Formal Methods for System Design

Chapter 2: Modeling systems

Mickael Randour

Mathematics Department, UMONS

September 2021

UMONS
Université de Mons

1. Transition systems

2. Comparing TSs: why, how, graph isomorphism, trace equivalence

3. Bisimulation

4. Simulation

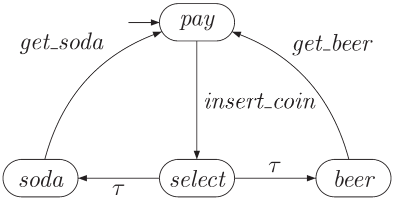
1 Transition systems

2 Comparing TSs: why, how, graph isomorphism, trace equivalence

3 Bisimulation

4 Simulation

Transition system



Transition system for a (rather stupid) beverage vending machine [BK08].

- Model describing the behavior of a system.
- Directed graphs: vertices = *states*, edges = *transitions*.
- **State**: current mode of the system, current values of program variables, current color of a traffic light...
- **Transition** as atomic actions: mode switching, execution of a program instruction, change of color...

Formal definition

Definition: Transition system (TS)

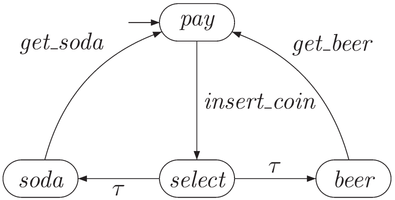
Tuple $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$ with

- S the set of states,
- Act the set of actions,
- $\longrightarrow \subseteq S \times Act \times S$ the transition relation,
- $I \subseteq S$ the set of initial states,
- AP the set of atomic propositions, and
- $L\colon S \rightarrow 2^{AP}$ the labeling function.

We often consider *finite* TSs, i.e., $|S|, |Act|, |AP| < \infty$, but not necessarily true in general.

Notation: sometimes we write $s \xrightarrow{\alpha} s'$ instead of $(s, \alpha, s') \in \longrightarrow$.

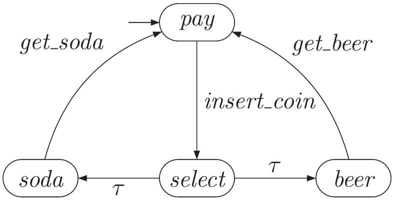
Back to the example



- $S = \{\mathit{pay}, \mathit{select}, \mathit{beer}, \mathit{soda}\}$,
- $Act = \{\mathit{insert\_coin}, \mathit{get\_beer}, \mathit{get\_soda}, \tau\}$,
- Some transitions: $\mathit{pay} \xrightarrow{\mathit{insert\_coin}} \mathit{select}$, $\mathit{select} \xrightarrow{\tau} \mathit{beer}$.
- $I = \{\mathit{pay}\}$,

What about the labeling?

Back to the example

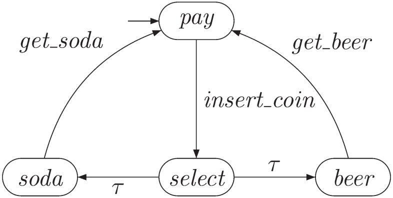


Depends on what we want to model!

- Simple choice: $\forall\, s, L(s) = \{s\}$.
- Say the property is "the vending machine only delivers a drink after providing a coin"
  $\hookrightarrow$ $AP = \{paid, drink\}$, $L(pay) = \emptyset$, $L(select) = \{paid\}$ and $L(soda) = L(beer) = \{paid, drink\}$.

$\Rightarrow$ **useful to model check logic formulae.**

Back to the example



- $\hookrightarrow$ When the labeling is not important, we often omit it.
- $\hookrightarrow$ We do the same for actions or simply use *internal actions* (τ).

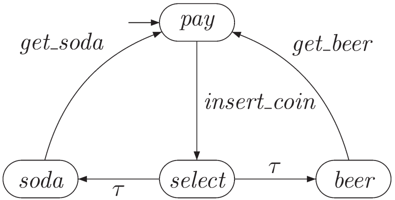
Actions are often used to model communication mechanism (e.g., parallel processes).

Related models

We talk about **transition systems (TSs)** and adopt the definition of [BK08]. Equivalent models are often used in the literature.

- *Kripke structure (KS)* $\sim$ TS without labels on actions.
- *Labeled transition system (LTS)* $\sim$ TS without labels on states.

Semantics of TSs: non-determinism



When two actions are possible (*select*), the choice is made **non-deterministically**!

Also true for the initial state if $|I| > 1$.

$\hookrightarrow$ Meaningful to model *interleaving* of $\parallel$ executions for example.

$\hookrightarrow$ Also for *abstraction* or to model an *uncontrollable environment* (here, drink choice by the user).

Basic concepts: predecessors and successors

Let $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$ be a TS. For $s \in S$ and $\alpha \in Act$, we define the following sets.

Direct (α-)successors of s:

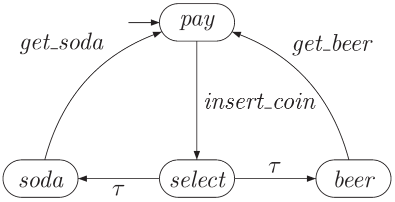
$$Post(s, \alpha) = \left\{ s' \in S \mid s \xrightarrow{\alpha} s' \right\}, \quad Post(s) = \bigcup_{\alpha \in Act} Post(s, \alpha).$$

Direct (α-)predecessors of s:

$$Pre(s, \alpha) = \left\{ s' \in S \mid s' \xrightarrow{\alpha} s \right\}, \quad Pre(s) = \bigcup_{\alpha \in Act} Pre(s, \alpha).$$

$+$ natural extensions to subsets of S.

Back to the example



Some examples:

- $Post(\textsf{select}) = \{\textsf{soda}, \textsf{beer}\}$,
- $Pre(\textsf{pay}, \textsf{get\_beer}) = \{\textsf{beer}\}$,
- $Post(\textsf{beer}, \tau) = \emptyset$.

Terminal states

A state $s \in S$ is called terminal iff $Post(s) = \emptyset$.

$\hookrightarrow$ For *reactive systems*, those states should in general be avoided.

$\Rightarrow$ **deadlocks**

Basic concepts: executions (1/2)

Let $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$ be a TS.

Finite execution fragment:

$\varrho = s_0\alpha_1 s_1 \alpha_2 \ldots \alpha_n s_n$ such that $s_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} s_n$.

Infinite execution fragment:

$\rho = s_0\alpha_1 s_1 \alpha_2 \ldots$ such that $s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$ for all $i \geq 0$.

Maximal execution fragment:

Fragment that cannot be prolonged.

Initial execution fragment:

Fragment starting in $s_0 \in I$.

Basic concepts: executions (2/2)

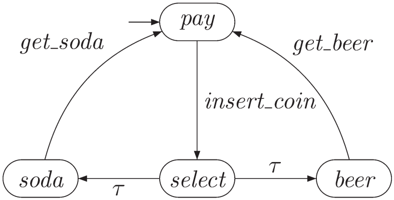
Execution:

Initial *and* maximal execution fragment.

Reachable states:

$$\begin{aligned} Reach(\mathcal{T}) &= \left\{ s \in S \mid \exists\, s_0 \in I \wedge\, s_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} s_n = s \right\} \\ &= Post^*(I) \end{aligned}$$

Back to the example



Some examples.

