CS 515: Communicating Sequential Processes (CSP)

Stefan D. Bruda

Winter 2019

Events and Processes

- Process a self-contained entity that interacts with its environment through events or actions from an interface (set of events)
- Events are atomic entities; we abstract over their input of output nature
- To describe CSP we need to provide its syntax (what are the acceptable constructs) and a semantics (when is the behaviour of the constructs)
- Multiple ways of giving the semantics, grouped into two categories
  - Operational semantics describes how the system runs
  - Denotational semantics describes the system without explicitly describing its execution
  - Example (from formal languages): Regular expressions (denotational) vs. finite automata (operational)

(Labelled) Transition Systems

- Transition system: directed graph with labeled edges (and one designated root vertex)
- To execute a CSP process we walk the respective transition system
- Several actions may be available from any given state (the labels of the outgoing edges) $\Rightarrow$ process interface
- A process is at any given time in a (named) state (vertex in the graph)
- Once an action is performed (an edge is traversed) the process changes its state (to the end vertex of the respective edge)
- More formally a transition (edge in the graph) $P_1 \xrightarrow{\mu} P_2$ means that the process P_1 is able to execute action μ and if it does so then it will become p_2
  - Let Σ be the set of all the possible actions in a transition system
  - Special actions τ (internal action) and $\checkmark$ (termination)
  - Put $\Sigma^{\checkmark} = \Sigma \cup \{\checkmark\}$
  - μ ranges over $\Sigma \cup \{\tau, \checkmark\}$

Structural Operational Semantics (SOS)

- Convenient way of describing the transition system of a CSP process based on the structure of that process
  - "Less operational" than the transition system itself but "more structural"
- Based on inference rules (also SOS rules or just rules)

$$\frac{\begin{array}{c}\text{fact } 1\\ \cdots\\ \text{fact } n\end{array}}{\begin{array}{c}\text{conclusion } 1\\ \cdots\\ \text{conclusion } m\end{array}} \ [\text{side condition}]$$

  - If the antecedent facts (1 to n) are true and the side condition is also true, then it can be inferred that all the conclusions (1 to m) are true
  - The side condition is optional
  - If no facts are specified then the conclusions are a priori true
- Special SOS ternary relation: $P_1 \xrightarrow{\mu} P_2$ is true iff there is a transition labeled μ between P_1 and P_2
  - This relation is defined axiomatically using inference rules

CSP Baic Processes and Operators

- One basic process $STOP \Rightarrow$ never performs any action
  - No operational semantics (there are no transitions)
- Event prefix: if P is a CSP process and a an action then $a \to P$ is also a CSP process
  - Operational semantics:

$$\frac{}{(a \to P) \xrightarrow{a} P}$$

- (Menu) prefix choice: for $A \subseteq \Sigma$ and $P(a)$ CSP processes, $a \in A$, the following is a CSP process: $x : A \to P(x)$
  - Operational semantics:

$$\frac{}{(x : A \to P(x)) \xrightarrow{a} P(a)} \; [a \in A]$$

  - Syntactic sugar: $x : \{a_1, \ldots, a_n\} \to P(x)$ is often written

$$\begin{array}{l} a_1 \to P(a_1) \\ \mid \; a_2 \to P(a_2) \\ \ldots \\ \mid \; a_n \to P(a_n) \end{array}$$

Input and Output, Termination

- A communication channel has a name c and a type T (values that can be passed along c)
  - The set of events performed through the channel is $c.T = \{c.t : t \in T\}$
- Output to a channel: c!v→P with c a channel of type T and $v \in T$

$$\frac{}{(c!v \to P) \xrightarrow{c.v} P}$$

- Input from a channel: $c?x : T \to P(x)$

$$\frac{}{(c?x : T \to P(x)) \xrightarrow{c.v} P(v)} \; [v \in T]$$

- Successful termination is defined by the special action ✓ and the special process $SKIP$:

$$\frac{}{SKIP \xrightarrow{\checkmark} STOP}$$

Recursion

- Recursion happens whenever a process N is defined as $N = P$ where P is an expression containing N
- Often written explicitly $N = F(N)$
  - Substitution: $P_1[P_2/N]$ is P_1 with all the free instances of N replaced by P_2
  - If $N = P$ is a recursive definition then $N = F(N)$ is the same definition, where $F(Y) = P[Y/N]$
- Semantics given by the following unwinding rule:

$$\frac{P \xrightarrow{\mu} P'}{N \xrightarrow{\mu} P'} \; [N = P]$$

- Mutual recursion: two (or more) processes defined recursively in terms of each other
  - Same operational semantics as simple recursion

