ABSTRACTION, REFINEMENT AND EQUIVALENCE

Trace equivalence and bisimulation

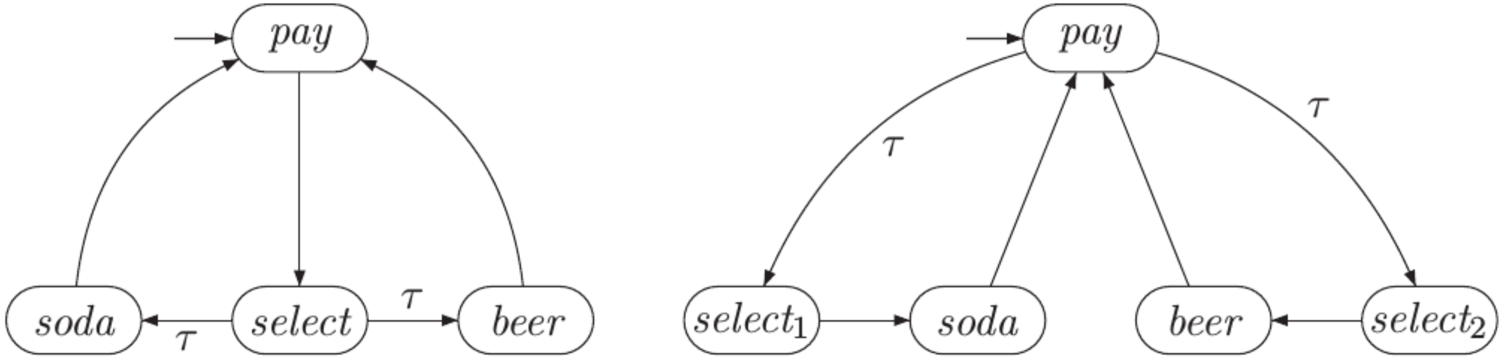
Trace Equivalence (reminder)

- Transitions systems TS and TS' are **trace equivalent** wrt to a set AP if $Traces_{AP}(TS)=Traces_{AP}(TS')$

- **Theorem**: Let P be a linear-time property (over AP). If $Traces_{AP}(TS) \subseteq Traces_{AP}(TS')$ then $TS \vDash P \rightarrow TS' \vDash P$

- **Corollary**: Trace-equivalents transition systems satisfy the same LT property

2 vending machines for soda and beer



- Left: a machine that after insertion of a coin **nondeterministically** chooses to either provide soda or beer.
- Right: Two selection buttons (one for each beverage), but after insertion of a coin, nondeterministically blocks one of the buttons.
- In either case, the user has no control over the beverage obtained—the choice of beverage is under full control of the vending machine.
- Let AP = { pay, soda, beer}. The machines exhibit the same traces over AP (both traces are alternating sequences of pay and either soda or beer).
- The vending machines are thus trace-equivalent, and then satisfy exactly the same LT properties (there is no LT property distinguishing them)
- For a user (or environment or concurrent process) interacting with the machines, their behavior is different! **One has one button, the other has two**.

Need a «finer» concept of equivalence

- Trace-equivalence can be inadequate for reactive (interactive, concurrent) systems, as shown by the example.
- Reason: it totally ignores the internal branching structure
  - This is what we want for parsers and compilers, where we only care for language (i.e., trace) equivalence
- Idea of **simulation** among machines: a transition system TS' can simulate transition system TS if every step of TS can be matched by one (or more) steps in TS.
- **Bisimulation equivalence** denotes the possibility of mutual, stepwise simulation.
-

Bisimulation equivalence

Let Post(s) be the set of states reachable in one step from s.

Definition 7.1. Bisimulation Equivalence

Let $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP, L_i)$, $i = 1, 2$, be transition systems over AP. A *bisimulation* for (TS_1, TS_2) is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ such that

Condition (A) asserts that every initial state of TS1 is related to an initial state of TS2, and vice versa.

(A) $\forall s_1 \in I_1 \ (\exists s_2 \in I_2.\ (s_1, s_2) \in \mathcal{R})$ and $\forall s_2 \in I_2 \ (\exists s_1 \in I_1.\ (s_1, s_2) \in \mathcal{R})$

(B) for all $(s_1, s_2) \in \mathcal{R}$ it holds:

(1) $L_1(s_1) = L_2(s_2)$ (s1 and s2 are equally labeled, ensuring their “local” equivalence)

(2) if $s_1' \in Post(s_1)$ then there exists $s_2' \in Post(s_2)$ with $(s_1', s_2') \in \mathcal{R}$

(3) if $s_2' \in Post(s_2)$ then there exsist $s_1' \in Post(s_1)$ with $(s_1', s_2') \in \mathcal{R}$.

(every outgoing transition of s1 is matched by an outgoing transition of s2, and viceversa)

TS_1 and TS_2 are *bisimulation-equivalent* (bisimilar, for short), denoted $TS_1 \sim TS_2$, if there exists a bisimulation $\mathcal{R}$ for (TS_1, TS_2). ■

Explanation of Conditions B

for all $(s_1, s_2) \in \mathcal{R}$

(1) $L_1(s_1) = L_2(s_2)$

(2) if $s_1' \in Post(s_1)$ then there exists $s_2' \in Post(s_2)$ with $(s_1', s_2') \in \mathcal{R}$

every outgoing transition of s1 is matched by an outgoing transition of s2:

$$\begin{array}{ccc} s_1 & \mathcal{R} & s_2 \\ \downarrow & & \\ s_1' & & \end{array} \quad \text{can be complemented to} \quad \begin{array}{ccc} s_1 & \mathcal{R} & s_2 \\ \downarrow & & \downarrow \\ s_1' & \mathcal{R} & s_2' \end{array}$$

(3) if $s_2' \in Post(s_2)$ then there exsist $s_1' \in Post(s_1)$ with $(s_1', s_2') \in \mathcal{R}$.

$$\begin{array}{ccc} s_1 & \mathcal{R} & s_2 \\ & & \downarrow \\ & & s_2' \end{array} \quad \text{can be complemented to} \quad \begin{array}{ccc} s_1 & \mathcal{R} & s_2 \\ \downarrow & & \downarrow \\ s_1' & \mathcal{R} & s_2' \end{array}$$

Examples

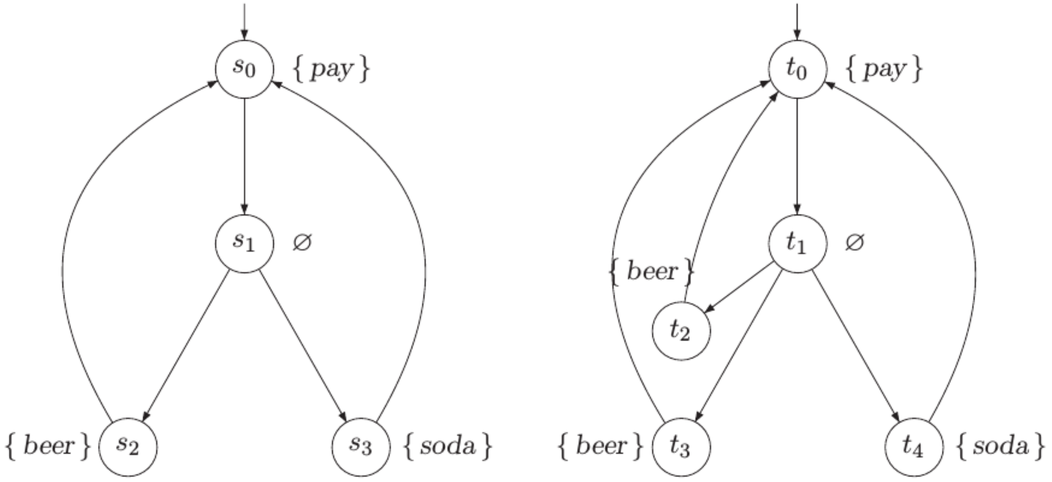


Figure 7.2: Two bisimilar beverage vending machines

The right-hand transition system has an additional option to deliver beer, but this is not observable by a user. This suggests an equivalence

$$\mathcal{R} \;=\; \{(s_0, t_0), (s_1, t_1), (s_2, t_2), (s_2, t_3), (s_3, t_4)\}$$