- $\rho_1 = pay \xrightarrow{insert\_coin} select \xrightarrow{\tau} beer \xrightarrow{get\_beer} pay \xrightarrow{insert\_coin} \ldots$
  $\hookrightarrow$ ρ_1 is an execution.
- $\rho_2 = beer \xrightarrow{get\_beer} pay \xrightarrow{insert\_coin} select \xrightarrow{\tau} beer \xrightarrow{get\_beer} \ldots$
  $\hookrightarrow$ ρ_2 is not (maximal but not initial).
- $\varrho_3 = pay \xrightarrow{insert\_coin} select \xrightarrow{\tau} soda \xrightarrow{get\_soda} pay$
  $\hookrightarrow$ ϱ_3 is not (initial but not maximal).
- $Reach(\mathcal{T}) = S$.

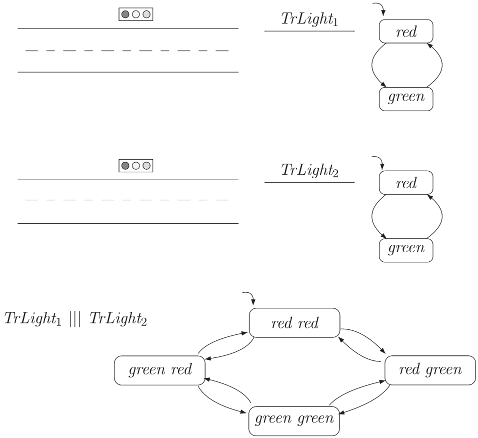
Modeling systems

The reference book [BK08] contains different examples illustrating how to construct formal models from real applications or segments of program code.

$\Rightarrow$ **We survey some of them in the following.**

$\Rightarrow$ **Focus on concurrency: prone to errors.**

Independent traffic lights on non-intersecting roads



Interleaving semantics [BK08].

- Concurrency is represented by **interleaving**.
  - ▷ Non-deterministic choice between activities of simultaneously acting processes.
  - ▷ In general, needs to be complemented with **fairness** assumptions.

Mutex with semaphores (1/3)

P_i **loop forever**

$\vdots$ (* noncritical actions *)

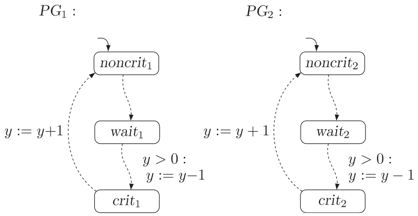
request

critical section

release

$\vdots$ (* noncritical actions *)

end loop



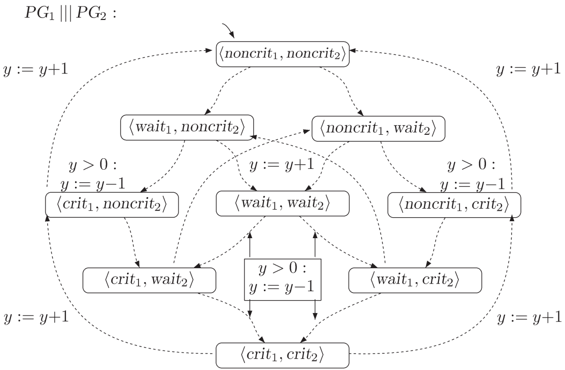
Program graphs for semaphore-based mutex [BK08].

- **Program graphs (PGs)** retain conditional transitions.

$\hookrightarrow$ Interleaving must be done at this level to deal with *shared variables*.

$\Rightarrow$ Then we consider the TS $\mathcal{T}(PG_1 \interleave PG_2)$.

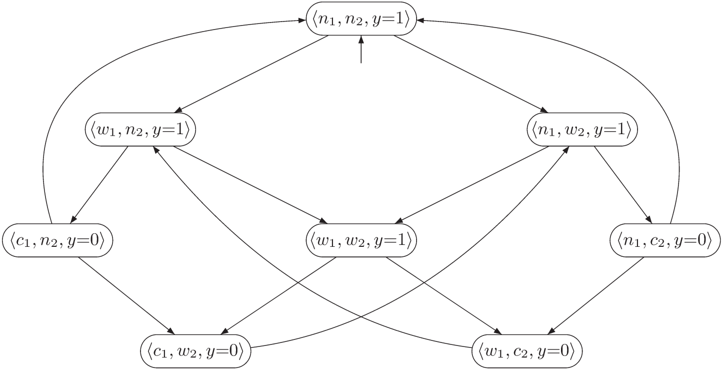
Mutex with semaphores (2/3)



$PG_1 \,|||\, PG_2$ for semaphore-based mutex [BK08].

The TS unfolding will tell us if $\langle \text{crit}_1, \text{crit}_2 \rangle$ is reachable (which we want to avoid obviously).

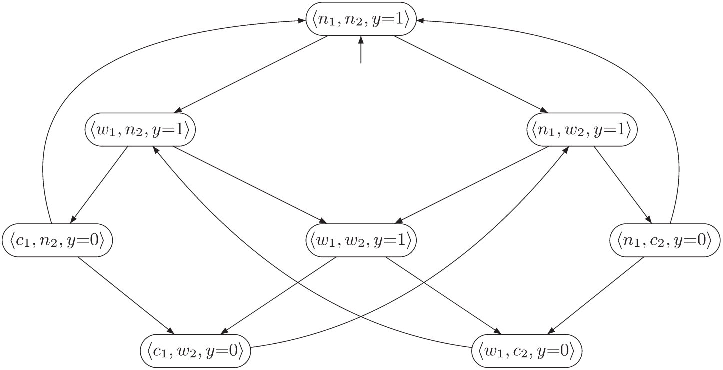
Mutex with semaphores (3/3)



$\mathcal{T}(PG_1 \mid\mid\mid PG_2)$ *for semaphore-based mutex [BK08].*

Mutual exclusion is verified:
$\langle c_1, c_2, y = \ldots\rangle \notin Reach(\mathcal{T}(PG_1 \mid\mid\mid PG_2))$.

Mutex with semaphores (3/3)



$\mathcal{T}(PG_1 \,|||\, PG_2)$ *for semaphore-based mutex [BK08].*

The scheduling problem in $\langle \mathbf{w}_1, \mathbf{w}_2, y = 1\rangle$ is left open.
$\hookrightarrow$ implement a discipline later (LIFO, FIFO, etc) or use an algorithm solving the issue explicitly: **Peterson's mutex**.

Peterson's mutex algorithm (1/2)

P_1 **loop forever**

⋮ (* noncritical actions *)

$\langle b_1 := \text{true};\ x := 2\rangle$; (* request *)

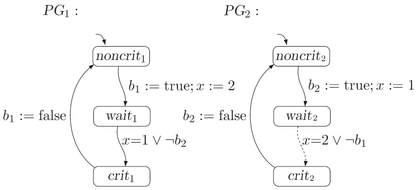
wait until $(x = 1\ \vee\ \neg b_2)$

do critical section **od**

$b_1 := \text{false}$ (* release *)

⋮ (* noncritical actions *)

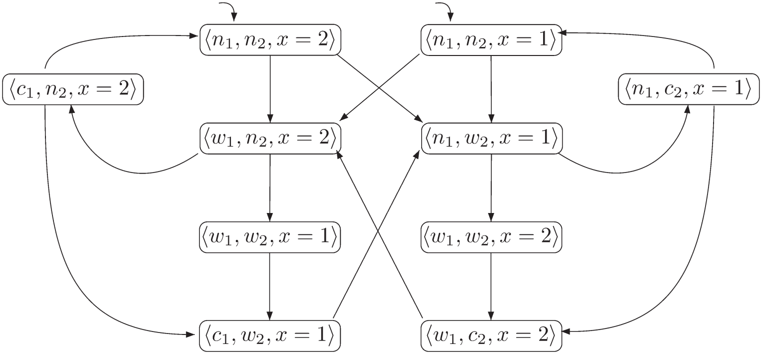
end loop



Program graphs for Peterson's mutex [BK08].

