CSP and ADA

Guarded Commands

• Monitor/Serializer: 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/Ada: 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/Serializer: the monitor/ synchronizer is passive (has no independent task/thread/activity)
• CSP/Ada: synchronization between peer, autonomous activities.
CSP and ADA

Distribution:

- Monitor/Serializer: inherently non-distributed in outlook and implementation
- CSP/Ada: 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
Communicating Sequential Processes (CSP)

operators:

? (receive)

! (send)

usage:

Send to

A!x

receive from

B?y

message

buffer

A

A!x

x

B

y

B?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).

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

Guarded Commands

<guard> --> <command list>

boolean expression

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

Examples

n < 10 --> A!index(n); n := n + 1;
n < 10; A?index(n) --> next = A(n);
Communicating Sequential Processes (CSP)

Alternative Command

\[
[ \text{G1} \rightarrow \text{S1} \] \ [ \text{G2} \rightarrow \text{S2} \] \ldots \ [ \text{Gn} \rightarrow \text{Sn} ]
\]

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

\* \[
[ \text{G1} \rightarrow \text{S1} \] \ [ \text{G2} \rightarrow \text{S2} \] \ldots \ [ \text{Gn} \rightarrow \text{Sn} ]
\]

Repeatedly execute the alternative command until it terminates.
Communicating Sequential Processes (CSP)

Examples:

\[
[x \geq y \rightarrow m := x \quad \quad y \geq x \rightarrow m ; + y]
\]
\[
i := 0; \quad * \quad [i < \text{size}; \quad \text{content}(i) \neq n \rightarrow i := i + 1]
\]
\[
* \quad [\quad c : \text{character}; \quad \text{west}?c \rightarrow \text{east}!c]
\]
\[
* \quad [\quad n : \text{integer}; \quad \text{X}!\text{insert}(n) \rightarrow \text{INSERT}
\]
\[
[]
\]
\[
n : \text{integer}; \quad \text{X}?\text{has}(n) \rightarrow \text{SEARCH}; \quad \text{X}!(i < \text{size})
\]

BoundedBuffer::

buffer: (0..9) portion;
in, out : integer; in := 0; out := 0;
* [ in < out + 10; \quad \text{producer}?\text{buffer}(\text{in} \mod 10)
   \rightarrow \text{in} := \text{in} + 1;
\]

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