Sorting Machine

- A sorting machine $Sorter_n$, for each $n \geq 0$, is capable of sorting n-length sequences of positive integers
- Suppose $Sorter_n$ has sort $\{in, \overline{out}\}$
  - It must accept exactly n intergers one by one at in;
  - Then it must deliver them up one by one in descending order at $\overline{out}$, terminated by a zero;
  - After that, it must return to its start state

Sorting Machine

Specification:

$$Sortspec_n \stackrel{\text{def}}{=} in\ x_1.\cdots.in\ x_n.Hold_n\{x_1,\ldots,x_n\}$$

$$Hold_n(S) \stackrel{\text{def}}{=} \overline{out}(max\ S).Hold_n(S-\{max\ S\})\ \ (S\neq\emptyset)$$

$$Hold_n(\emptyset) \stackrel{\text{def}}{=} \overline{out}\ 0.Sortspec_n$$

where S ranges over multisets, and max S, min S are the maximum and minimum elements of S

Sorting Machine

Implementation

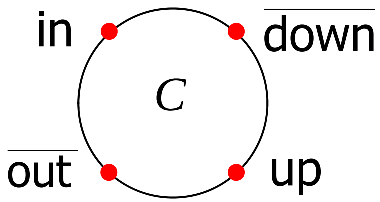
$$Sorter_n \stackrel{\text{def}}{=} \overbrace{C \frown \cdots \frown C}^{n \text{ times}} \frown B$$

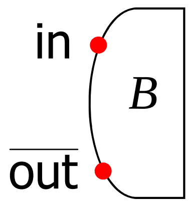
where

$$\begin{aligned}
C &\stackrel{\text{def}}{=} in(x).C'(x) \\
C'(x) &\stackrel{\text{def}}{=} \overline{down}(x).C + up(y).C''(x,y) \\
C''(x,y) &\stackrel{\text{def}}{=} \overline{out}(max\{x,y\}).C'''(min\{x,y\}) \\
C'''(x) &\stackrel{\text{def}}{=} if\ x = 0\ then\ \overline{out}(0).C\ else\ C'(x) \\
B &\stackrel{\text{def}}{=} \overline{out}(0).B
\end{aligned}$$

Note: The sorter is of fixed size

Sorting Machine





- B is a single barrier cell
- Assume C as a hardware component of fixed finite size which can be used to build sorting machines of any size
- C is independent of n

Sorting Machine

The idea in designing C is as follows:

- C should have storage capacity for two numbers, and be able to compare them
- Its behaviors must have two phases:
  - First phase: It receives inputs at **in** and put them out at **down**
  - Second phase: it receives inputs at **up** and (using comparison) puts them out at **out**
- For independence of the size of the sorter, it must be ready to change at any moment to its second phase
- Some cells will still be in the first phase while others are in the second

Sorting Machine

The defining equation of $Sortspec_{n+1}$ can be rewritten as follows:

$$Sortspec_{n+1} = in\, x_1. \cdots .in\, x_{n+1}.Hold_{n+1}\{x_1, \ldots, x_{n+1}\}$$

$$Hold_{n+1}\{y_1, \ldots, y_{n+1}\} = \overline{out}\, y_1. \cdots .\overline{out}\, y_{n+1}.\overline{out}\, 0.Sortspec_{n+1}$$

assuming that $y_1 \geq \ldots \geq y_{n+1}$.

Observational Equivalence

Content

- Introduction
- Strong Bisimulation
- Weak Bisimulation
- More Observational Equivalence

Introduction

Observational Equivalence

- Basic Idea
  - Two processes are observationally equivalent if no processes can observe any difference between the two processes
  - To observe is to interact
  - To observe is to be observed