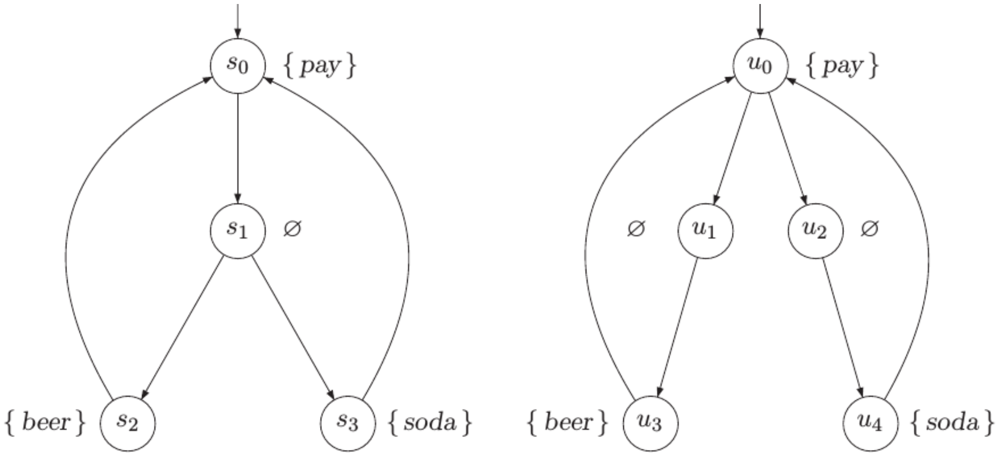


Figure 7.3: Nonbisimulation-equivalent beverage vending machines.

Not bisimilar for AP = { pay, beer, soda } .
The only candidates for mimicking state s1 are the states u1 and u2. However, neither of these states can mimic all transitions of s1: either the possibility for soda or for beer is missing.
They were bisimilar, however, if instead of *beer* and *soda* we had a label *drink*

Bisimulation on Paths

$$
\begin{array}{ccc}
s_1 & -\mathcal{R}- & s_2 \\
\downarrow & & \\
s_{1,1} & & \\
\downarrow & & \\
s_{2,1} & & \\
\downarrow & & \\
\vdots & &
\end{array}
\qquad \text{can be completed to} \qquad
\begin{array}{ccc}
s_1 & -\mathcal{R}- & s_2 \\
\downarrow & & \downarrow \\
s_{1,1} & -\mathcal{R}- & s_{1,2} \\
\downarrow & & \downarrow \\
s_{2,1} & -\mathcal{R}- & s_{2,2} \\
\downarrow & & \downarrow \\
\vdots & & \vdots
\end{array}
$$

Let TS_1 and TS_2 be transition systems over AP, $\mathcal{R}$ a bisimulation for (TS_1, TS_2), and $(s_1, s_2) \in \mathcal{R}$. Then for each (finite or infinite) path $\pi_1 = s_{0,1}\, s_{1,1}\, s_{2,1}\, \ldots \in Paths(s_1)$ there exists a path $\pi_2 = s_{0,2}\, s_{1,2}\, s_{2,2}\, \ldots \in Paths(s_2)$ of the same length such that $(s_{j,1}, s_{j,2}) \in \mathcal{R}$ for all j.

Properties

- Bisimulation equivalence implies trace-equivalence
  - As a consequence, LTL formulae cannot distinguish two bisimulation-equivalent TS
- Reflexivity, Transitivity, and Symmetry of $\sim$:
  - **For a fixed set AP of atomic propositions, the relation $\sim$ is an equivalence relation.**

- Advantage of bisimulation: given a TS, if we can find a TS', much smaller than TS, such that TS $\sim$ TS', then we can verify a property on TS' rather than on TS.
  - For instance, TS can be infinite-state and TS' can be finite-state
  - We need some additional definition

Bisimulation equivalence as relation on states

- Bisimulation can be considered as a relation between states of a *single* transition system.
- The goal is «minimization» of the number of states relevant to prove a certain property

Let $TS = (S, Act, \to, I, AP, L)$ be a transition system. A *bisimulation* for TS is a binary relation $\mathcal{R}$ on S such that for all $(s_1, s_2) \in \mathcal{R}$:

1. $L(s_1) = L(s_2)$.
2. If $s'_1 \in Post(s_1)$, then there exists an $s'_2 \in Post(s_2)$ with $(s'_1, s'_2) \in \mathcal{R}$.
3. If $s'_2 \in Post(s_2)$, then there exists an $s'_1 \in Post(s_1)$ with $(s'_1, s'_2) \in \mathcal{R}$.

States s_1 and s_2 are *bisimulation-equivalent* (or bisimilar), denoted $s_1 \sim_{TS} s_2$, if there exists a bisimulation $\mathcal{R}$ for TS with $(s_1, s_2) \in \mathcal{R}$. ■

$\sim_{TS}$ is an equivalence relation on states

For transition system $TS = (S, Act, \rightarrow, I, AP, L)$ *it holds that:*

1. $\sim_{TS}$ *is an equivalence relation on* S.
2. $\sim_{TS}$ *is a bisimulation for TS.*
3. $\sim_{TS}$ *is the coarsest bisimulation for TS.*

Quotient TS

An equivalence relation can be used to define the *quotient* of a given set S (the set of equivalence classes of elements of S. We can then define the *quotient TS*

For transition system $TS = (S, Act, \rightarrow, I, AP, L)$ and bisimulation $\sim_{TS}$, the *quotient transition system* $TS/\sim_{TS}$ is defined by

$$TS/\sim_{TS} = (S/\sim_{TS}, \{\tau\}, \rightarrow', I', AP, L')$$

where:

- $I' = \{ [s]_\sim \mid s \in I \}$,
- $\rightarrow'$ is defined by

(Actions are ignored!)

$$\frac{s \xrightarrow{\alpha} s'}{[s]_\sim \xrightarrow{\tau} [s']_\sim},$$

- $L'([s]_\sim) = L(s)$.