⇒ The value of x determines who will enter the critical section.

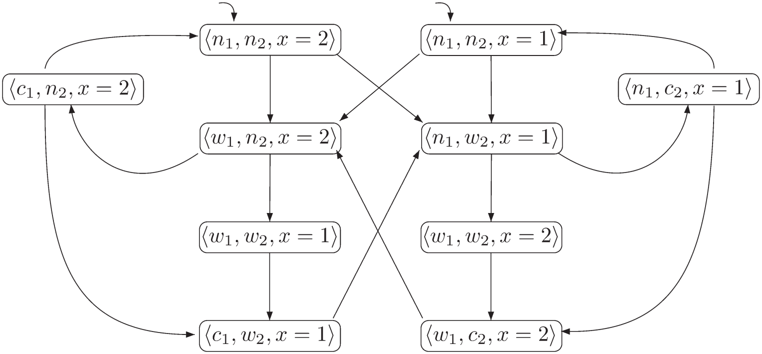
Peterson's mutex algorithm (2/2)



$\mathcal{T}(PG_1 \mid\mid\mid PG_2)$ for Peterson's mutex [BK08].

Mutual exclusion is verified:
$\langle c_1, c_2, x = \ldots\rangle \notin Reach(\mathcal{T}(PG_1 \mid\mid\mid PG_2))$.

Peterson's mutex algorithm (2/2)



$\mathcal{T}(PG_1 \mid\mid\mid PG_2)$ for Peterson's mutex [BK08].

Peterson's also has **bounded waiting**, hence **fairness** is satisfied.

Not true for semaphore-based (without discipline): processes could starve.

The state(-space) explosion problem

Verification techniques operate on TSs obtained from programs or program graphs. Their size can be **huge**, or they can even be **infinite**. Some sources:

- **Variables**
  - ▷ PG with 10 locations, three Boolean variables and five integers in $\{0, \ldots, 9\}$ already contains $10 \cdot 2^3 \cdot 10^5 = 8.000.000$ states.
  - ▷ Variable in infinite domain $\Rightarrow$ infinite TS!
- **Parallelism**
  - ▷ $\mathcal{T} = \mathcal{T}_1 \interleave \ldots \interleave \mathcal{T}_n \;\Rightarrow\; |S| = |S_1| \cdot \ldots \cdot |S_n|$.

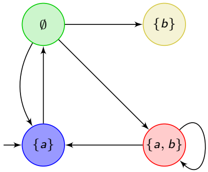
    $\hookrightarrow$ **Exponential blow-up!**

$\Rightarrow$ Need for (a lot of) **abstraction** and efficient **symbolic** techniques (Ch. 5) to keep the verification process tractable.

Why?

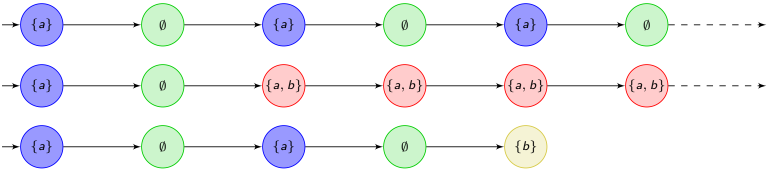
- To see if two TSs are *similar*.
  - ▷ Is one a **refinement** or an **abstraction** of the other?
  - ▷ Are the two *indistinguishable* w.r.t. observable properties?
- To be able to *model check large systems*.
  - ▷ If $\mathcal{T}_1$ is a small abstraction of $\mathcal{T}_2$ that preserves the property to be checked, then model checking $\mathcal{T}_1$ is more efficient!
  - ↪ Can help for large or infinite systems: not all complexity is necessary!
- What does it mean to *preserve a property*?
  - ▷ Each type of relation preserves a different logical fragment (intuitively, a different kind of properties).
  - ↪ Depends on what we are interested in.

Linear time vs. branching time semantics (1/2)

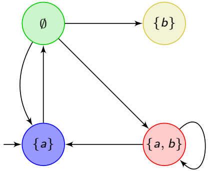


TS $\mathcal{T}$ with state labels $AP = \{a, b\}$ (state and action names are omitted).

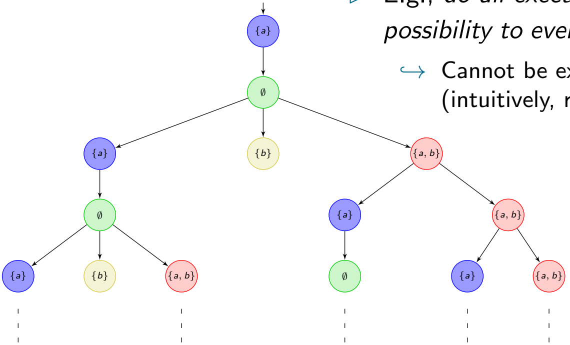
- **Linear time semantics** deals with *traces* of executions.
  - ▷ The language of (in)finite words described by $\mathcal{T}$.
  - ▷ See **LTL** in Ch. 3.
  - ▷ E.g., *do all executions eventually reach* $\{b\}$*?* No.



Linear time vs. branching time semantics (2/2)



- **Branching time semantics** deals with the *execution tree*.
  - ▷ Infinite unfolding considering all branching possibilities.
  - ▷ See **CTL** in Ch. 4.
  - ▷ E.g., *do all executions always have the possibility to eventually reach* {b}? Yes.
    - ↪ Cannot be expressed as a LT property (intuitively, requires *branching*).



Which type of relation between TSs should we use?

- **Linear time properties** (e.g., LTL)
  - $\Rightarrow$ **Trace equivalence/inclusion** is an obvious choice.
  - ⚠ But **language inclusion is costly!** (PSPACE-complete)
  - $\hookrightarrow$ Other relations provide a *more efficient alternative* (P-complete).

- **Branching time semantics** (e.g., CTL)
  - $\Rightarrow$ **Bisimulation**: related states can mutually mimic all individual transitions.
  - $\Rightarrow$ **Simulation**: one state can mimic all stepwise behaviors of the other, but the reverse is not necessary.

In the following, we assume state-based labeling and often that there is no deadlock ($\rightsquigarrow$ self-loops otherwise).

Graph isomorphism (1/2)

Idea: isomorphism up to renaming of the states and actions.

