Does not engage in any events
- \( a \rightarrow Q \) \hspace{1em} | \hspace{1em} -- Engages in \( a \), then acts like \( P \)
- \( Q \land R \) \hspace{1em} | \hspace{1em} -- Internal choice
- \( Q \lor R \) \hspace{1em} | \hspace{1em} -- External choice
- \( Q \parallel R \) \hspace{1em} | \hspace{1em} -- Concurrency \((E: \text{subset of alphabet})\)
- \( Q \parallel R \) \hspace{1em} | \hspace{1em} -- Lock-step concurrency (same as \( P \parallel_P P \))
- \( Q \setminus E \) \hspace{1em} | \hspace{1em} -- Hiding
- \( \mu Q \cdot f (Q) \) \hspace{1em} -- 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}, \text{coffee}, \text{coin}, \text{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

\[ P ::= \]

\[
\begin{align*}
\text{STOP} & \quad | \quad \text{Does not engage in any events} \\
\alpha \rightarrow Q & \quad | \quad \text{Engages in } \alpha, \text{ then acts like } P \\
Q \parallel R & \quad | \quad \text{Internal choice} \\
Q \Box R & \quad | \quad \text{External choice} \\
Q \parallel E \parallel R & \quad | \quad \text{Concurrency (} E \text{: subset of alphabet)} \\
Q \parallel R & \quad | \quad \text{Lock-step concurrency (same as } P \parallel P) \\
Q \setminus E & \quad | \quad \text{Hiding} \\
\mu Q \cdot f (Q) & \quad | \quad \text{Recursion}
\end{align*}
\]
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:
\[ ? \ y: c.A \rightarrow d.y' \]
(\( c.A \) 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

\[ \text{COPYBIT} = (\text{in.0} \rightarrow \text{out.0} \rightarrow \text{COPYBIT}) \]

\[ \text{COPYBIT} = (\text{in.0} \rightarrow \text{out.0} \rightarrow \text{COPYBIT}) \]

\[ \text{COPYBIT} = (\text{in.1} \rightarrow \text{out.1} \rightarrow \text{COPYBIT}) \]
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

$$CHM_1 = (in1f \rightarrow out50rp \rightarrow out20rp \rightarrow out20rp \rightarrow out10rp)$$
$$CHM_2 = (in1f \rightarrow out50rp \rightarrow out50rp)$$

$$CHM = CHM_1 \square CHM_2$$
More examples

\[ VMC = (in2f \rightarrow ((large \rightarrow VMC) \square (small \rightarrow out1f \rightarrow VMC))) \]

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

\[ FV = FOOLCUST || VMC = \mu P \bullet (in2f \rightarrow large \rightarrow FV \square in2f \rightarrow STOP) \]
Hiding

Consider

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

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

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

More generally:

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

Consider

\[ R = (a \rightarrow P) \square (b \rightarrow Q) \]

Then

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

\[ CH1F = (\text{in1f} \rightarrow \\
\quad ((\text{out20rp} \rightarrow \text{out20rp} \rightarrow \\
\quad \quad \text{out20rp} \rightarrow \text{out20rp} \rightarrow \text{out20rp} \rightarrow CH1F)) \\
\quad \Pi \\
\quad (\text{out50rp} \rightarrow \text{out50rp} \rightarrow CH1F)) \]
Non-deterministic internal choice: another application

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

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

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

Then
\[ P || Q = \ ?x \rightarrow \]
\[ \left\{ \begin{array}{ll}
(P' || Q') & \text{if } x \in E \cap A \cap B \\
(P' || Q) & \text{if } x \in A-B-E \\
(P || Q') & \text{if } x \in B-A-E \\
(P' || Q) \cap (P || Q') & \text{if } x \in (A \cap B) - E 
\end{array} \right. \]
Special cases of concurrency

Lock-step concurrency:

\[
P \parallel Q = \parallel \frac{Q}{\sum} \parallel Q
\]

Interleaving:

\[
P \parallel\parallel Q = \parallel \frac{Q}{\varnothing} \parallel Q
\]
Laws of non-deterministic internal choice

\[
\begin{align*}
P \land P &= P \\
P \land Q &= Q \land P \\
P \land (Q \land R) &= (P \land Q) \land R \\
x \to (P \land Q) &= (x \to P) \land (x \to Q) \\
\end{align*}
\]

\[
\begin{align*}
P \lor (Q \land R) &= (P \lor Q) \land (P \lor R) \\
(P \land Q) \lor R &= (P \lor R) \land (Q \lor R) \\
\end{align*}
\]

The recursion operator is not distributive; consider:

\[
\begin{align*}
P &= \mu X \bullet ((a \to X) \land (b \to X)) \\
Q &= (\mu X \bullet (a \to X)) \land (\mu X \bullet (b \to X)) \\
\end{align*}
\]
Refinement

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

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

For example:

$$P \text{ refines } P \cap Q$$
Traces

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

\[ P ::= \]

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

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

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

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

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

\[
Q \mathrel{||} R \quad | \quad -- \text{Lock-step concurrency (same as} \ P \mathrel{||}_\Sigma P)\]

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

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

\( P = a \rightarrow b \rightarrow \text{STOP} \)
\( Q = b \rightarrow a \rightarrow \text{STOP} \)

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 $es \subseteq P(\Sigma)$ is a 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 $P$ after $t$:

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

Compare

- \( P = a \rightarrow \text{STOP} \quad \square \quad b \rightarrow \text{STOP} \)
- \( Q = a \rightarrow \text{STOP} \quad \sqcap \quad b \rightarrow \text{STOP} \)

Same traces, but:

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

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

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

\[ \text{a} \rightarrow \text{STOP} \quad \tau \quad \text{b} \rightarrow \text{STOP} \]
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 □ 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 \( \tau \) transitions.

Another 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