Example: Bakery Mutex Algorithm

Process 1:

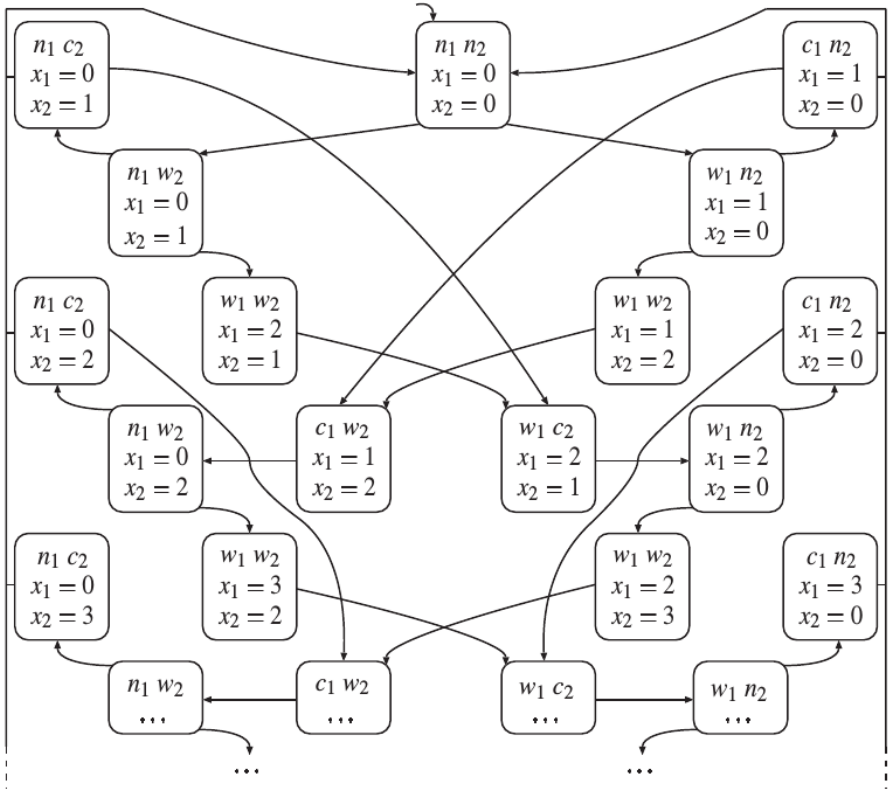
$$
\begin{array}{ll}
 & \ldots\ldots \\
 & \textbf{while}\ \text{true}\quad \{ \\
 & \quad \ldots\ldots \\
n_1 : & \quad x_1 := x_2 + 1; \\
w_1 : & \quad \textbf{wait until}\ (x_2 = 0 \,||\, x_1 < x_2)\,\{ \\
c_1 : & \quad\quad \ldots \text{critical section} \ldots \} \\
 & \quad x_1 := 0; \\
 & \quad \ldots\ldots \\
 & \}
\end{array}
$$

Process 2:

$$
\begin{array}{ll}
 & \ldots\ldots \\
 & \textbf{while}\ \text{true}\quad \{ \\
 & \quad \ldots\ldots \\
n_2 : & \quad x_2 := x_1 + 1; \\
w_2 : & \quad \textbf{wait until}\ (x_1 = 0 \,||\, x_2 < x_1)\,\{ \\
c_2 : & \quad\quad \ldots \text{critical section} \ldots \} \\
 & \quad x_2 := 0; \\
 & \quad \ldots\ldots \\
 & \}
\end{array}
$$

- x1, x2 initialized at 0. They are used to resolve a conflict if both processes want to enter the critical section—the value is like a ticket for a queue in a shop.
- On requesting access, P1 sets x1 to x2 +1 (P1 gives priority to P2)
- If process P1 is waiting, and x1 < x2 or x2 =0, then it may enter; at exit x1:=0
- Symmetrically for P2.
- Unlike Peterson's, *"easy" to extend to any number of processes* (x_i:=x_j+1, x_j is max ticket)
- NB: in practice, it requires no reordering of the instructions (as Peterson's), which is nowadays rarely true.

- As the value of x1 and x2 may grow unboundedly, the **underlying transition system P1|| P2 is infinite**.



Example (cont.)

Model checking a LTL formula becomes impossible but…

- x1 > x2 > 0 or x2 > x1 > 0 or x1 = 0 or x2 = 0 are the only conditions relevant to enter the critical section.

- Let AP= $\{ noncrit_i, wait_i , crit_i \mid i = 1, 2 \} \cup \{ x1 > x2 > 0, x2 > x1 > 0, x1 = 0, x2 = 0\}$
- i.e., AP is used to represent current state and current value of conditions

- We can define a relation on states of the original transition system using the above conditions
  – We could do the same also for more than 2 processes, as long as the max number is fixed.

Bisimulation quotient transition system

A TS and its quotient are bisimilar

Theorem 7.14. ***Bisimulation Equivalence of*** TS ***and*** $TS/\!\sim$

For any transition system TS *it holds that* $TS \sim TS/\!\sim$.

- Therefore, we can prove LTL properties on the quotient TS rather than on the original one.
- For instance, the following LTL properties hold for the example (safety, non-starvation):

$$\Box(\neg crit_1 \vee \neg crit_2)$$

$$(\Box\Diamond\, wait_1 \Rightarrow \Box\Diamond\, crit_1) \wedge (\Box\Diamond\, wait_2 \Rightarrow \Box\Diamond\, crit_2)$$

What about CTL and bisimulations?

- Does bisimulation equivalence helps in proving CTL properties?
- We need to define *CTL equivalence* and compare it with bisimulation equivalence

CTL equivalence

- In general, states in a transition system are equivalent with respect to a logic whenever these states cannot be distinguished by the truth value of any formulae of the logic.

- Let TS, TS1, and TS2 be transition systems over AP *without terminal states (infinite paths only)*.

- States s_1, s_2 in TS are **CTL-equivalent**, denoted $s_1 \equiv_{CTL} s_2$, if $s_1 \models \Phi$ iff $s_2 \models \Phi$ for all CTL formulae over AP.
- TS1 and TS2 are **CTL-equivalent**, denoted TS1 $\equiv_{CTL}$ TS2, if TS1 $\models \Phi$ iff TS2 $\models \Phi$ for all CTL formulae over AP.

CTL and bisimulation equivalence coincide!

- For a **finite** transition system TS *without terminal states*:

$$\sim_{TS} \; = \; \equiv_{CTL}$$

- Similarly for **finite** transition systems TS1, TS2 (over AP) without terminal states, the following two statements are equivalent:

(a) TS1 $\sim$ TS2

(b) TS1 $\equiv_{CTL}$ TS2 , i.e., TS1 and TS2 satisfy the same CTL formulae.

NB: finiteness assumption is necessary. However, for an *infinite-state* TS, bisimulation equivalence implies CTL equivalence (so we can still prove CTL formulae on the quotient TS).