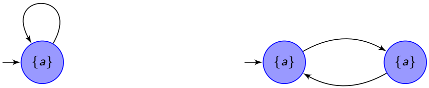
Definition: TS isomorphism

$\mathcal{T}_1 = (S_1, Act_1, \longrightarrow_1, I_1, AP_1, L_1)$ and
$\mathcal{T}_2 = (S_2, Act_2, \longrightarrow_2, I_2, AP_2, L_2)$ are isomorphic if there exists a bijection f such that

- $S_2 = f(S_1)$,
- $Act_2 = f(Act_1)$,
- $s \xrightarrow{\alpha}_1 s' \iff f(s) \xrightarrow{f(\alpha)}_2 f(s')$,
- $s \in I_1 \iff f(s) \in I_2$,
- $AP_1 = AP_2$,
- $\forall\, s \in S_1,\ L_1(s) = L_2(f(s))$.

Preserves properties but **much too restrictive!**

Graph isomorphism (2/2)



Those TSs are clearly "equivalent" (i.e., indistinguishable for meaningful properties) but *are not isomorphic*.

⇒ Graph isomorphism is not interesting for model checking.

Trace inclusion and trace equivalence (1/6)

What is a trace?

▷ An execution seen through its labeling.

Definition: paths and traces

Let $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$ be a TS and $\rho = s_0\alpha_1 s_1\alpha_2 \ldots$ one of its executions:

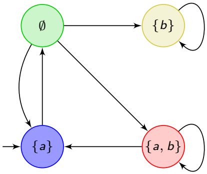
- its *path* is $\pi = path(\rho) = s_0 s_1 s_2 \ldots$,
- its *trace* is $trace(\pi) = L(\pi) = L(s_0)L(s_1)L(s_2)\ldots$

We denote $Paths(\mathcal{T})$ (resp. $Traces(\mathcal{T})$) the set of all paths (resp. traces) in $\mathcal{T}$.

Defined for *executions* (i.e., maximal and initial fragments), but also for fragments starting in a state s ($Paths(s)$ and $Traces(s)$) or a subset of states $S' \subseteq S$ ($Paths(S')$ and $Traces(S')$), as well as for *finite* fragments ($Paths_{fin}$ and $Traces_{fin}$).

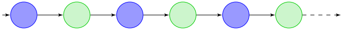
Trace inclusion and trace equivalence (2/6)

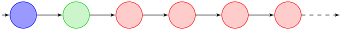
Example

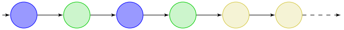


- Notice the added self-loop on 

- Paths:

$\pi_1 =$ 

$\pi_2 =$ 

$\pi_3 =$ 

- Corresponding traces:

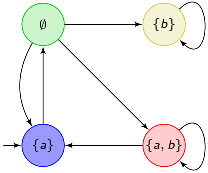
$$trace(\pi_1) = \{a\}\emptyset\{a\}\emptyset\{a\}\emptyset\ldots = (\{a\}\emptyset)^\omega$$
$$trace(\pi_2) = \{a\}\emptyset\{a,b\}\{a,b\}\{a,b\}\{a,b\}\ldots = \{a\}\emptyset\{a,b\}^\omega$$
$$trace(\pi_3) = \{a\}\emptyset\{a\}\emptyset\{b\}\{b\}\ldots = \{a\}\emptyset\{a\}\emptyset\{b\}^\omega$$

Traces are (infinite) words on alphabet 2^{AP}.

$\hookrightarrow$ **alphabet exponential in $|AP|$.**

Trace inclusion and trace equivalence (3/6)

Example (cont'd)



Which languages does this TS describe?

- Finite traces:

$$Traces_{fin}(\mathcal{T}) = \{a\}(\emptyset\{a,b\}^*\{a\})^*\Big[\varepsilon \,\Big|\, \emptyset\big(\{b\}^*|\{a,b\}^*\big)\Big]$$

- Traces:

$$R = (\emptyset\{a,b\}^*\{a\})$$
$$Traces(\mathcal{T}) = \{a\}R^*\Big[R^\omega \,\Big|\, (\emptyset\{a,b\}^\omega) \,\Big|\, \emptyset\{b\}^\omega\Big]$$

Trace inclusion and trace equivalence (4/6)

Trace inclusion

- Linear-time (LT) properties (e.g., LTL) specify which traces a TS should exhibit.
- Trace inclusion $\sim$ implementation relation.

$\mathit{Traces}(\mathcal{T}) \subseteq \mathit{Traces}(\mathcal{T}')$ means $\mathcal{T}$ "is a correct implementation of" $\mathcal{T}'$.

$\hookrightarrow$ $\mathcal{T}$ is seen as a refinement/implementation of the more abstract model $\mathcal{T}'$.

Theorem: trace inclusion and LT properties

Let $\mathcal{T}$ and $\mathcal{T}'$ be two TSs without terminal states and with the same set of propositions AP. The following statements are equivalent:

(a) $\mathit{Traces}(\mathcal{T}) \subseteq \mathit{Traces}(\mathcal{T}')$

(b) For any LT property P: $\mathcal{T}' \models P \Longrightarrow \mathcal{T} \models P$.

Trace inclusion and trace equivalence (5/6)

Trace inclusion (cont'd) and equivalence

Thus, **trace inclusion preserves LTL properties**.

▷ Useful when refining systems: automatic proof of correctness for the refined system.

We can go further and consider *trace equivalence*.

Theorem: trace equivalence and LT properties

Let $\mathcal{T}$ and $\mathcal{T}'$ be two TSs without terminal states and with the same set of propositions AP. Then:

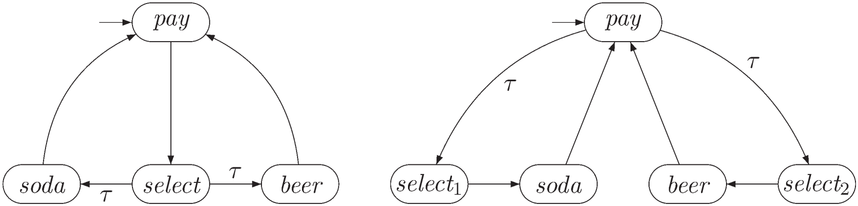
$$Traces(\mathcal{T}) = Traces(\mathcal{T}')$$
$$\Updownarrow$$
$\mathcal{T}$ and $\mathcal{T}'$ satisfy the same LT properties.

But, **testing trace inclusion/equivalence is costly!**

▷ PSPACE-complete (i.e., in pratice requires exponential time).

Trace inclusion and trace equivalence (6/6)

Example



Trace-equivalent systems [BK08].

For $AP = \{pay, soda, beer\}$, those TSs are *trace-equivalent*.

$\hookrightarrow$ They are indistinguishable by LT properties.

Idea

Goal

Identify TSs with the same branching structure.

Intuitively: $\mathcal{T}$ is bisimilar to $\mathcal{T}'$ if both TSs can simulate each other in a mutual, stepwise manner.

Definition

Definition: bisimulation equivalence

Let $\mathcal{T}_i = (S_i, Act_i, \longrightarrow_i, I_i, AP, L_i)$, $i = 1, 2$, be TSs over AP.
A **bisimulation** for $(\mathcal{T}_1, \mathcal{T}_2)$ is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ s.t.

(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) $L_1(s_1) = L_2(s_2)$
- (2) $s_1' \in Post(s_1) \Longrightarrow \big(\exists\, s_2' \in Post(s_2)\ \wedge\ (s_1', s_2') \in \mathcal{R}\big)$
- (3) $s_2' \in Post(s_2) \Longrightarrow \big(\exists\, s_1' \in Post(s_1)\ \wedge\ (s_1', s_2') \in \mathcal{R}\big)$.