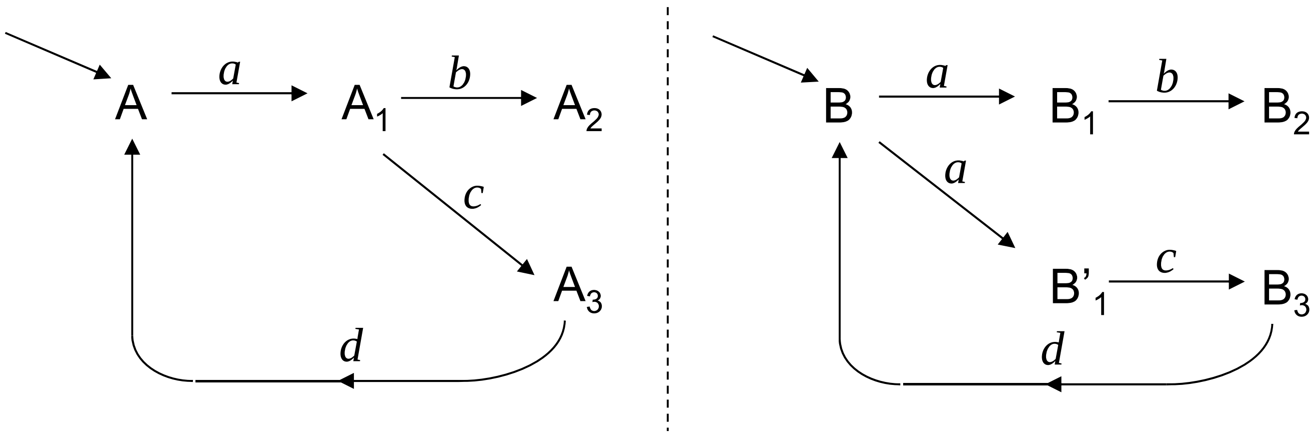
An Example

Consider two agents *A* and *B*:

$$
\begin{array}{ll|ll}
A \stackrel{\text{def}}{=} a.A_1 & & B \stackrel{\text{def}}{=} a.B_1 + a.B_1' & \\
A_1 \stackrel{\text{def}}{=} b.A_2 + c.A_3 & & B_1 \stackrel{\text{def}}{=} b.B_2 & B_1' \stackrel{\text{def}}{=} c.B_3 \\
A_2 \stackrel{\text{def}}{=} \mathbf{0} & & B_2 \stackrel{\text{def}}{=} 0 & \\
A_3 \stackrel{\text{def}}{=} d.A & & B_3 \stackrel{\text{def}}{=} d.B &
\end{array}
$$

Automata View

If A and B are thought of as finite-state automata over the set $\mathcal{A}$, then their transition graphs are as follows:



Trace Equivalence

If A2 and B2 are taken to be the accepting states, then A and B denote the same language:

$$\begin{aligned} A &= a.(b.\mathbf{0} + c.d.A) && \text{by substitution} \\ &= a.b.\mathbf{0} + a.c.d.A && \text{using the distributive law} \end{aligned}$$

Hence $\boxed{A = (a.c.d)^*.a.b.\mathbf{0}}$

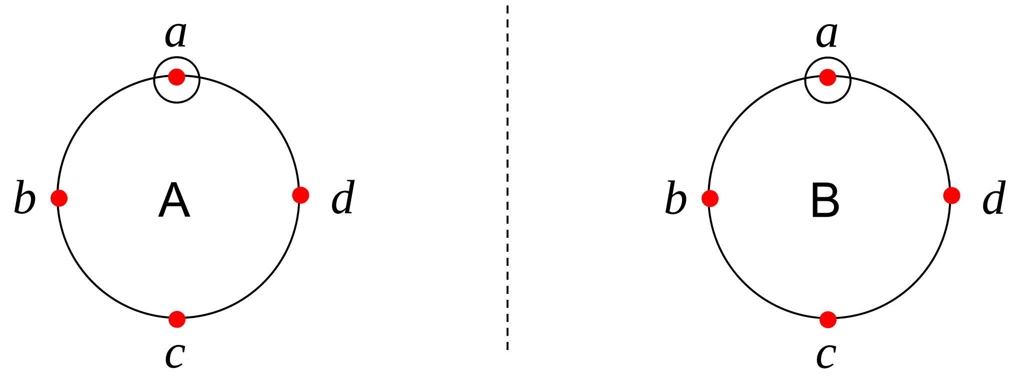
Similarly $B = a.b.\mathbf{0} + a.c.d.B$