EXAMPLE

- CTL may (only) distinguish non-bisimilar systems
- **AXEX beer**

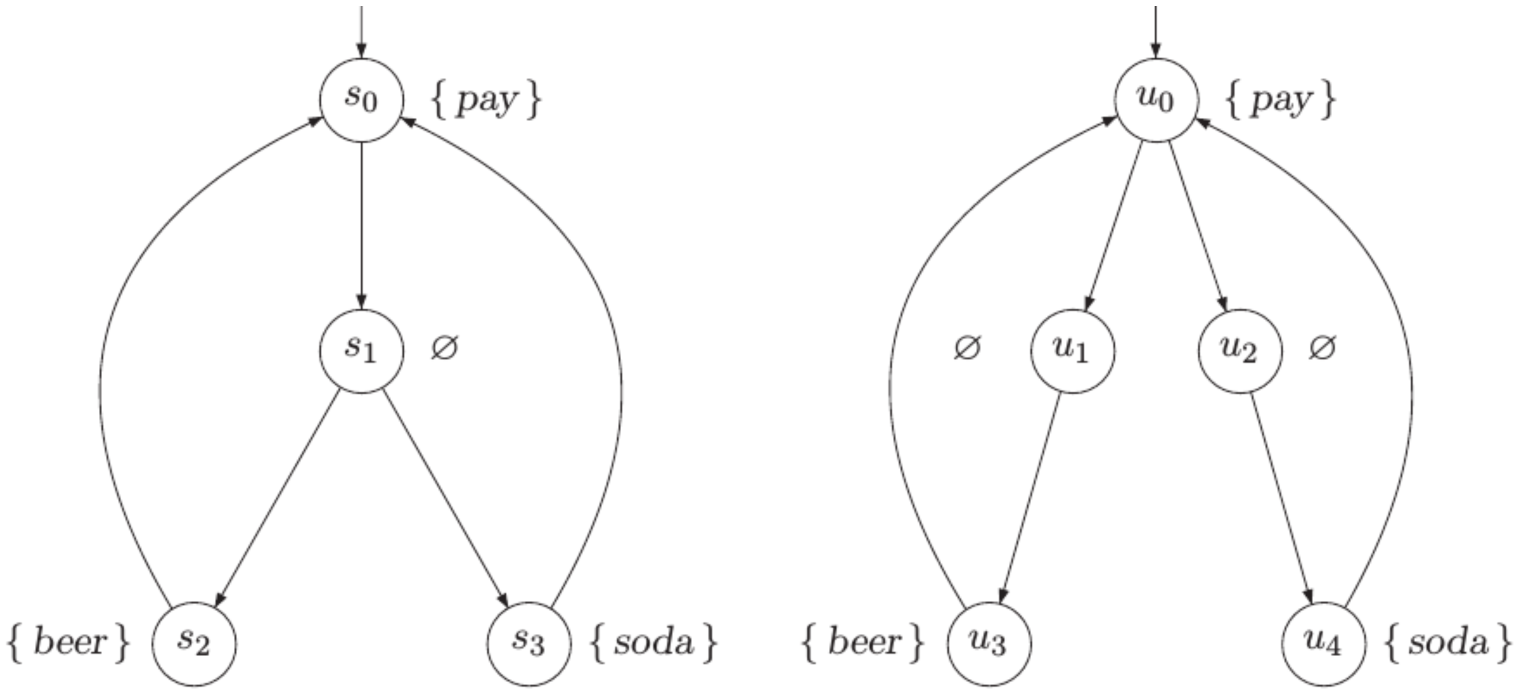


Figure 7.3: Nonbisimulation-equivalent beverage vending machines.

TRUE **FALSE**

Deciding bisimulation equivalence?

- **Theorem**: The bisimulation quotient of a ***finite*** transition system TS = (S, Act, →, I, AP, L) can be computed in time O (|S|·|AP| + M · log |S|)---where M denotes the number of edges in the state graph

- An algorithm (n. 32) can be found in Baier&Katoen's book.
- Goals of the algorithm:
  1. Verify the bisimilarity of two finite transition systems TS1 and TS2 (by considering the quotient of the composite TS1 ⊕ TS2, a disjoint union of TS1,TS2). Then TS1 ∼ TS2 if and only if, for every initial state s1 of TS1, there exists a bisimilar initial state s2 of TS2, and vice versa.

     If TS1 ∼ TS2 then TS1 and TS2 are trace equivalent: an efficient (but not necessary) condition for Trace equivalence (PSPACE-complete for TS and NFA).
  2. Obtain the abstract (and thus smaller) quotient transition system TS/∼ in a fully automated manner. As TS ∼ TS/∼ then any verification result for TS/∼—either being negative or positive—carries over to TS.

ABSTRACTION AND REFINEMENT: SIMULATION RELATION

Abstraction and Refinement

- Transition systems can model software or hardware at various abstraction levels.
  - The lower the abstraction level, the more implementation details are present; at high abstraction levels, such details are deliberately left unspecified.
- As in the Bakery example, we may start from a «detailed» TS and want to define a suitable *more abstract* TS
  - possibily preserving properties of interest, but easier to manage.
  - This process is called *abstraction*
- We may instead start from a more abstract model and want to add more implementation details
  - possibily preserving properties of interest, but closer to real system
  - This process is called *refinement*

Implementation Relation

- An *implementation relation* is a binary relation between two TS at different abstraction levels.
  - Example: *trace inclusion*: Trace(TS)$\subseteq$Trace(TS'). TS is an implementation of TS'

- When two TS,TS' are related by an implementation relation, one model is said to be *refined* by the other; the second is said to be an *abstraction* of the first.

- If the implementation relation is an equivalence, then the two TS cannot be distinguished (same observable properties at the relevant abstraction level).

- Since it is possible to define many different implementation relations, there are different concepts of abstraction and refinement

Examples

- A Linear-time property P (e.g., a LTL formula) is preserved by trace equivalence, bisimulation equivalence, but also trace inclusion.
  - Trace equivalence: Traces(TS)=Traces(TS') : TS' |= P iff TS |= P .
  - Bisimulation equivalence (which implies trace equivalence)
  - Trace inclusion: Traces(TS)$\subseteq$Traces(TS’):
    if TS' |= P then TS |= P
- All those are implementation relations.
- Equivalences are usually harder to come by, so we are looking for other useful implementation relations

Remark on set AP and bisimulations

- The fixed set AP plays a crucial role in comparing transition systems using bisimulation for checking the implementation relation.
- The set AP used in a bisimulation stands for the set of all “relevant” atomic propositions.
  - All other atomic propositions are understood as negligible and are ignored in the comparison.
- If TS is a *refinement* of TS' (e.g., it incorporates some implementation details), then usually the set AP of TS is a proper superset of the set AP' of TS' .
  - To compare TS and TS’, the set of common atomic propositions, AP’, is a reasonable choice.
  - In this way, it is possible to check whether the branching structure of TS agrees with that of TS’ *when considering all observable information in AP’ .*
- For checking the equivalence of TS and TS' wrt the satisfaction of a temporal logic formula Φ, it suffices to consider as AP the atomic propositions of Φ.

Simulations that are not bisimulations

- Bisimulation relations are equivalences requiring two bisimilar states to exhibit identical stepwise behavior.
- *Simulation* relations only require that whenever a state s' simulates state s, then s' can mimic all stepwise behavior of s, but the reverse is not guaranteed
  - s' may perform transitions that cannot be matched by s.
  - hence, every successor of s has a corresponding successor of s', but the reverse does not necessarily hold.
- Simulation relations often used for:
  - showing that one system correctly implements another, more abstract system.
  - finding a smaller abstract model preserving at least some properties of interest

Formal definition of simulation

Definition 7.47. Simulation Order

Let $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP, L_i)$, $i = 1, 2$, be transition systems over AP. A *simulation* for (TS_1, TS_2) is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ such that

(A) $\forall s_1 \in I_1 \;.\; (\exists s_2 \in I_2.\ (s_1, s_2) \in \mathcal{R})$

(B) for all $(s_1, s_2) \in \mathcal{R}$ it holds that:

(1) $L_1(s_1) \;=\; L_2(s_2)$

(2) if $s_1' \in Post(s_1)$, then there exists $s_2' \in Post(s_2)$ with $(s_1', s_2') \in \mathcal{R}$.

TS_1 is simulated by TS_2 (or, equivalently, TS_2 simulates TS_1), denoted $TS_1 \preceq TS_2$, if there exists a simulation $\mathcal{R}$ for (TS_1, TS_2). ■

(A) requires that all initial states in TS1 are related to an initial state of TS2 (but, there might be initial states of TS2 that are not matched by an initial state of TS1).

(B) are as for bisimulations, but the symmetric counterpart of (B.2) is not required.

