

CS 515: Communicating Sequential Processes (CSP)

Stefan D. Bruda

Winter 2019



- **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 or 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



- **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 or 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 $\Sigma \cup \{\tau, \checkmark\}$



- 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”



- 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**)

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

- 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



- 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**)

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

- 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



- One basic process *STOP* \Rightarrow never performs any action



- 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 $a \rightarrow P$ is also a CSP process



- 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 $a \rightarrow P$ is also a CSP process
 - Operational semantics:

$$\overline{(a \rightarrow P) \xrightarrow{a} P}$$

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

$$\overline{(x : A \rightarrow P(x)) \xrightarrow{a} P(a)} \quad [a \in A]$$

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

$$\begin{array}{l} a_1 \rightarrow P(a_1) \\ | a_2 \rightarrow P(a_2) \\ \dots \\ | a_n \rightarrow P(a_n) \end{array}$$



- 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: $c!v \rightarrow P$ with c a channel of type T and $v \in T$

$$\overline{(c!v \rightarrow P) \xrightarrow{c.v} P}$$

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

$$\overline{(c?x : T \rightarrow P(x)) \xrightarrow{c.v} P(v)} \quad [v \in T]$$



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: $c!v \rightarrow P$ with c a channel of type T and $v \in T$

$$\overline{(c!v \rightarrow P) \xrightarrow{c.v} P}$$

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

$$\overline{(c?x : T \rightarrow P(x)) \xrightarrow{c.v} P(v)} \quad [v \in T]$$

- **Successful termination** is defined by the special action \checkmark and the special process **SKIP**:

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



- **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 $F(Y) = P[Y/N]$



- **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**:

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



- **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**:

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

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



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

$$\frac{P \xrightarrow{a} P'}{P \square Q \xrightarrow{a} P' \quad Q \square P \xrightarrow{a} P'} [a \neq \tau] \qquad \frac{P \xrightarrow{\tau} P'}{P \square Q \xrightarrow{\tau} P' \square Q \quad Q \square P \xrightarrow{\tau} Q \square P'}$$



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

$$\frac{P \xrightarrow{a} P'}{P \square Q \xrightarrow{a} P'} \quad [a \neq \tau] \qquad \frac{P \xrightarrow{\tau} P'}{P \square Q \xrightarrow{\tau} P' \square Q}$$

$$Q \square P \xrightarrow{a} P' \qquad Q \square P \xrightarrow{\tau} Q \square P'$$

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

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

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

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

$$\frac{}{P \sqcap Q \xrightarrow{\tau} P}$$
$$P \sqcap Q \xrightarrow{\tau} Q$$

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

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



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

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q'}{P \mathbin{A \parallel B} Q \xrightarrow{a} P' \mathbin{A \parallel B} Q'} \quad [a \in A \setminus B \cup \{\tau\}]$$

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q'}{P \mathbin{A \parallel B} Q \xrightarrow{a} P' \mathbin{A \parallel B} Q'} \quad [a \in A \cap B]$$

- $P \parallel Q$ is short for $P \mathbin{\alpha(P) \parallel \alpha(Q)} Q$ (with $\alpha(X)$ = the interface of X)



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

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q'}{P \mathbin{A \parallel B} Q \xrightarrow{a} P' \mathbin{A \parallel B} Q'} \quad [a \in A \setminus B \cup \{\tau\}]$$

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q'}{P \mathbin{A \parallel B} Q \xrightarrow{a} P' \mathbin{A \parallel B} Q'} \quad [a \in A^\vee \cap B^\vee]$$

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

$$\mathbin{\parallel_{A_i}^{i \in \{i_1, i_2\}}} P_i = P_{i_1} \mathbin{A_{i_1} \parallel A_{i_2}} P_{i_2} \quad \mathbin{\parallel_{A_i}^{i \in I \cup \{j\}}} P_i = \left(\mathbin{\parallel_{A_i}^{i \in I}} P_i \right) \mathbin{\cup_{i \in I} A_i \parallel A_j} P_j$$



- Pipelining

$$G = in?x : \mathbb{Z} \rightarrow com!g(x) \rightarrow G$$

$$F = com?y : \mathbb{Z} \rightarrow out!f(y) \rightarrow F$$

$$PIPE = G \text{ in.}\mathbb{Z} \cup com.\mathbb{Z} \parallel com.\mathbb{Z} \cup out.\mathbb{Z} F$$



- Pipelining

$$G = in?x : \mathbb{Z} \rightarrow com!g(x) \rightarrow G$$

$$F = com?y : \mathbb{Z} \rightarrow out!f(y) \rightarrow F$$

$$PIPE = G \text{ in.}\mathbb{Z} \cup com.\mathbb{Z} \parallel com.\mathbb{Z} \cup out.\mathbb{Z} F$$

