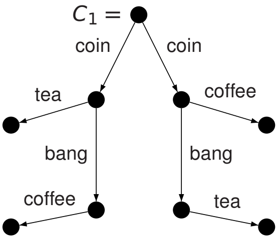
Introduction to formal methods

Stefan D. Bruda

CS 310, Winter 2025

- The method of algorithm verification discussed earlier needs access to the code (white box)
- Formal methods do not assume that code is available (black box)
  - We ignore everything in terms of internal functionality of the system
  - At any given time the system is in one of the many states from a given set S
    - Alternate definition: Set V of variables, each state is an interpretation over V
    - A finite / countable set of states given us a finite automaton / transition system
  - We change state by performing one action from a set of actions L
    - Only certain actions are enabled in each state
    - Special action: unobservable (τ)
  - Systems are considered reactive (react to the environment)



C_1 =
coin →
(tea → STOP □
bang → coffee → STOP)
□
coin →
(coffee → STOP □
bang → tea → STOP)

Communicating Sequential Processes (CSP)

- System specification uses a process algebra = identifies the states and actions
  - Compositional specification (start from simple "processes" and combine them to obtain increasingly complex ones)
  - Descriptions compilable into state transition diagrams / transition systems
- Such a process algebra is CSP:
  - Simplest process: STOP $\Rightarrow$ does not perform anything
  - Perform an action: $R = a \rightarrow P \Rightarrow R$ can perform a and then becomes P

$$\text{PRINTER} = \text{accept} \rightarrow \text{print} \rightarrow \text{STOP}$$

  - Choice: $R = P \,\square\, Q \Rightarrow R$ becomes either P or Q depending on what action is offered by the environment

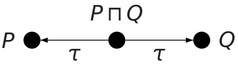
$$\text{PRINTER} = (\text{accept} \rightarrow \text{print} \rightarrow \text{STOP}) \,\square\, (\text{shutdown} \rightarrow \text{STOP})$$

  - Recursion: some action gets us back into a process (or state) already named

$$\text{PRINTER} = (\text{accept} \rightarrow \text{print} \rightarrow \text{PRINTER}) \,\square\, (\text{shutdown} \rightarrow \text{STOP})$$

MORE CSP

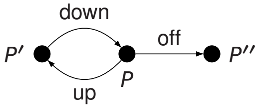
- Internal choice: $R = P \sqcap Q \Rightarrow R$ first becomes either P or Q "spontaneously" before any interaction with the environment:

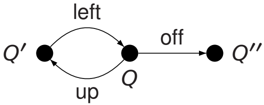


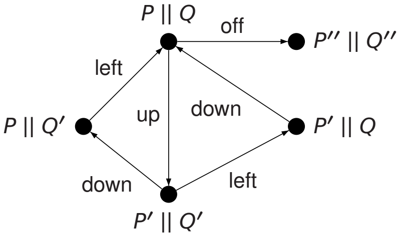
$$\text{PRINTER} = (\text{accept} \rightarrow \text{print} \rightarrow \text{STOP}) \sqcap (\text{shutdown} \rightarrow \text{STOP})$$

- Parallel composition: $R = P \,||\, Q \Rightarrow P$ and Q execute common actions synchronized and individual actions separately

$$P \,||\, Q, \text{ where: } \quad P = \text{up} \rightarrow \text{down} \rightarrow P \,\square\, \text{off} \rightarrow \text{STOP}$$
$$Q = \text{up} \rightarrow \text{left} \rightarrow Q \,\square\, \text{off} \rightarrow \text{STOP}$$







CSP Example: Kids Painting

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

$$
\text{PAINTING} = \text{ISABELLA} \mid\mid \text{KATE} \mid\mid \text{EASEL} \mid\mid \text{BOX}
$$

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

$$\mathsf{PAINTING} = \mathsf{I} \parallel \mathsf{K} \parallel \mathsf{E} \parallel \mathsf{B}$$

CSP Example: Kids Painting (cont'd)

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

$$\text{PAINTING} = I \,||\, K \,||\, E \,||\, B$$

- The process I1 || K5 || B1 || 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