Example: AP = {pay, beer, soda }

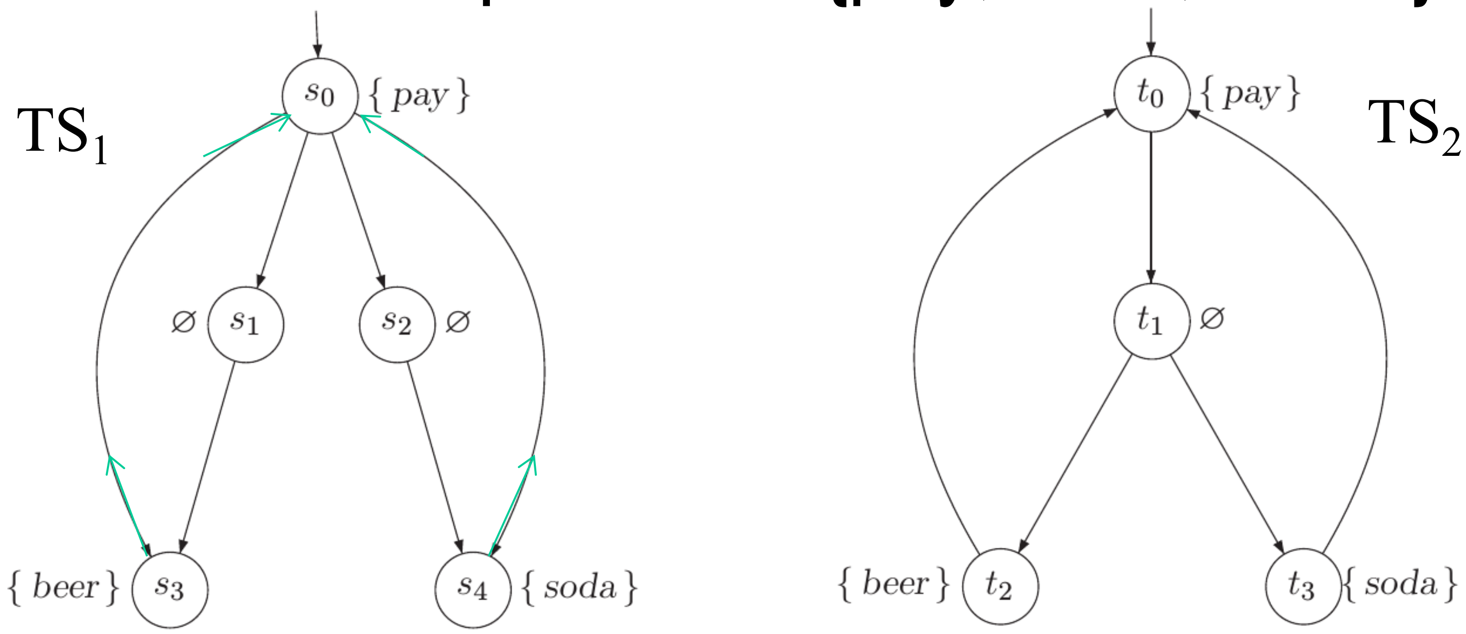


Figure 7.17: The vending machine on the right simulates the one on the left.

$$\mathcal{R} = \{ (s_0, t_0), (s_1, t_1), (s_2, t_1), (s_3, t_2), (s_4, t_3) \}$$

is a simulation for (TS_1, TS_2). Since $\mathcal{R}$ contains the pair of initial states (s_0, t_0), $TS_1 \preceq TS_2$. The reverse does not hold, i.e., $TS_2 \not\preceq TS_1$, since there is no state in TS_1 that can mimic state t_1. This is due to the fact that the options "*beer*" and "*soda*" are possible in state t_1, but in no state in TS_1.

Example: AP = { pay, drink }

- Assume states s3 , s4 , t2 and t3 labeled with {drink}.

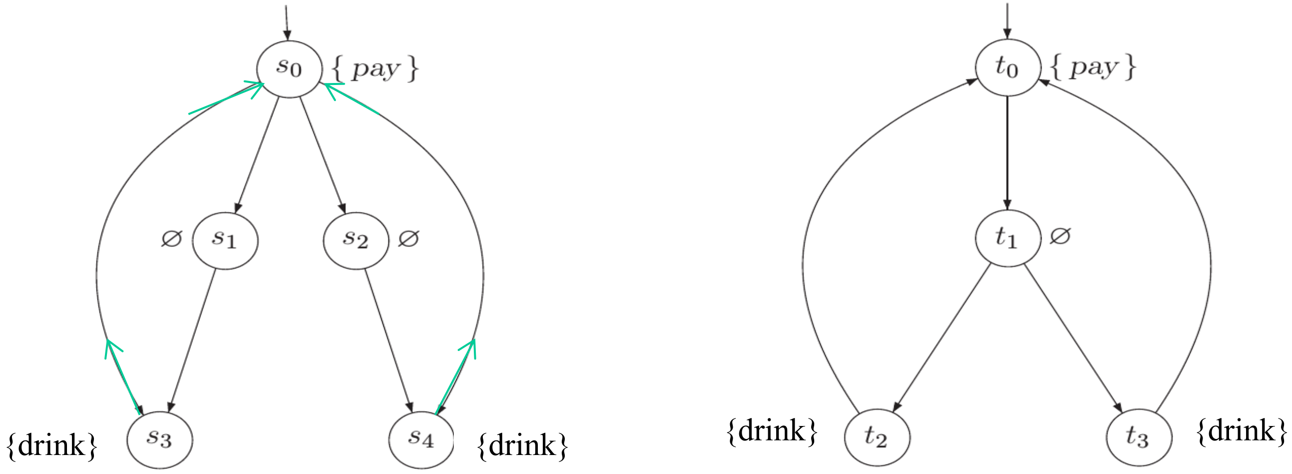


Figure 7.17: The vending machine on the right simulates the one on the left.

Relation $\mathcal{R}$ is again a simulation for (TS_1, TS_2), and thus $TS_1 \preceq TS_2$. Its inverse,

$$\mathcal{R}^{-1} = \{ (t_0, s_0), (t_1, s_1), (t_1, s_2), (t_2, s_3), (t_3, s_4) \},$$

is a simulation for (TS_2, TS_1). We thus also obtain $TS_2 \preceq TS_1$. ■

Refinement and abstraction vs. simulations

1. If TS1 is obtained from TS2 by deleting transitions of TS2 (e.g., replacing nondeterministic choices in TS2 with only one alternative) then *TS1 is simulated by TS2*
   - TS1 is thus a **refinement** of TS2, since TS1 resolves some nondeterminism in TS2.
2. If TS2 is obtained from TS1 with some "abstraction" then *TS1 is simulated by TS2*.

- Need to define that T2 is an *abstraction* of TS1 if:
  - There is a common set AP of atomic propositions.
  - States of TS2 are a set of "abstract states".
  - There is an *abstraction function* f associating each (concrete) state s of TS1 with the "abstract state" f(s) of TS2 (and respecting the label in AP)

- Abstractions differ in the choice of the abstract states, the abstraction function f, and the relevant propositions AP.

Abstraction via simulations

- TS2 is an abstraction of TS1 if:
  - There is a common set AP of atomic propositions.
  - States of TS2 are a set of “abstract states”.
  - There is an *abstraction function* f associating each (concrete) state s of TS1 with the “abstract state” f(s) of TS2 (and respecting the label in AP)

- Abstractions differ in the choice of the abstract states, the abstraction function f, and the relevant propositions AP.