And therefore $\boxed{B = (a.c.d)^*.a.b.\mathbf{0}}$

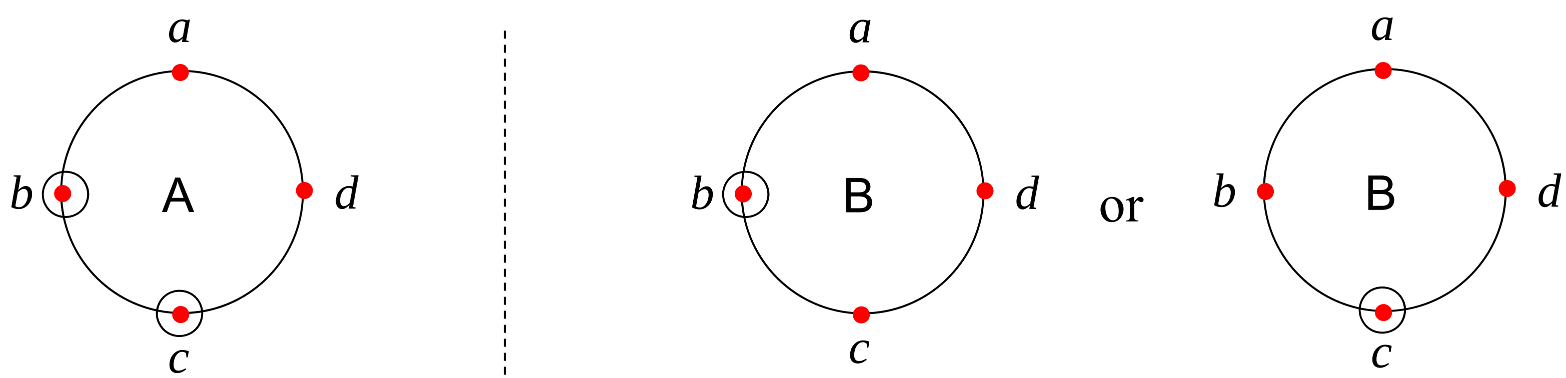
An Example: A Refined View

A and *B* may interact with their environment through the ports *a*, *b*, *c*, *d*.

At start:

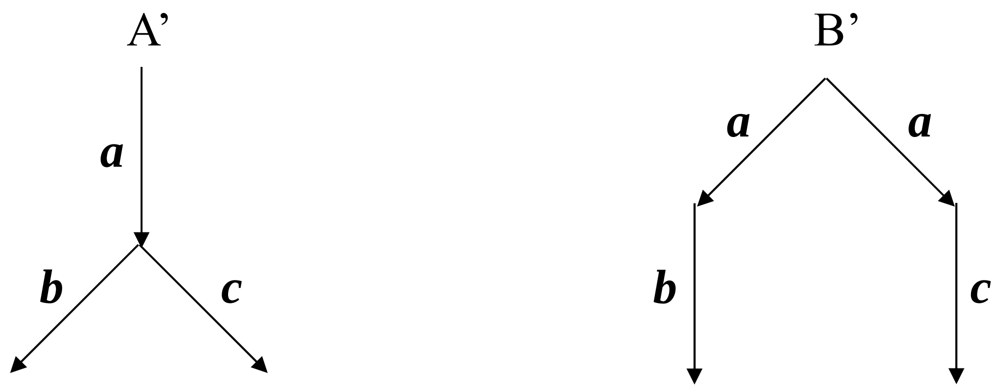


After the *a*-button is fired a difference emerges between *A* and *B*:



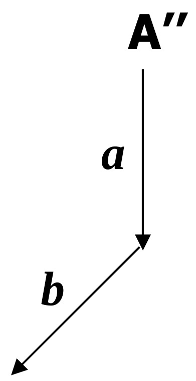
Nondeterminism

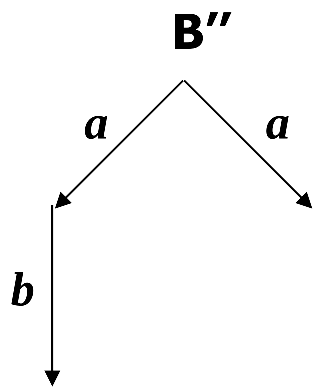
In our observational theory we shall differentiate between a deterministic choice and a nondeterministic choice