$\mathcal{T}_1$ and $\mathcal{T}_2$ are bisimulation-equivalent, or *bisimilar*, denoted $\mathcal{T}_1 \sim \mathcal{T}_2$, if there exists a bisimulation $\mathcal{R}$ for $(\mathcal{T}_1, \mathcal{T}_2)$.

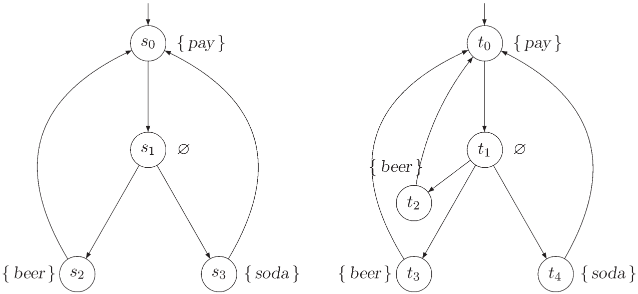
Illustration

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

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

Conditions (B.2) and (B.3) of bisimulation equivalence [BK08].

Examples

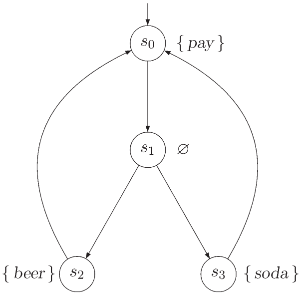


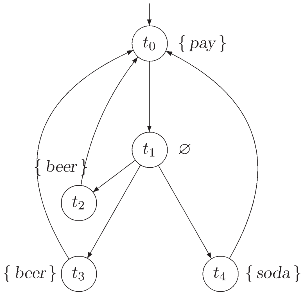
Bisimilar beverage vending machines [BK08].

▷ Intuitively, the additional option to deliver beer in $\mathcal{T}_2$ is not observable by users.

↪ Equivalence in terms of observable behaviors.

Examples



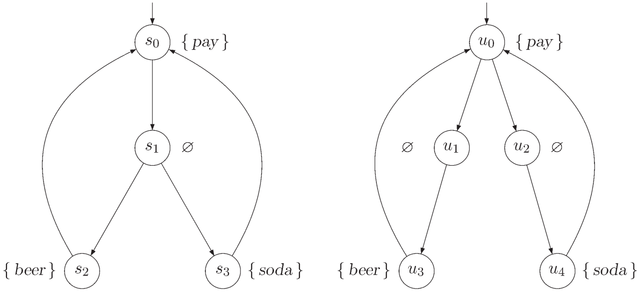


Bisimilar beverage vending machines [BK08].

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

$\Longrightarrow$ **Blackboard proof.**

Examples (cont'd)

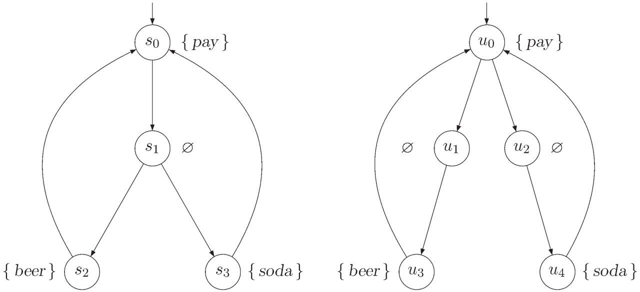


Non-bisimilar beverage vending machines [BK08].

State s_1 cannot be mimicked! Candidates are u_1 and u_2 but they do not satisfy condition (B.2).

- ▷ $u_1 \not\rightarrow \mathit{soda}$ and $u_2 \not\rightarrow \mathit{beer}$.
- ▷ $\mathcal{T}_1 \not\sim \mathcal{T}_3$ for $AP = \{\mathit{pay}, \mathit{beer}, \mathit{soda}\}$.

Examples (cont'd)

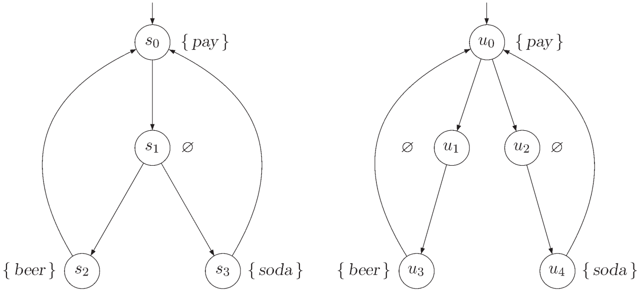


Non-bisimilar beverage vending machines [BK08].

What if we take a more abstract labeling $AP = \{pay, drink\}$?

- $L(s_0) = L(t_0) = \{pay\}$, $L(s_1) = L(u_1) = L(u_2) = \emptyset$, all other labels $= \{drink\}$.

Examples (cont'd)



Non-bisimilar beverage vending machines [BK08].

Then, bisimulation $\mathcal{R} = \{(s_0, u_0), (s_1, u_1), (s_1, u_2), (s_2, u_3), (s_2, u_4), (s_3, u_3), (s_3, u_4)\}$.

▷ $\mathcal{T}_1 \sim \mathcal{T}_3$ for $AP = \{pay, drink\}$.

$\Longrightarrow$ **Blackboard proof.**

Properties (1/3)

Equivalence

Bisimulation is an equivalence relation

For a fixed set AP of propositions, the bisimulation relation $\sim$ is an equivalence relation, i.e., it is reflexive, transitive and symmetric.

- Reflexivity: $\mathcal{T} \sim \mathcal{T}$.
- Transitivity: $\mathcal{T} \sim \mathcal{T}' \wedge \mathcal{T}' \sim \mathcal{T}'' \Longrightarrow \mathcal{T} \sim \mathcal{T}''$.
- Symmetry: $\mathcal{T} \sim \mathcal{T}' \Longleftrightarrow \mathcal{T}' \sim \mathcal{T}$.

$\Longrightarrow$ **Exercise.**

Properties (2/3)

Linear-time properties

Bisimulation and trace equivalence

$\mathcal{T}_1 \sim \mathcal{T}_2 \implies \mathit{Traces}(\mathcal{T}_1) = \mathit{Traces}(\mathcal{T}_2)$

$\hookrightarrow$ $\mathcal{T}_1$ and $\mathcal{T}_2$ satisfy the same LT properties.

$\hookrightarrow$ Will be an interesting alternative to trace equivalence complexity-wise as bisimulation can be checked in polynomial time.

The converse is false!

$\hookrightarrow$ Recall previous example of non-bisimilar beverage vending machines (same language but not bisimilar).

Properties (3/3)

Branching-time properties

One can show that **bisimulation also preserves branching-time properties** (e.g., CTL).

Quotienting (1/7)

Idea

Idea

1. See bisimulation as a relation between states of a *single* TS.
2. Quotient the TS by this relation.
   - ▷ Obtain a smaller TS that preserves properties.
3. Model check the smaller TS.
   - ▷ More efficient! (quotienting is "cheap" in comparison to model checking)

Quotienting (2/7)

Bisimulation on states

Definition: bisimulation equivalence as a relation on states

Let $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$ be a TS. A **bisimulation** for $\mathcal{T}$ is a binary relation $\mathcal{R}$ on $S \times S$ s.t. for all $(s_1, s_2) \in \mathcal{R}$:

(1) $L(s_1) = L(s_2)$

(2) $s_1' \in Post(s_1) \Longrightarrow \left(\exists\, s_2' \in Post(s_2) \ \wedge\ (s_1', s_2') \in \mathcal{R}\right)$

(3) $s_2' \in Post(s_2) \Longrightarrow \left(\exists\, s_1' \in Post(s_1) \ \wedge\ (s_1', s_2') \in \mathcal{R}\right)$.