Example

```
1: while x > 0 {
2:     x = x − 1;
3:     y = y + 1;
   }
4: if (even(y)) return 1;
5: return 0;
```

we want keep track of whether x > 0 or x=0, and whether y is even or odd, but not of the precise values of x and y.

dom(x) = {gzero, zero}

dom(y) = {even, odd}.

The abstraction function f which maps a concrete state (location, value, value) to an abstract one:

$$f(\langle \ell, x = v, y = w \rangle) \;=\; \begin{cases} \langle \ell, x = \mathit{gzero}, y = \mathit{even} \rangle & \text{if } x > 0 \wedge y \text{ is even} \\ \langle \ell, x = \mathit{gzero}, y = \mathit{odd} \rangle & \text{if } x > 0 \wedge y \text{ is odd} \\ \langle \ell, x = \mathit{zero},\ y = \mathit{even} \rangle & \text{if } x = 0 \wedge y \text{ is even} \\ \langle \ell, x = \mathit{zero},\ y = \mathit{odd} \rangle & \text{if } x = 0 \wedge y \text{ is odd} \end{cases}$$

Ex.: define the abstract operations

- To obtain an abstract transition system TS', with TS simulated by TS', the operations in TS must be replaced with abstract operations that yield values from the abstract domains.
- y = y + 1 is replaced by
$$y \mapsto \begin{cases} even & \text{if } y = odd \\ odd & \text{if } y = even \end{cases}$$
- x = x − 1 depends on the value of x, not known in the abstraction. Therefore, the statement is replaced by a nondeterministic choice:

$$x := gzero \quad \textbf{or} \quad x := zero$$

Final Result: an abstract program

```
1. while (x = gzero) {
2.         x = gzero or x = zero;
3.         if (y = even) y = odd; else y = even;
   }
4. if (y = even) return 1;
5. return 0;
```

- The abstract program originates from syntactic transformations which can be completely automated once the abstraction function has been defined.
- This program (if starting from a corresponding initial state) simulates the original program.
- Is this program «equivalent» to the concrete one?
- Of course not! but some properties can be proved on the abstract program and will be valid also in the concrete one.

- **BUT WHICH PROPERTIES?**

Safety properties!

- Simulation Preserves Safety Properties
- Let Psafe be a safety LT property and TS1 and TS2 transition systems (all over AP); then:

$$TS_1 \preceq TS_2 \quad and \quad TS_2 \models P_{safe} \quad implies \quad TS_1 \models P_{safe}.$$

- NB: Simulation is not an equivalence, so if TS2 is not «safe» then still TS1 might be.
- Safety properties are preserved because finite paths *fragments* are preserved.

Is Trace Inclusion Always Preserved?

Consider the transition systems TS_1 (left) and TS_2 (right) depicted in Figure 7.26. We have $TS_1 \preceq TS_2$, but $Traces(TS_1) \not\subseteq Traces(TS_2)$ since $\{ a \} \varnothing \in Traces(TS_1)$ but $\{ a \} \varnothing \notin Traces(TS_1)$. This is due to the fact that $s_2 \preceq t_2$, but whereas s_2 is a terminal state, t_2 is

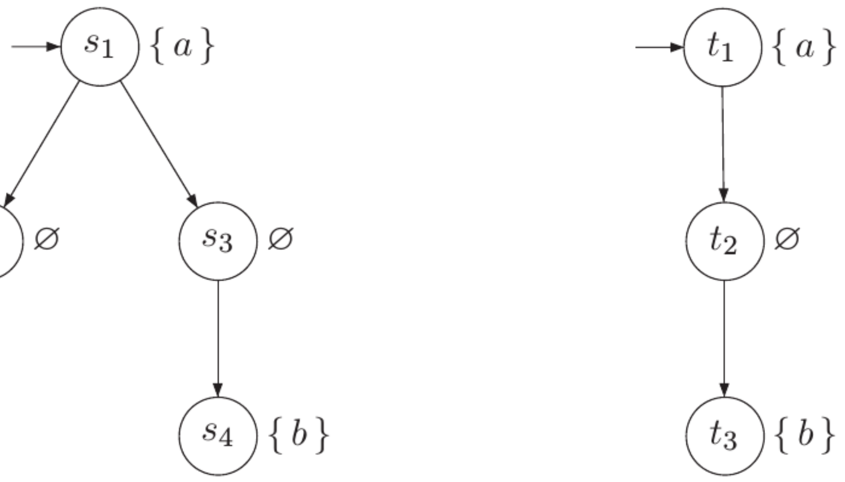


Figure 7.26: $TS_1 \preceq TS_2$, but $Traces(TS_1) \not\subseteq Traces(TS_2)$.

not. (Note that terminal states are simulated by any equally labeled state, be it terminal or not). As a result, a trace in TS_1 may end in s_2, but a trace in TS_2 cannot end in t_2. ■

Simulation on path *fragments*

$$
\begin{array}{ccc}
s_1 & -\mathcal{R}- & s_2 \\
\downarrow & & \\
s_{1,1} & & \\
\downarrow & & \\
s_{2,1} & & \\
\downarrow & & \\
\vdots & & \\
\downarrow & & \\
s_{n,1} & &
\end{array}
\quad \begin{array}{c}\text{can be}\\ \text{completed to}\end{array} \quad
\begin{array}{ccc}
s_1 & -\mathcal{R}- & s_2 \\
\downarrow & & \downarrow \\
s_{1,1} & -\mathcal{R}- & s_{1,2} \\
\downarrow & & \downarrow \\
s_{2,1} & -\mathcal{R}- & s_{2,2} \\
\downarrow & & \downarrow \\
\vdots & & \vdots \\
\downarrow & & \downarrow \\
s_{n,1} & -\mathcal{R}- & s_{n,2}
\end{array}
$$

Figure 7.22: Path fragment lifting for simulations.

- A finite path *fragment* p1 from s1 is simulated by a path *fragment* p2 from s2;
  However, if p1 ends in a terminal state and p2 does not, then p1 is a path from s2 but p2 is NOT a path from s2.
- Simulation preserves the set of all finite path fragments (from initial states), but not the set of paths ending in a terminal state
- Finite traces are defined as the traces corresponding to finite paths fragments: finite traces are preserved as well

Trace Inclusion with no terminal states!

- Terminal states in the simulated program TS1 are the problem. If TS1 has no terminal states trace inclusion is preserved for all traces

$$\textit{If } TS_1 \preceq TS_2 \textit{ and } TS_1 \textit{ does not have terminal states then } \mathrm{Traces}(TS_1) \subseteq \mathrm{Traces}(TS_2).$$

- As a corollary, *for transition systems without terminal states*, simulation preserves all LT properties and not just the safety properties.

Practical meaning of P1 simulated by P2

- Many *abstraction* techniques (e.g. SLAM, CBMC) based on building a P2 simulating P1, and then only verifying P2
- R*efinement* approaches build and verify P2 (i.e., a model), before implementing an actual program P1
- For sequential, non-reactive, programs we are only interested in *safety properties*
  - If implementation P1 is simulated by a more abstract P2 then safety properties proved on P2 («for the same AP») are preserved in P1
- For concurrent, reactive program we are also interested in liveness property (e.g., no deadlock, no starvation)
  - Typically we can consider the program P1 as nonterminating
  - Liveness properties proved on P2 are then preserved in P1 (trace inclusion!)