External Choice

- External choice is a choice between processes that is settled in conjunction with the environment: $P_1 \,\square\, P_2$

$$\frac{P \xrightarrow{a} P'}{\begin{array}{l} P \,\square\, Q \xrightarrow{a} P' \\ Q \,\square\, P \xrightarrow{a} P' \end{array}} \; [a \neq \tau] \qquad \frac{P \xrightarrow{\tau} P'}{\begin{array}{l} P \,\square\, Q \xrightarrow{\tau} P' \,\square\, Q \\ Q \,\square\, P \xrightarrow{\tau} Q \,\square\, P' \end{array}}$$

- Generalizes to indexed external choice $\square_{i \in I} P_i$

$$\frac{P_j \xrightarrow{a} P'}{\square_{i \in I} P_i \xrightarrow{a} P'} \; [j \in I, a \neq \tau]$$

$$\frac{P_j \xrightarrow{\tau} P'_j}{\square_{i \in I} Q \xrightarrow{\tau} \square_{i \in I} P'_i \text{ with } i \neq j \implies P'_i = P_i} \; [j \in I]$$

  - Can be defined in terms of external choice as follows: $\square_{j \in \{i\}} P_j = P_i$, $\square_{j \in I \cup \{i\}} = (\square_{j \in I}) \,\square\, P_i$

Internal Choice

- Internal choice is a choice between processes that is made independently of the environment: $P_1 \sqcap P_2$

$$\frac{}{\begin{array}{l} P \sqcap Q \xrightarrow{\tau} P \\ P \sqcap Q \xrightarrow{\tau} Q \end{array}}$$

- Generalizes to indexed internal choice $\sqcap_{i \in I} P_i$

$$\frac{}{\sqcap_{i \in I} P_i \xrightarrow{\tau} P_j} \quad [j \in I]$$

Alphabetized Parallel

- A concurrent process $P \ {}_A\|_B \ Q$ consists of two processes (P and Q) whose interface is restricted to A and B, respectively and which synchronize over common actions ($A \cap B$):

$$\frac{\begin{array}{c} P \xrightarrow{a} P' \\ \hline P \ {}_A\|_B \ Q \xrightarrow{a} P' \ {}_A\|_B \ Q \\ Q \ {}_B\|_A \ P \xrightarrow{a} Q \ {}_B\|_A \ P' \end{array}}{} \quad [a \in A \setminus B \cup \{\tau\}]$$

$$\frac{\begin{array}{c} P \xrightarrow{a} P' \\ Q \xrightarrow{a} Q' \\ \hline P \ {}_A\|_B \ Q \xrightarrow{a} P' \ {}_A\|_B \ Q' \end{array}}{} \quad [a \in A^{\checkmark} \cap B^{\checkmark}]$$

- $P \parallel Q$ is short for $P \ {}_{\alpha(P)}\|_{\alpha(Q)} \ Q$ (with $\alpha(X)$ = the interface of X)
- Generalization: indexed alphabetized parallel $\|_{A_i}^{i \in I} P_i$ defined inductively:

$$\|_{A_i}^{i \in \{i_1, i_2\}} P_i = P_1 \ {}_{A_1}\|_{A_2} \ P_2 \qquad \|_{A_i}^{i \in I \cup \{j\}} P_i = \left(\|_{A_i}^{i \in I} P_i\right) \ {}_{\cup_{i \in I} A_i}\|_{A_j} \ P_j$$

Examples

- Pipelining

$$\begin{array}{rcl} G & = & in?x : \mathbb{Z} \rightarrow com!g(x) \rightarrow G \\ F & = & com?y : \mathbb{Z} \rightarrow out!f(y) \rightarrow F \\ PIPE & = & G \ {}_{in.\mathbb{Z} \cup com.\mathbb{Z}}\|_{com.\mathbb{Z} \cup out.\mathbb{Z}} \ F \end{array}$$

- Kids painting

$$\begin{array}{rcl} ISABELLA & = & isabella.get.box \rightarrow isabella.get.easel \rightarrow isabella.paint \rightarrow \\ & & isabella.drop.box \rightarrow isabella.drop.easel \rightarrow ISABELLA \\ & \Box & isabella.get.easel \rightarrow isabella.get.box \rightarrow isabella.paint \rightarrow \\ & & isabella.drop.box \rightarrow isabella.drop.easel \rightarrow ISABELLA \\ KATE & = & kate.get.box \rightarrow kate.get.easel \rightarrow kate.paint \rightarrow \\ & & kate.drop.box \rightarrow kate.drop.easel \rightarrow KATE \\ & \Box & kate.get.easel \rightarrow kate.get.box \rightarrow kate.paint \rightarrow \\ & & kate.drop.box \rightarrow kate.drop.easel \rightarrow KATE \\ EASEL & = & isabella.get.easel \rightarrow isabella.drop.easel \rightarrow EASEL \\ & \Box & kate.get.easel \rightarrow kate.drop.easel \rightarrow EASEL \\ BOX & = & isabella.get.box \rightarrow isabella.drop.box \rightarrow BOX \\ & \Box & kate.get.box \rightarrow kate.drop.box \rightarrow BOX \end{array}$$