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

Remark: equivalent to $\mathcal{T}_1 \sim \mathcal{T}_2$ with $\mathcal{T}_1 = \mathcal{T}_2 = \mathcal{T}$.

Remark: $\sim_{\mathcal{T}}$ is the *coarsest bisimulation* for $\mathcal{T}$ (i.e., yielding the largest $\mathcal{R}$, i.e., the fewer *equivalence classes*).

Quotienting (3/7)

Notations

Let S be a set and $\mathcal{R}$ an equivalence on S.

- **$\mathcal{R}$-equivalence class of** $s \in S$: $[s]_{\mathcal{R}} = \{s' \in S \mid (s, s') \in \mathcal{R}\}$.
  - ▷ $\forall\, s' \in [s]_{\mathcal{R}}$, $[s']_{\mathcal{R}} = [s]_{\mathcal{R}}$.
- **Quotient space of** S **under** $\mathcal{R}$: $S/\mathcal{R} = \{[s]_{\mathcal{R}} \mid s \in S\}$.
  - ▷ Set of all $\mathcal{R}$-equivalence classes.

Quotienting (4/7)

Bisimulation quotient

For simplicity, we write $\sim$ for $\sim_{\mathcal{T}}$ in the following.

Quotient

Let $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$ be a TS with (coarsest) bisimulation $\sim$. The **bisimulation quotient** of $\mathcal{T}$ is defined by

$$\mathcal{T}/\sim = (S/\sim, \{\tau\}, \longrightarrow', I', AP, L')$$

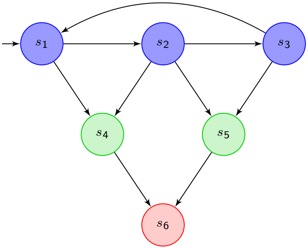
where:

- $I' = \{[s]_\sim \mid s \in I\}$,
- $s \xrightarrow{\alpha} s' \implies [s]_\sim \xrightarrow{\tau}' [s']_\sim$,
- $L'([s]_\sim) = L(s)$.

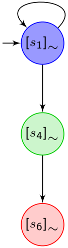
It is easily shown that $\mathcal{T} \sim \mathcal{T}/\sim$.

Quotienting (5/7)

Illustration



TS $\mathcal{T}$ (all labels $= \emptyset$)



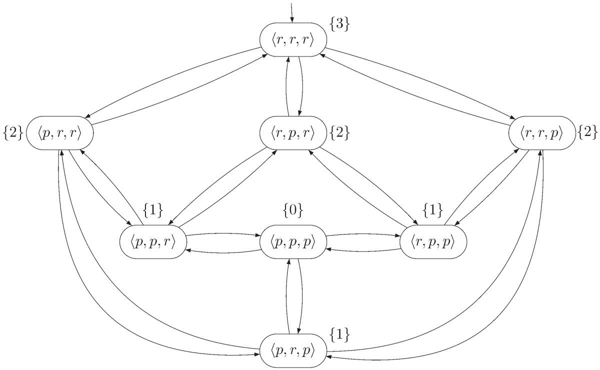
Bisimulation quotient $\mathcal{T}/\sim$

Each color = one $\mathcal{R}$-equivalence class.

$\Longrightarrow$ **Blackboard explanation: $\mathcal{R}$ is a bisimulation and quotienting.**

Quotienting (6/7)

Example: many printers (1/2)



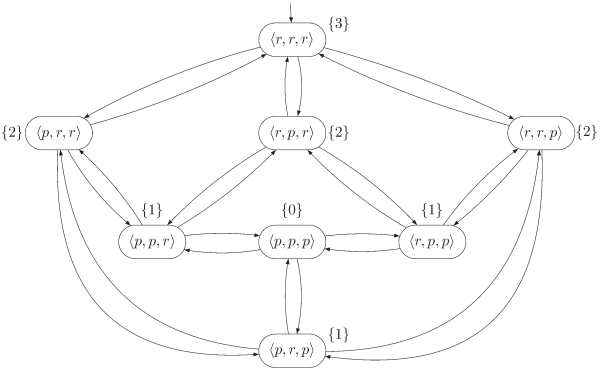
TS $\mathcal{T}_3$ for three printers [BK08].

System composed of n printers with two states: *ready* and *print*.

$\hookrightarrow$ Entire system $\mathcal{T}_n = \mathit{Printer} \,|||\, \dots \,|||\, \mathit{Printer}$.

Quotienting (6/7)

Example: many printers (1/2)

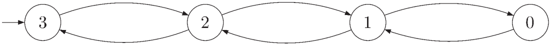


TS $\mathcal{T}_3$ for three printers [BK08].

- ▷ $AP = \{0, 1, \ldots, n\}$ (number of ready printers).
- ▷ $|\mathcal{T}_n| = 2^n \implies$ **exponential!** $\implies$ **let's quotient it!**

Quotienting (7/7)

Example: many printers (2/2)



Bisimulation quotient $\mathcal{T}_3/\sim$ [BK08].

- $\mathcal{R}$-equivalence classes based on number of available printers.
- $|\mathcal{T}_n/\sim| = n + 1$. $\Longrightarrow$ **now only linear!**

Quotienting can lead to huge gain in the model size while preserving needed properties.
$\Longrightarrow$ **powerful abstraction mechanism.**

It can even help in reducing infinite TSs to finite quotients. See *bakery algorithm* example in the book.

Quotienting algorithm (1/11)

Sketch

Goal

Given a TS $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$, compute its bisimulation quotient $\mathcal{T}/\sim$.

Partition-refinement technique.

$\hookrightarrow$ Partition state space S in *blocks*: pairwise disjoint sets of states.

1. Start with a straightforward initial partition.
2. Refine iteratively up to the point where *each block only contains bisimilar states*.

Quotienting algorithm (2/11)

Partitions and blocks

Definition: partition

A **partition** of S is a set $\Pi = \{B_1, \dots, B_k\}$ such that

- $\forall\, i,\ B_i \neq \emptyset$,
- $\forall\, i, j,\ i \neq j,\ B_i \cap B_j = \emptyset$,
- $S = \bigcup_{1 \leq i \leq k} B_i$.

Definition: block and superblock

$B_i \in \Pi$ is called a **block**. A **superblock** of Π is a set $C \subseteq S$ such that $C = B_{i_1} \cup \ldots \cup B_{i_l}$ for some $B_{i_1}, \ldots, B_{i_l} \in \Pi$.

A partition Π is finer than Π' if $\forall\, B \in \Pi,\ \exists\, B' \in \Pi',\ B \subseteq B'$.

$\hookrightarrow$ Each block of Π' (coarser) is the disjoint union of blocks in Π.

$\triangleright$ *Strictly* finer if $\Pi \neq \Pi'$.