Very simple example: a terminating program

Program1

```
1: while x > 0 {
2:     x = x − 1;
3:     y = y + 1;
}
4: if (even(y)) return 1;
5: return 0;
```

Program2

```
1. while (x = gzero) {
2.         x = gzero or x = zero;
3.         if (y = even) y = odd; else y = even;
}
4. if (y = even) return 1;
5. return 0;
```

- We want prove a safety property of Program1:
  - *When selecting a value of x>0 and any value for y, there is a path returning 1 and there is a path returning 0.*
- We can just prove the following abstract property on Program2, and result is guaranteed to be valid on Program1.
  - *When selecting x=gzero and any y=even or y=odd, there is a path returning 1 and there is a path returning 0.*

Simulation equivalence

- Simulation relation is transitive and reflexive but not symmetric
  - if TS1 simulated by TS2, TS1$\leq$TS2, then it may not be the case that TS2$\leq$TS1 the vice versa)
- It may however happen then TS2 can actually be simulated by TS1, so TS2$\leq$TS1.
- TS1 and TS2 are **simulation equivalent**, TS1$\cong$TS2, if both TS1$\leq$TS2 and TS2$\leq$TS1.
  - Advantage: TS1 and TS2 verify the same safety properties

Example

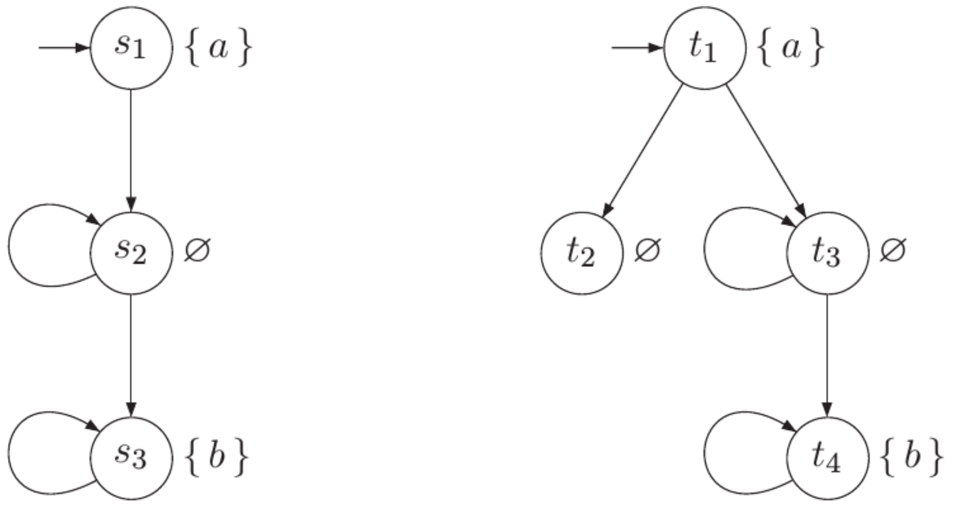


Figure 7.23: Simulation-equivalent transition systems.

- TS1 (left) is a subgraph of TS2 (right), up to isomorphism), so clearly TS1 $\leq$ TS2.
- *R* = { (t1, s1), (t2, s2), (t3, s2), (t4, s3) } is a simulation for (TS2, TS1), so TS2$\leq$ TS1

Simulation on states

- As it is the case for bisimulation, a simulation relation R can be defined on the states a *single* transition system TS.
  - Details are obivious
- State s_1 of TS is simulated by s_2, $s_1 \leq_{TS} s_2$, if there is a simulation R for a TS such $(s_1,s_2) \in R$.
- States s_1 and s_2 are **simulation-equivalent**, $s_1 \simeq_{TS} s_2$, if $s_1 \leq_{TS} s_2$ and $s_2 \leq_{TS} s_1$
  - NB: The simulation relation may be different in the two cases

Simulation Quotient

- As we did for bisimulation, given a transition system TS and a simulation equivalence $\simeq_{TS}$ on states we can define a «quotient» transition system $TS_{/\cong}$
- **For any transition system TS it holds that TS $\simeq_{TS}$ $TS_{/\cong}$**
- We can then reduce the size of a system using this quotient.
- But what is the relation with bisimulation equivalence?

Reminder

simulation order

$$s_1 \preceq_{TS} s_2 \quad :\Leftrightarrow \quad \text{there exists a simulation } \mathcal{R} \text{ for } TS \text{ with } (s_1, s_2) \in \mathcal{R}$$

simulation equivalence

$$s_1 \simeq_{TS} s_2 \quad :\Leftrightarrow \quad s_1 \preceq_{TS} s_2 \text{ and } s_2 \preceq_{TS} s_1$$

bisimulation equivalence

$$s_1 \sim_{TS} s_2 \quad :\Leftrightarrow \quad \text{there exists a bisimulation } \mathcal{R} \text{ for } TS \text{ with } (s_1, s_2) \in \mathcal{R}$$

Figure 7.24: Summary of the relations $\preceq_{TS}$, $\sim_{TS}$, and $\simeq_{TS}$.

Bisimulation is Strictly Finer than Simulation

$TS_1 \sim TS_2$ implies $TS_1 \simeq TS_2$, but $TS_1 \not\sim TS_2$ and $TS_1 \simeq TS_2$ is possible.

Example 7.63. Similar but not Bisimilar Transition Systems

Consider the transition systems TS_1 (left) and TS_2 (right) in Figure 7.25. $TS_1 \not\sim TS_2$, as there is no bisimilar state in TS_2 that mimics state s_2; the only candidate would be t_2, but s_2 cannot mimic $t_2 \to t_4$. TS_1 and TS_2 are, however, simulation-equivalent. As TS_2 is a subgraph (up to isomorphism) of TS_1, we obtain $TS_2 \preceq TS_1$. In addition, $TS_1 \preceq TS_2$ as

$$\mathcal{R} = \{ (s_1, t_1), (s_2, t_2), (s_3, t_2), (s_4, t_3), (s_5, t_4) \}$$

is a simulation relation for (TS_1, TS_2). ■

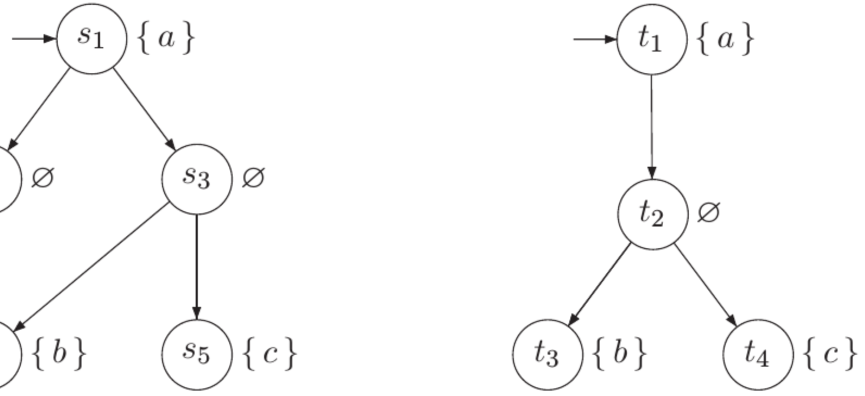
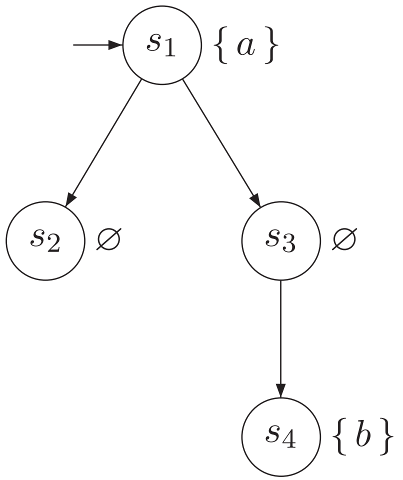


Figure 7.25: Simulation-, but not bisimulation-equivalent transition systems.