- Kids painting

$$ISABELLA = isabella.get.box \rightarrow isabella.get.easel \rightarrow isabella.paint \rightarrow isabella.drop.box \rightarrow isabella.drop.easel \rightarrow ISABELLA$$

$$\square isabella.get.easel \rightarrow isabella.get.box \rightarrow isabella.paint \rightarrow isabella.drop.box \rightarrow isabella.drop.easel \rightarrow ISABELLA$$

$$KATE = kate.get.box \rightarrow kate.get.easel \rightarrow kate.paint \rightarrow kate.drop.box \rightarrow kate.drop.easel \rightarrow KATE$$

$$\square kate.get.easel \rightarrow kate.get.box \rightarrow kate.paint \rightarrow kate.drop.box \rightarrow kate.drop.easel \rightarrow KATE$$

$$EASEL = isabella.get.easel \rightarrow isabella.drop.easel \rightarrow EASEL$$

$$\square kate.get.easel \rightarrow kate.drop.easel \rightarrow EASEL$$

$$BOX = isabella.get.box \rightarrow isabella.drop.box \rightarrow BOX$$

$$\square kate.get.box \rightarrow kate.drop.box \rightarrow BOX$$

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



EXAMPLE: KIDS PAINTING (CONT'D)

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

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



EXAMPLE: KIDS PAINTING (CONT'D)

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

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

- 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



- 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

$$\begin{aligned}
 \text{PHIL}_i &= \text{enter}.i \rightarrow \\
 &\quad ((\text{pick}.i.i \rightarrow \text{pick}.i.((i + 1) \bmod 5) \rightarrow \text{eat}.i \\
 &\quad \quad \rightarrow \text{put}.i.i \rightarrow \text{put}.i.((i + 1) \bmod 5) \rightarrow \text{leave}.i \rightarrow \text{PHIL}_i) \\
 &\quad \square \\
 &\quad (\text{pick}.i.((i + 1) \bmod 5) \rightarrow \text{pick}.i.i \rightarrow \text{eat}.i \\
 &\quad \quad \rightarrow \text{put}.i.((i + 1) \bmod 5) \rightarrow \text{put}.i.i \rightarrow \text{leave}.i \rightarrow \text{PHIL}_i))
 \end{aligned}$$

$$\text{PHILS} = \text{PHIL}_0 \parallel \text{PHIL}_1 \parallel \text{PHIL}_2 \parallel \text{PHIL}_3 \parallel \text{PHIL}_4$$

$$\begin{aligned}
 \text{CHOP}_j &= \text{pick}.j.j \rightarrow \text{put}.j.j \rightarrow \text{CHOP}_j \\
 &\quad \square \text{pick}.((j - 1) \bmod 5).j \rightarrow \text{put}.((j - 1) \bmod 5).j \rightarrow \text{CHOP}_j
 \end{aligned}$$

$$\text{CHOPS} = \text{CHOP}_0 \parallel \text{CHOP}_1 \parallel \text{CHOP}_2 \parallel \text{CHOP}_3 \parallel \text{CHOP}_4$$

$$\text{COLLEGE} = \text{PHILS} \parallel \text{CHOPS}$$



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

$$\frac{P \xrightarrow{a} P'}{P \parallel Q \xrightarrow{a} P' \parallel Q} \quad [a \neq \checkmark] \quad \frac{P \xrightarrow{\checkmark} P' \quad Q \xrightarrow{\checkmark} Q'}{P \parallel Q \xrightarrow{\checkmark} P' \parallel Q}$$

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

$$PROC = PARENT \rightarrow (PROC \parallel CHILD)$$

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



- The process $P \parallel_A Q$ runs P and Q synchronized over exactly all the events in A :

$$\frac{\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q'}{P \parallel_A Q \xrightarrow{a} P' \parallel_A Q'}}{P \parallel_A Q \xrightarrow{a} P' \parallel_A Q'} \quad [a \in A^\vee] \qquad \frac{\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} P' \parallel_A Q'}{P \parallel_A Q \xrightarrow{a} P' \parallel_A Q'} \quad Q \parallel_A P \xrightarrow{a} Q \parallel_A P'}{P \parallel_A Q \xrightarrow{a} P' \parallel_A Q'} \quad [a \notin A^\vee]$$

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



- $P \setminus A$ behaves like P except that all the events from A are **hidden**



HIDING, RENAMING

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

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

- 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**



HIDING, RENAMING

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

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

- 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 $a = \checkmark$:

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



HIDING, RENAMING

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

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

- 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 $f : \Sigma^\vee \Rightarrow \Sigma^\vee$ 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 $P; Q$ runs P and then Q :

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



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

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

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

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

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