CSP

Guarded Commands
- Monitor: begin executing every call as soon as possible, waiting if the object is not in a proper state and signaling when the state is proper
- CSP: the called object establishes conditions under which the call is accepted; calls not satisfying these conditions are held pending (no need for programmed wait/signal operations).

Rendezvous
- Monitor: the monitor is passive (has no independent task/thread/activity)
- CSP: synchronization between peer, autonomous activities.

CSP

Distribution:
- Monitor: inherently nondistributed in outlook and implementation
- CSP: possibility for distributed programming using synchronous message passing
Communicating Sequential Processes (CSP)

- single thread of control
- autonomous
- encapsulated
- named
- static
- synchronous
- reliable
- unidirectional
- point-to-point
- fixed topology

operators:

\(!\) (send)

? (receive)

usage:

Send to

\[ \text{message} \]

receive from

buffer

A

B

A!x

B!x

x

y

A?y

CS 5204 Spring 99
Communicating Sequential Processes (CSP)

- rendezvous semantics: senders (receivers) remain blocked at send (receive) operation until a matching receive (send) operation is made.
- typed messages: the type of the message sent by the sender and the type of the message expected by the receiver must match (otherwise abort).

\[
\begin{align*}
\text{A!vec(x,y)} & \quad \text{B?vec(s,t)} \\
\text{OK} \\
\text{A!count(x)} & \quad \text{B?index(y)} \\
\text{NO}
\end{align*}
\]

Guarded Commands

\[
\langle \text{guard} \rangle \Rightarrow \langle \text{command list} \rangle
\]

- boolean expression
- only one ?, must be at end of guard, considered true iff message pending

Examples

\[
\begin{align*}
n < 10 \Rightarrow & \quad \text{A!index(n)}; \quad n := n + 1; \\
n < 10; \quad & \quad \text{A?index(n)} \Rightarrow \text{next = MyArray(n)};
\end{align*}
\]
Communicating Sequential Processes (CSP)

Alternative Command

\[
\begin{align*}
& [ \text{G}_1 \Rightarrow \text{S}_1 ] \ [ \text{G}_2 \Rightarrow \text{S}_2 ] \ldots [ \text{G}_n \Rightarrow \text{S}_n ] \\
& 1. \text{evaluate all guards} \\
& 2. \text{if more than one guard is true, nondeterministically select one.} \\
& 3. \text{if no guard is true, terminate.} \\
& \text{Note: if all true guards end with an input command for which there is no pending message, then delay the evaluation until a message arrives. If all senders have terminated, then the alternative command terminates.}
\end{align*}
\]

Repetitive Command

\[
\begin{align*}
& * [ \text{G}_1 \Rightarrow \text{S}_1 ] \ [ \text{G}_2 \Rightarrow \text{S}_2 ] \ldots [ \text{G}_n \Rightarrow \text{S}_n ] \\
& \text{repeatedly execute the alternative command until it terminates}
\end{align*}
\]

Examples:

- \[
\begin{align*}
& [ x \geq y \Rightarrow m := x ] \ [ y \geq x \Rightarrow m := y ] \\
& i := 0; * [ i < \text{size}; \text{content}(i) \neq n \Rightarrow i := i + 1 ] \\
& * [ c: \text{character}; \text{west?}c \Rightarrow \text{east!}c ] \\
& * [ n: \text{integer}; X?\text{insert}(n) \Rightarrow \text{INSERT} ] \\
& \ [ \text{search}(n) \Rightarrow \text{SEARCH}; X!(i < \text{size}) ]
\end{align*}
\]

BoundedBuffer::

\[
\begin{align*}
& \text{buffer: (0..9) portion; } \\
& \text{in, out: integer; in := 0; out := 0; } \\
& * [ \text{in < out + 10; producer?buffer(in mod 10)} \\
& \quad \Rightarrow \text{in := in + 1; } ] \\
& \ [ \text{out < in; consumer?more()} \\
& \quad \Rightarrow \text{consumer!buffer(out mod 10); } \\
& \quad \text{out := out + 1; } ]
\end{align*}
\]
ADA Example

task bounded-buffer is
  entry store(x : buffer);
  entry remove(y: buffer);
end;
task body bounded-buffer is
  ...declarations...
begin
  loop
    select
      when head < tail + 10 =>
        accept store(x : buffer) ... end store;
    or
      when tail < head =>
        accept remove(y: buffer) ... end remove;
    end select;
  end loop
end