Deadlock

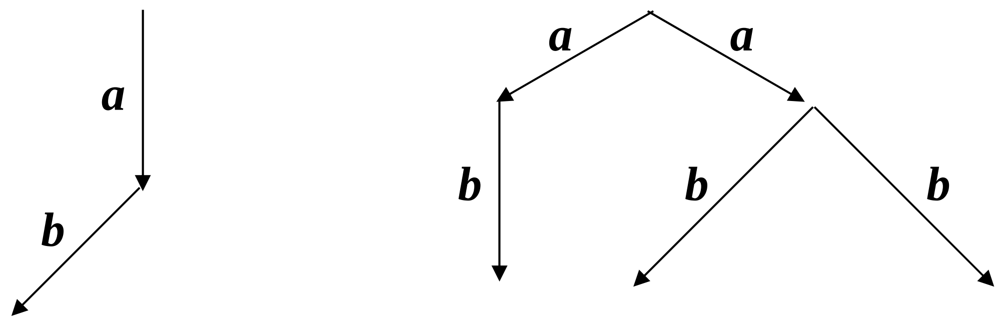
We shall also be able to detect deadlock





But

We shall want to equate the processes whose trees are as follows



Strong Bisimulation

Basic Idea

- All Actions, Including the τ Actions, are Treated Equally
- Strong Equivalence is the Core of All Equivalences

Bisimulation Property

P and Q are equivalent iff, for every action λ, every λ-derivative of *P* is equivalent to some λ-derivative of Q, and conversely.

It can be written formally as follows, using ~ for our equivalence relation:

P~Q iff, for all $\lambda \in \mathcal{A}$,
(i) If $P \xrightarrow{\lambda} P'$ then, for some Q', $Q \xrightarrow{\lambda} Q'$ and P'~Q' (*)
(ii) If $Q \xrightarrow{\lambda} Q'$ then, for some P', $P \xrightarrow{\lambda} P'$ and P'~Q'

Strong Bisimulation

Definition: A binary relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ over processes is a strong bisimulation if $(P, Q) \in \mathcal{S}$ implies that, for all $\lambda \in \mathcal{A}$,

(i) If $P \xrightarrow{\lambda} P'$ then, for some Q', $Q \xrightarrow{\lambda} Q'$ and $(P', Q') \in \mathcal{S}$

(ii) If $Q \xrightarrow{\lambda} Q'$ then, for some P', $P \xrightarrow{\lambda} P'$ and $(P', Q') \in \mathcal{S}$

Notice that any relation ~ which satisfies (*) is a strong bisimulation.

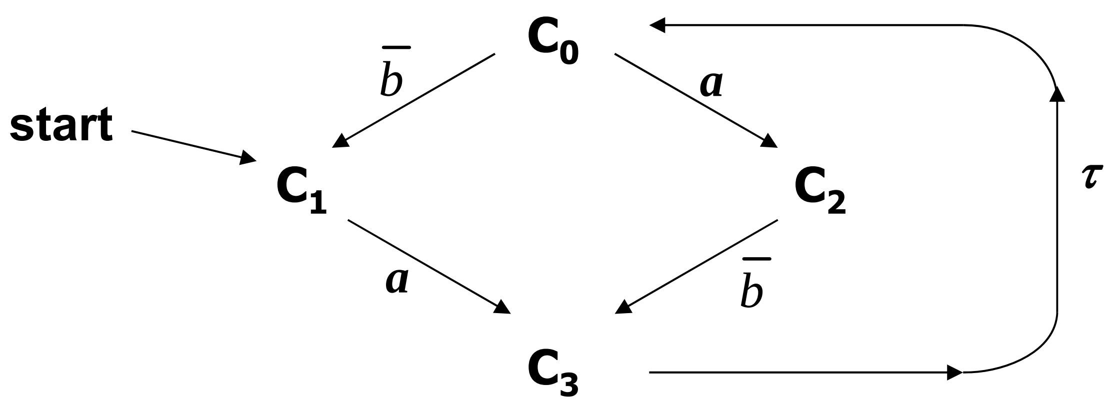
An Example

$$A \overset{\text{def}}{=} a.A' \qquad B \overset{\text{def}}{=} c.B'$$
$$A' \overset{\text{def}}{=} \overline{c}.A \qquad B' \overset{\text{def}}{=} \overline{b}.B$$