$$PAINTING = ISABELLA \parallel KATE \parallel EASEL \parallel BOX$$

Example: Kids Painting (cont'd)

$$\begin{array}{rcl} I & = & i.g.b \overset{\boxed{I1}}{\Rightarrow} i.g.e \overset{\boxed{I2}}{\Rightarrow} i.p \overset{\boxed{I3}}{\Rightarrow} i.d.b \overset{\boxed{I4}}{\Rightarrow} i.d.e \rightarrow I \\ & \Box & i.g.e \overset{\boxed{I5}}{\Rightarrow} i.g.b \overset{\boxed{I6}}{\Rightarrow} i.p \overset{\boxed{I7}}{\Rightarrow} i.d.b \overset{\boxed{I8}}{\Rightarrow} i.d.e \rightarrow I \\ K & = & k.g.b \overset{\boxed{K1}}{\Rightarrow} k.g.e \overset{\boxed{K2}}{\Rightarrow} k.p \overset{\boxed{K3}}{\Rightarrow} k.d.b \overset{\boxed{K4}}{\Rightarrow} k.d.e \rightarrow K \\ & \Box & k.g.e \overset{\boxed{K5}}{\Rightarrow} k.g.b \overset{\boxed{K6}}{\Rightarrow} k.p \overset{\boxed{K7}}{\Rightarrow} k.d.b \overset{\boxed{K8}}{\Rightarrow} k.d.e \rightarrow K \\ E & = & i.g.e \overset{\boxed{E1}}{\Rightarrow} i.d.e \overset{\boxed{\ }}{\Rightarrow} \Box \quad k.g.e \overset{\boxed{E2}}{\Rightarrow} k.d.e \rightarrow E \\ B & = & i.g.b \overset{\boxed{B1}}{\Rightarrow} i.d.b \rightarrow B \quad \Box \quad k.g.b \overset{\boxed{B2}}{\Rightarrow} k.d.b \rightarrow B \end{array}$$

$$PAINTING = I \parallel K \parallel E \parallel B$$

- The process $I1 \parallel K5 \parallel B1 \parallel E2$ is an example of deadlock
- Two (or more) processes are deadlocked whenever they are able perform actions individually in isolation from each other, but no action is possible when they are combined together in a parallel composition

Dining Philosophers

- Five philosophers sit at a round table with a bowl of rice in the middle and five chopsticks (one in between each philosopher)
- In order to eat a philosopher must acquire both the adjacent chopsticks

$$
\begin{aligned}
\mathrm{PHIL}_i &= \mathrm{enter}.i \rightarrow \\
&\quad ((\mathrm{pick}.i.i \rightarrow \mathrm{pick}.i.((i+1) \bmod 5) \rightarrow \mathrm{eat}.i \\
&\quad\quad \rightarrow \mathrm{put}.i.i \rightarrow \mathrm{put}.i.((i+1) \bmod 5) \rightarrow \mathrm{leave}.i \rightarrow \mathrm{PHIL}_i) \\
&\quad \Box \\
&\quad (\mathrm{pick}.i.((i+1) \bmod 5) \rightarrow \mathrm{pick}.i.i \rightarrow \mathrm{eat}.i \\
&\quad\quad \rightarrow \mathrm{put}.i.((i+1) \bmod 5) \rightarrow \mathrm{put}.i.i \rightarrow \mathrm{leave}.i \rightarrow \mathrm{PHIL}_i)) \\
\mathrm{PHILS} &= \mathrm{PHIL}_0 \,||\, \mathrm{PHIL}_1 \,||\, \mathrm{PHIL}_2 \,||\, \mathrm{PHIL}_3 \,||\, \mathrm{PHIL}_4 \\
\mathrm{CHOP}_j &= \mathrm{pick}.j.j \rightarrow \mathrm{put}.j.j \rightarrow \mathrm{CHOP}_j \\
&\quad \Box\, \mathrm{pick}.((j-1) \bmod 5).j \rightarrow \mathrm{put}.((j-1) \bmod 5).j \rightarrow \mathrm{CHOP}_j \\
\mathrm{CHOPS} &= \mathrm{CHOP}_0 \,||\, \mathrm{CHOP}_1 \,||\, \mathrm{CHOP}_2 \,||\, \mathrm{CHOP}_3 \,||\, \mathrm{CHOP}_4 \\
\mathrm{COLLEGE} &= \mathrm{PHILS} \,||\, \mathrm{CHOPS}
\end{aligned}
$$