AP-determinism➔ simulation equiv. = bisim. equiv.

Transition system $TS = (S, Act, \to, I, AP, L)$ is *AP-deterministic* if

1. for $A \subseteq AP$: $|I \cap \{ s \mid L(s) = A \}| \leqslant 1$, and
2. for $s \in S$: if $s \xrightarrow{\alpha} s'$ and $s \xrightarrow{\alpha} s''$ and $L(s') = L(s'')$, then $s' = s''$.



TS is not AP-deterministic as its initial state has two distinct Ø-successors.

Theorem 7.66. ***AP-Determinism Implies $\sim$ and $\simeq$ Coincide***

If TS_1 and TS_2 are AP-deterministic, then $TS_1 \sim TS_2$ if and only if $TS_1 \simeq TS_2$.

Summary: Relation among equivalences

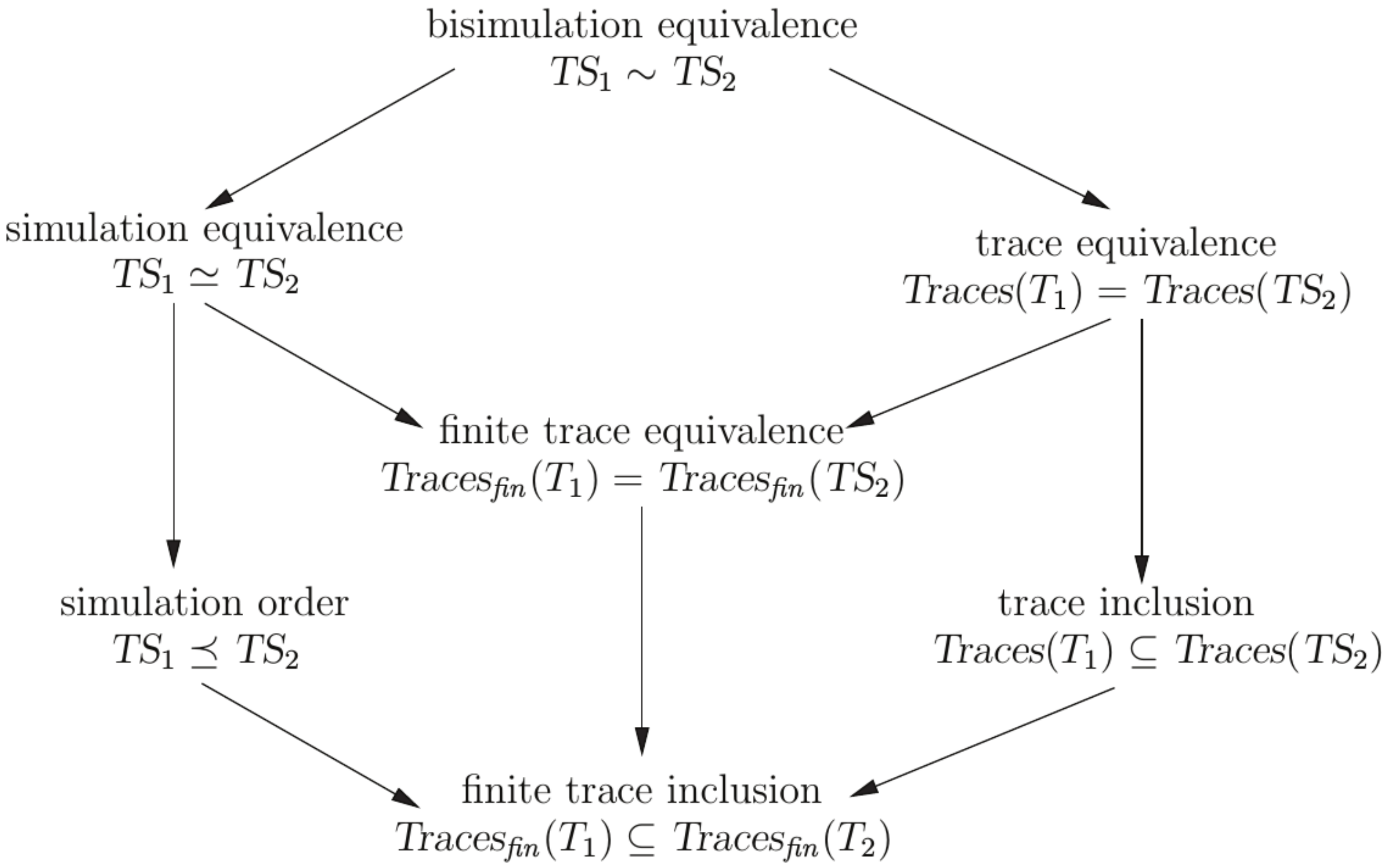


Figure 7.27: Relation between equivalences and preorders on transition systems.

What about actions?

- Our definition considered only state labels and ignored actions!
- This is consistent with our interest in model checking, where labels of transition are irrelevant.
  - For instance, in a temporal logic formula we only talk of set AP.
  - An external event triggering a transition can be «stored» in the state as part of the state label.
  - Actions are used mainly for communicating processes
- It is possible to define an «action-based» bisimulation.
- The two notions are of course strictly related.
- Action-based bsimulations are studied extensively in concurrency theory («process algebras» such as CSP, labeled transition systems).

Action-based bisimulation

Definition 7.15. Action-Based Bisimulation Equivalence

Let $TS_i = (S_i, Act, \rightarrow_i, I_i, AP_i, L_i)$, $i{=}1,2$, be transition systems over the set Act of actions. An *action-based bisimulation* for (TS_1, TS_2) is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ such that

(A) $\forall s_1 \in I_1 \, \exists s_2 \in I_2. \, (s_1, s_2) \in \mathcal{R}$ and $\forall s_2 \in I_2 \, \exists s_1 \in I_1. \, (s_1, s_2) \in \mathcal{R}$

(B) for any $(s_1, s_2) \in \mathcal{R}$ it holds that

- (2') if $s_1 \xrightarrow{\alpha}_1 s_1'$, then $s_2 \xrightarrow{\alpha}_2 s_2'$ with $(s_1', s_2') \in \mathcal{R}$ for some $s_2' \in S_2$
- (3') if $s_2 \xrightarrow{\alpha}_2 s_2'$, then $s_1 \xrightarrow{\alpha}_1 s_1'$ with $(s_1', s_2') \in \mathcal{R}$, for some $s_1' \in S_1'$.

TS_1 and TS_2 are *action-based bisimulation equivalent* (or action-based bisimilar), denoted $TS_1 \sim^{Act} TS_2$, if there exists an action-based bisimulation $\mathcal{R}$ for (TS_1, TS_2). ■

All results and concepts presented for $\sim$ can be adapted for $\sim^{\text{Act}}$ in a straightforward manner. For instance, $\sim^{\text{Act}}$ is an equivalence and can be adapted to an equivalence $\sim^{\text{Act}}_{\text{TS}}$ also for the states of a single transition system TS.

Definition can be used also for defining bisim. for NFA (adding condition that final states are not equivalent to nonfinal states).