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

operators:
! (send)
?
(receive)

usage:
Send to
A!x
message

receive from
B?y
buffer
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).

\[\text{A}!\text{vec}(x,y) \quad \text{B}?\text{vec}(s,t)\]

\[\text{A}!\text{count}(x) \quad \text{B}?\text{index}(y)\]

\[\text{A}!\text{vec}(x,y) \quad \text{B}?\text{vec}(s,t)\]

\[\text{A}!\text{count}(x) \quad \text{B}?\text{index}(y)\]

Communicating Sequential Processes (CSP)

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

\[n < 10 \rightarrow A!\text{index}(n); \quad n := n + 1;\]
\[n < 10; \quad A?\text{index}(n) \rightarrow \text{next} = A(n);\]
Communicating Sequential Processes (CSP)

Alternative Command

\[ [\ G_1 \ --\ > \ S_1 ] [\ G_2 \ --\ > \ S_2 ] \ldots [\ G_n \ --\ > \ S_n ] \]

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

\* [\ G_1 \ --\ > \ S_1 ] [\ G_2 \ --\ > \ S_2 ] \ldots [\ G_n \ --\ > \ 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: character; west?c \rightarrow east!c]

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

\[ n: \text{integer}; \text{X?has}(n) \rightarrow \text{SEARCH}; \text{X!}(i < \text{size}) \]

BoundedBuffer::

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

\[ \]

out < in; \text{consumer?more()}
  \rightarrow \text{consumer!buffer}(\text{out mod 10});
  \text{out} := \text{out} + 1;
\]
ADA Example

task bounded-buffer is
  entry store(x : buffer);
  entry remove(y: buffer);
end;
task body bounded-buffer is
  ...declarations...
begins
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