Interleaving

- The interleaving $P \,|||\, Q$ of two processes (P and Q) is their unsynchronized execution:

$$\frac{P \xrightarrow{a} P'}{\begin{array}{c} P \,|||\, Q \xrightarrow{a} P' \,|||\, Q \\ Q \,|||\, P \xrightarrow{a} Q \,|||\, P' \end{array}} \; [a \neq \checkmark] \qquad \frac{\begin{array}{c} P \xrightarrow{\checkmark} P' \\ Q \xrightarrow{\checkmark} Q' \end{array}}{P \,|||\, Q \xrightarrow{\checkmark} P' \,|||\, Q'}$$

- A model for dynamic process creation (aka `fork()`):

$$PROC = PARENT \rightarrow (PROC \,|||\, CHILD)$$

  - For example: $NODE = in?x \rightarrow (NODE \,|||\, OUT(x))$ with $OUT(x) = out!x \rightarrow SKIP$, $OUT(x) = out!x \rightarrow SKIP \,|||\, log!x \rightarrow SKIP$, etc.
- Usual generalization: $|||_{i \in I} P_i$ (indexed interleaving)

Interface Parallel

- The process $P \underset{A}{\|} Q$ runs P and Q synchronized over exactly all the events in A:

$$\frac{\begin{array}{c} P \xrightarrow{a} P' \\ Q \xrightarrow{a} Q' \end{array}}{P \underset{A}{\|} Q \xrightarrow{a} P' \underset{A}{\|} Q'} \; [a \in A^{\checkmark}] \qquad \frac{P \xrightarrow{a} P'}{\begin{array}{c} P \underset{A}{\|} Q \xrightarrow{a} P' \underset{A}{\|} Q \\ Q \underset{A}{\|} P \xrightarrow{a} Q \underset{A}{\|} P' \end{array}} \; [a \notin A^{\checkmark}]$$

- Mix of alphabetized parallel and interleaving
- Usual generalization: indexed interface parallel
- Is it true that $P \underset{A}{\|} Q = P \,{}_{\alpha(P) \cup A}\|_{\alpha(Q) \cup A}\, Q$? Why or why not?

Hiding, Renaming

- $P \setminus A$ behaves like P except that all the events from A are hidden
  - Meaning that they are replaced with τ

$$\frac{P \xrightarrow{a} P'}{P \setminus A \xrightarrow{\tau} P' \setminus A} \; [a \in A] \qquad \frac{P \xrightarrow{a} P'}{P \setminus A \xrightarrow{a} P' \setminus A} \; [a \notin A]$$

  - Purpose: limit the interface of a process only to those events that really belong to the interface (as opposed to being "internal" to the object) $\Rightarrow$ encapsulation
- Forward renaming: $f(P)$ for some function $f : \Sigma^{\checkmark} \Rightarrow \Sigma^{\checkmark}$ such that $f(a) = \checkmark$ iff $a = \checkmark$:

$$\frac{P \xrightarrow{a} P'}{f(P) \xrightarrow{f(a)} f(P')} \qquad \frac{P \xrightarrow{\tau} P'}{f(P) \xrightarrow{\tau} f(P')}$$

- Backward renaming: $f^{-1}(P)$, same as forward renaming, only... backward

$$\frac{P \xrightarrow{f(a)} P'}{f^{-1}(P) \xrightarrow{a} f^{-1}(P')} \qquad \frac{P \xrightarrow{\tau} P'}{f^{-1}(P) \xrightarrow{\tau} f^{-1}(P')}$$

- Note: $f^{-1}(f(P))$ is not necessarily the same as P (why?)

Sequential Composition and Interrupt

- Sequential composition $P; Q$ runs P and then Q:

$$\frac{P \xrightarrow{a} P'}{P; Q \xrightarrow{a} P'; Q} \; [a \neq \checkmark] \qquad \frac{P \xrightarrow{\checkmark} P'}{P; Q \xrightarrow{\tau} Q}$$

- Interrupt $P \triangle Q$ allows Q to preempt and replace P at any time:

$$\frac{P \xrightarrow{a} P'}{P \triangle Q \xrightarrow{a} P' \triangle Q} \; [a \neq \checkmark] \qquad \frac{P \xrightarrow{\checkmark} P'}{P \triangle Q \xrightarrow{\checkmark} P'}$$

$$\frac{Q \xrightarrow{\tau} Q'}{P \triangle Q \xrightarrow{\tau} P \triangle Q'} \qquad \frac{Q \xrightarrow{a} Q'}{P \triangle Q \xrightarrow{a} Q'}$$