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 non-distributed 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:

? (receive)

! (send)

usage:

Send to
B!x
message

receive from
A?y
buffer

A
B

B!x
x

A?y
y
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).

A!vec(x,y)  B?vec(s,t)

\[ \text{OK} \]

A!count(x)  B?index(y)

\[ \text{NO} \]

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

\[ n < 10 \rightarrow A!index(n); \ n := n + 1; \]
\[ n < 10; A?index(n) \rightarrow \text{next} = \text{MyArray}(n); \]

Communicating Sequential Processes (CSP)

Alternative Command

\[
[ \text{G}_1 \rightarrow \text{S}_1 \] [ \text{G}_2 \rightarrow \text{S}_2 \] \ldots [ \text{G}_n \rightarrow \text{S}_n ]
\]

1. evaluate all guards
2. if more than one guard is true, nondeterministically select one.
3. if no guard is true, terminate.

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.

Repetitive Command

\* \[
[ \text{G}_1 \rightarrow \text{S}_1 \] [ \text{G}_2 \rightarrow \text{S}_2 \] \ldots [ \text{G}_n \rightarrow \text{S}_n ]
\]

repeatedly execute the alternative command until it terminates

Examples:

\[
[x \geq y \rightarrow m := x \] [ y \geq x \rightarrow m := y ]
\]

\[i := 0; \* [ i < \text{size}; \text{content}(i) \neq n \rightarrow i := i + 1 ]\]

\* \[
[ c : \text{character}; \text{west?}c \rightarrow \text{east!}c ]
\]

\* \[
[ n : \text{integer}; \text{X?}\text{insert}(n) \rightarrow \text{INSERT} \]
\]

\[
\text{BoundedBuffer::}
\]

\[
\text{buffer: } (0..9) \text{ portion;}
\]

\[
\text{in, out : integer; in := 0; out := 0;}
\]

\* \[
[ \text{in < out + 10; \text{producer?buffer} (\text{in mod 10})}
\rightarrow \text{in := in + 1;}
\]

\[
\text{[]}
\]

\[
\text{out < in; \text{consumer?}more()}
\rightarrow \text{consumer!buffer (out mod 10);}
\]

\[
\text{out := out + 1;}
\]

\]
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