Quotienting algorithm (3/11)

Partitions and equivalences

- $\mathcal{R}$ is an equivalence on $S \implies S/\mathcal{R}$ is a partition of S.
- $\Pi = \{B_1, \ldots, B_k\}$ is a partition of $S \implies \mathcal{R}_\Pi$ is an equivalence relation

$$\begin{aligned}\mathcal{R}_\Pi &= \{(s, s') \mid \exists\, B_i \in \Pi,\ s \in B_i \wedge s' \in B_i\} \\ &= \{(s, s') \mid [s]_\Pi = [s']_\Pi\}.\end{aligned}$$

- $S/\mathcal{R}_\Pi = \Pi$.

Quotienting algorithm (4/11)

Partition-refinement: key steps

Goal: iteratively compute a partition of S.

1. Initial partition: $\Pi_0 = \Pi_{AP} = S/\mathcal{R}_{AP}$ with

$$\mathcal{R}_{AP} = \{(s, s') \in S \times S \mid L(s) = L(s')\}.$$

   - Group states with identical labels $\Longrightarrow \mathcal{R}_{AP} \supseteq \sim$.

2. Repeat $\Pi_{i+1} = \mathit{Refine}(\Pi_i)$ until stabilization.
   - Loop invariant: Π_i is coarser than $S/\sim$ and finer than $\{S\}$.

3. Return Π_i.
   - Termination: $S \times S \supseteq \mathcal{R}_{\Pi_0} \supsetneq \mathcal{R}_{\Pi_1} \supsetneq \mathcal{R}_{\Pi_2} \supsetneq \ldots \supsetneq \mathcal{R}_{\Pi_i} = \sim$.

Quotienting algorithm (5/11)

Coarsest partition

Theorem

$S/\sim$ is the coarsest partition Π of S such that:

(i) Π is finer than $\Pi_0 = \Pi_{AP}$,

(ii) $\forall\, B, B' \in \Pi,\ B \cap \mathit{Pre}(B') = \emptyset \ \vee\ B \subseteq \mathit{Pre}(B')$.

Moreover, if Π satisfies (ii), then it is also the case that $B \cap \mathit{Pre}(C) = \emptyset \ \vee\ B \subseteq \mathit{Pre}(C)$ for all blocks $B \in \Pi$ and all superblocks C of Π.

Intuitively, (ii) says that if one state in B may lead to B', then all of them must also allow it (otherwise they would not be bisimilar).

$\Longrightarrow$ **The partition-refinement algorithm will lead to the coarsest partition satisfying (i) and (ii), hence to $S/\sim$.**

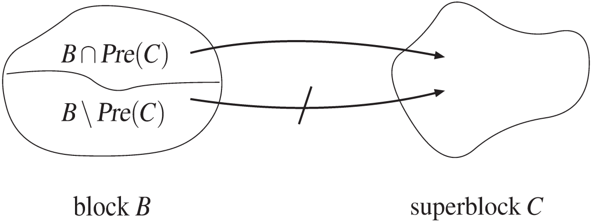
Quotienting algorithm (6/11)

Refinement operator

Definition: refinement operator

$\mathit{Refine}(\Pi, C) = \bigcup_{B \in \Pi} \mathit{Refine}(B, C)$ for C a superblock of Π.

$\mathit{Refine}(B, C) = \{B \cap \mathit{Pre}(C), B \setminus \mathit{Pre}(C)\} \setminus \{\emptyset\}$.



Refinement operator [BK08].

Quotienting algorithm (7/11)

Refinement operator: properties

Correctness

For Π finer than Π_{AP} and coarser than $S/\sim$, we have that:

(a) $\mathit{Refine}(\Pi, C)$ is finer than Π,

(b) $\mathit{Refine}(\Pi, C)$ is coarser than $S/\sim$.

Termination criterion

For Π finer than Π_{AP} and coarser than $S/\sim$, we have that:

Π is strictly coarser than $S/\sim$

$\Updownarrow$

$\exists$ a **splitter** for Π.

$\Longrightarrow$ **When no more splitters, we are done:** $\Pi_i = S/\sim$.

Quotienting algorithm (8/11)

Splitters

Definitions: splitter, stability

Let Π be a partition of S and C a superblock of Π.

- C is a *splitter* of Π if $\exists\, B \in \Pi$ such that

$$B \cap Pre(C) \neq \emptyset \;\wedge\; B \setminus Pre(C) \neq \emptyset.$$

- $B \in \Pi$ is *stable* w.r.t. C if

$$B \cap Pre(C) = \emptyset \;\vee\; B \setminus Pre(C) = \emptyset.$$

- Π is stable w.r.t. C if all $B \in \Pi$ are stable w.r.t. C.

Quotienting algorithm (9/11)

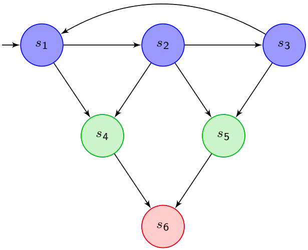
Algorithm (sketch)

Input: TS $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$
Output: bisimulation quotient state space $S/\sim$
    $\Pi := \Pi_{AP}$
    while $\exists$ a splitter for Π **do**
        choose a splitter C for Π
        $\Pi := \mathit{Refine}(\Pi, C)$ $\{\mathit{Refine}(\Pi, C)$ is strictly finer than $\Pi\}$
    return Π

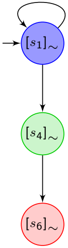
$\Longrightarrow$ **Blackboard illustration on previous example.**

Quotienting algorithm (10/11)

Illustration (summary)



TS $\mathcal{T}$ (all labels $= \emptyset$)



Bisimulation quotient $\mathcal{T}/\sim$

- $\Pi_0 \coloneqq \Pi_{AP} = \{S\}$
- $C = S$, $\Pi \coloneqq \mathit{Refine}(\Pi, C) = \big\{\{s_1, s_2, s_3, s_4, s_5\}, \{s_6\}\big\}$
- $C = \{s_1, s_2, s_3, s_4, s_5\}$, $\Pi \coloneqq \big\{\{s_1, s_2, s_3\}, \{s_4, s_5\}, \{s_6\}\big\}$
- **No more splitters** $\Longrightarrow \Pi = S/\sim$

Quotienting algorithm (11/11)

How should we choose splitters?

What is a good splitter candidate for Π_{i+1}?

1. *Simple strategy*: use any block of Π_i as candidate.
   - $\hookrightarrow$ Complexity of whole algorithm: $\mathcal{O}(|S| \cdot (|AP| + M))$, with M the number of edges.
2. *Advanced strategy*: use only "smaller" blocks of Π_i as candidates and apply "simultaneous" refinement.
   - $\hookrightarrow$ Complexity of whole algorithm: $\mathcal{O}(|S| \cdot |AP| + M \cdot \log |S|)$, with M the number of edges.

$\Longrightarrow$ **See book for more on the advanced strategy.**

Equivalence checking through quotienting (1/2)

Idea

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two TSs. The partition-refinement algorithm can be used to check if $\mathcal{T}_1 \sim \mathcal{T}_2$.

Procedure:

1. Compute the composite TS $\mathcal{T} = \mathcal{T}_1 \oplus \mathcal{T}_2$ defined as

$$\mathcal{T} := (S_1 \uplus S_2, Act_1 \cup Act_2, \longrightarrow_1 \cup \longrightarrow_2, I_1 \cup I_2, AP, L)$$

with $L(s) = L_i(s)$ if $s \in S_i$.

