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.

---

**Communicating Sequential Processes (CSP)**

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

---

**Communicating Sequential Processes (CSP)**

Guarded Commands

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

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

**Examples**

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

Alternative Command

\[ G_1 \rightarrow S_1 ] \ G_2 \rightarrow S_2 ] \ldots [ \ G_n \rightarrow S_n ] \]

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

\[ \ast \ [ G_1 \rightarrow S_1 ] \ G_2 \rightarrow S_2 ] \ldots [ \ G_n \rightarrow S_n ] \]

repeatedly execute the alternative command until it terminates

Examples:

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

1 := 0; \ast \ [ i < size; content[i] := n \rightarrow i := i + 1 ]

\[ c: \text{character}; \operatorname{west}(c) \rightarrow \operatorname{east}(c) \]

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

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

BoundedBuffer::

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

ADA Example

\text{task bounded-buffer is}

\text{entry store(x : buffer);} \\
\text{entry remove(y: buffer);} \\
\text{end;} \\
\text{task body bounded-buffer is}

\text{...declarations...}

\text{begin}

\text{loop}

\text{select}

\text{when head < tail + 10 =>}

\text{accept store(x : buffer) ... end store;}

\text{or}

\text{when tail < head =>}

\text{accept remove(y: buffer) ... end remove;}

\text{end select;}

\text{end loop}

\text{end}