$$\begin{aligned}\mathcal{S} = \{ & ((A \mid B)\backslash c, C_1), \\ & ((A' \mid B)\backslash c, C_3), \\ & ((A \mid B')\backslash c, C_0), \\ & ((A' \mid B')\backslash c, C_2) \ \}\end{aligned}$$

is a bisimulation.

$$C_0 \overset{\text{def}}{=} \overline{b}.C_1 + a.C_2$$
$$C_1 \overset{\text{def}}{=} a.C_3$$
$$C_2 \overset{\text{def}}{=} \overline{b}.C_3$$
$$C_3 \overset{\text{def}}{=} \tau.C_0$$



Equivalence Property

The converse $\mathcal{R}^{-1}$ of a binary relation and

the composition $\mathcal{R}_1\mathcal{R}_2$wo binary relations are defined by

$$\mathcal{R}^{-1} = \{(y,x) : (x,y) \in \mathcal{R}\}$$

$$\mathcal{R}_1\mathcal{R}_2 = \{(x,z) : \text{for some } y, (x,y) \in \mathcal{R}_1 \text{ and } (y,z) \in \mathcal{R}_2\}$$

Property:

Assume that each $\mathcal{S}_i$ (i = 1, 2, …) is a strong bisimulation.

Then the following relations are all strong bisimulations:

(1) $Id_{\mathcal{P}}$	(3) $\mathcal{S}_1\mathcal{S}_2$
(2) $\mathcal{S}_i^{-1}$	(4) $\bigcup_{i \in I} \mathcal{S}_i$

Strong Bisimilarity

Definition:

P and Q are strongly bisimilar, written P ~ Q, if $(P, Q) \in \mathcal{S}$ for some strong bisimulation $\mathcal{S}$. This may be equivalently expressed as follows:

$$\sim = \bigcup\{\mathcal{S} : \mathcal{S} \textit{ is a strong bisimulation}\}$$

Proposition

(1) ~ is the largest strong bisimulation.

(2) ~ is an equivalence relation.

Strong Bisimilarity, Continued

Definition:

$P \sim' Q$ iff, for all $\lambda \in \mathcal{A}$,

(i) Whenever $P \xrightarrow{\lambda} P'$ then for some Q' , $Q \xrightarrow{\lambda} Q'$ and $P' \sim Q'$

(ii) Whenever $Q \xrightarrow{\lambda} Q'$ then for some P' , $P \xrightarrow{\lambda} P'$ and $P' \sim Q'$

Property: $P \sim Q$ implies $P \sim' Q$ (#)

Strong Bisimilarity, Continued

Lemma The relation $\sim'$ is a strong bisimulation.

Proof. Let $P \sim' Q$, and let $P \xrightarrow{\lambda} P'$. It will be enough, for the lemma, to find Q' so that $Q \xrightarrow{\lambda} Q'$ and $P' \sim' Q'$. But by the definition of $\sim'$ we can indeed find Q' so that $Q \xrightarrow{\lambda} Q'$ and $P' \sim Q'$; hence also $P' \sim' Q'$ by ($\sharp$), and we are done. $\square$

Strong Bisimilarity, Continued

$\sim$ satisfies the (*) property:

P ~ Q iff, for all $\lambda \in \mathcal{A}$,

(i) Whenever $P \xrightarrow{\lambda} P'$ then for some Q', $Q \xrightarrow{\lambda} Q'$ and P' ~ Q'

(ii) Whenever $Q \xrightarrow{\lambda} Q'$ then for some P', $P \xrightarrow{\lambda} P'$ and P' ~ Q'

How to Prove Processes Equivalent

- We are to prove $P \sim Q$
  1. Construct some binary relation R on processes such that $(P,Q) \in R$
  2. Show that R is a strong bisimulation
  3. Conclude that $P \sim Q$