CSP Example: 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 first

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

Verification with Traces

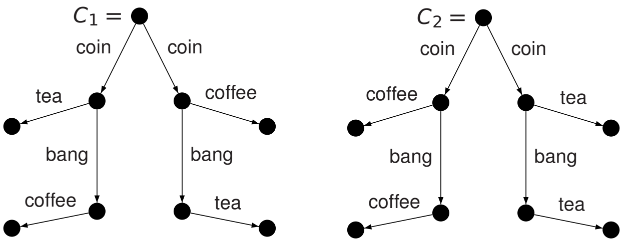
- Black box testing = we do not necessarily have access even to the CSP description
  - Typical scenario: CSP is used to define a specification (intended minimal system behaviour), but the system under test is a true back box
- Comparison between processes must be based on observable features
- Simplest observation: trace the execution of the process = record the actions of the process as they happen
  - $\text{traces}(\text{accept} \to \text{print} \to \text{STOP}) = \{\varepsilon, \langle\text{accept}\rangle, \langle\text{accept}, \text{print}\rangle\}$
  - $\text{PRINT} = \text{accept} \to \text{print} \to \text{PRINT}$

    $\Rightarrow \text{traces}(\text{PRINT}) = \langle\text{accept}, \text{print}\rangle^*(\varepsilon + \langle\text{accept}\rangle)$
- Defines an implementation relation and associated equivalence relation:

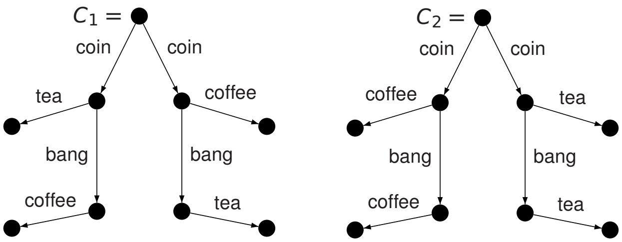
$$I \sqsubseteq_T S \quad \text{iff} \quad \text{traces}(S) \subseteq \text{traces}(I)$$
$$I \equiv_T S \quad \text{iff} \quad S \sqsubseteq_T I \;\&\&\; I \sqsubseteq_T S \qquad \text{iff} \quad \text{traces}(S) = \text{traces}(I)$$

- Simple verification with traces: Let $C = \text{traces}(S) \cap \overline{\text{traces}(I)}$
  - $I \sqsubseteq_T S$ whenever $C = \emptyset$; otherwise C contains counterexamples
  - C is cheap to compute whenever I and S are finite automata

Traces of Coffee Machines



- Are C_1 and C_2 equivalent?



- Are C_1 and C_2 equivalent?
- One may argue either way
- Trace-wise we actually have $C_1 \equiv_T C_2$

$$\text{traces}(C_1) = \text{traces}(C_2) = \{ \varepsilon, \langle\text{coin}\rangle, \langle\text{coin}, \text{tea}\rangle, \langle\text{coin}, \text{coffee}\rangle, \langle\text{coin}, \text{bang}\rangle, \langle\text{coin}, \text{bang}, \text{tea}\rangle, \langle\text{coin}, \text{bang}, \text{coffee}\rangle \}$$

IS TRACE PREORDER FINE ENOUGH?

$$\begin{aligned} X &= a \to b \to \text{STOP} \;\Box\; b \to a \to \text{STOP} \\ Y &= a \to b \to \text{STOP} \sqcap b \to a \to \text{STOP} \\ P &= a \to b \to \text{STOP} \end{aligned}$$

- In terms of traces the processes $X \| P$ and $Y \| P$ are the same: $\text{traces}(X \| P) = \text{traces}(Y \| P) = \{\varepsilon, a, ab\}$
- This trace equivalence however hides a difference in behaviour: $X \| P$ is always able to perform an a followed by a b, whereas $Y \| P$ can deadlock before any action is performed (depending on the choice Y makes before synchronizing with P)
  - Traces are not enough to identify deadlocks
- Trace equivalence is simple, but in some cases it may not be fine enough (depending on the application domain)