2. Compute $S/\sim$, the bisimulation quotient space of $\mathcal{T}$.
3. Check if, for all bisimulation equivalence class C of $\mathcal{T}$,

$$C \cap I_1 = \emptyset \iff C \cap I_2 = \emptyset.$$

4. **The answer is Yes if and only if $\mathcal{T}_1 \sim \mathcal{T}_2$.**

Equivalence checking through quotienting (2/2)

Complexity

Total complexity:

$$\mathcal{O}\big((|S_1| + |S_2|) \cdot |AP| + (M_1 + M_2) \cdot \log(|S_1| + |S_2|)\big)$$

where M_i is the number of edges of $\mathcal{T}_i$.

$\Longrightarrow$ Polynomial-time whereas trace equivalence is PSPACE-complete.

$\Longrightarrow$ **Much more efficient!**

But recall that:

$$\begin{array}{c}\text{bisimulation}\\ \Downarrow\ \not\Uparrow\\ \text{trace equivalence}\end{array}$$

$\Longrightarrow$ **Sound but incomplete way to check trace equivalence.**

Idea

Bisimulation $s_1 \sim s_2$.

- Equivalence relation.
- Identical stepwise behavior.

Simulation $s_1 \preceq s_2$.

- Preorder (i.e., reflexive, transitive).
- s_2 **simulates** s_1:
  - ▷ s_2 can mimic all stepwise behavior of s_1,
  - ▷ the reverse ($s_2 \preceq s_1$) is not guaranteed.
  - ↪ s_2 may perform transitions that s_1 cannot match.

Simulation $\Longrightarrow$ implementation relation, e.g., $\mathcal{T} \preceq \mathcal{T}_f$, with $\mathcal{T}_f$ an abstraction of $\mathcal{T}$, i.e., $\mathcal{T}$ correctly implements $\mathcal{T}_f$.

Definition

Definition: simulation preorder

Let $\mathcal{T}_i = (S_i, Act_i, \longrightarrow_i, I_i, AP, L_i)$, $i = 1, 2$, be TSs over AP.
A **simulation** for $(\mathcal{T}_1, \mathcal{T}_2)$ is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ s.t.

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

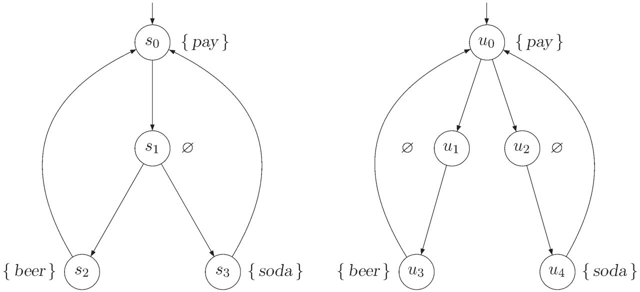
(B) for all $(s_1, s_2) \in \mathcal{R}$ it holds:

- (1) $L_1(s_1) = L_2(s_2)$
- (2) $s_1' \in Post(s_1) \Longrightarrow \big(\exists\, s_2' \in Post(s_2)\ \wedge\ (s_1', s_2') \in \mathcal{R}\big)$

$\mathcal{T}_1$ is simulated by $\mathcal{T}_2$, or equivalently $\mathcal{T}_2$ *simulates* $\mathcal{T}_1$, denoted $\mathcal{T}_1 \preceq \mathcal{T}_2$, if there exists a simulation $\mathcal{R}$ for $(\mathcal{T}_1, \mathcal{T}_2)$.

Observe that bisimulations are also simulations but not the opposite.

Example

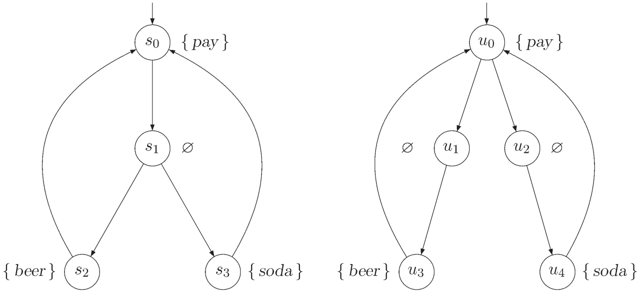


Beverage vending machines [BK08].

Recall that those machines, here called $\mathcal{T}$ and $\mathcal{T}'$, were shown to be **non-bisimilar** before for $AP = \{pay, beer, soda\}$.

What about simulation?

Example



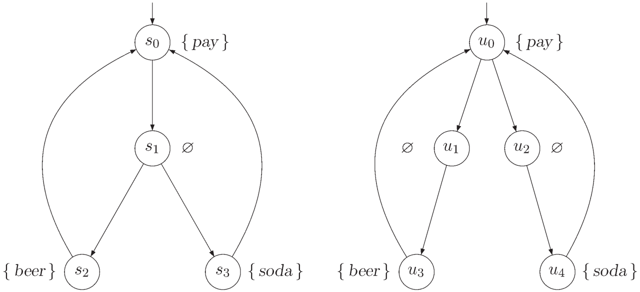
Beverage vending machines [BK08].

The left one simulates the other: $\mathcal{T}' \preceq \mathcal{T}$.

$$\mathcal{R} = \{(u_0, s_0), (u_1, s_1), (u_2, s_1), (u_3, s_2), (u_4, s_3)\}$$

$\Longrightarrow$ **Blackboard proof.**

Example



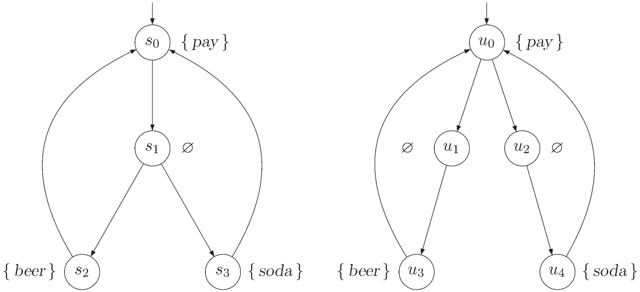
Beverage vending machines [BK08].

The right one does not simulate the other: $\mathcal{T} \not\preceq \mathcal{T}'$.

$\hookrightarrow$ State s_1 cannot be mimicked! Candidates are u_1 and u_2 but they do not satisfy condition (B.2).

- $u_1 \not\rightarrow$ *soda* and $u_2 \not\rightarrow$ *beer*.
- $\mathcal{T} \not\preceq \mathcal{T}'$ for $AP = \{pay, beer, soda\}$.

Example

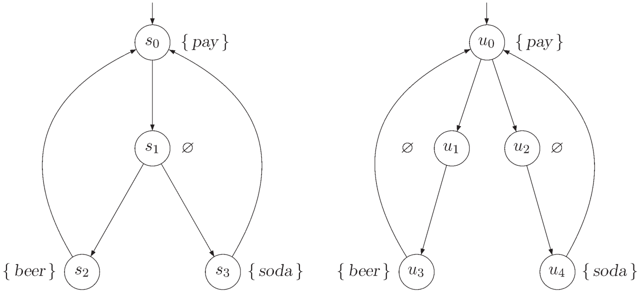


Beverage vending machines [BK08].

What if we take a more abstract labeling $AP = \{pay, drink\}$?

- ▷ $L(s_0) = L(t_0) = \{pay\}$, $L(s_1) = L(u_1) = L(u_2) = \emptyset$, all others labels $= \{drink\}$.

Example



Beverage vending machines [BK08].

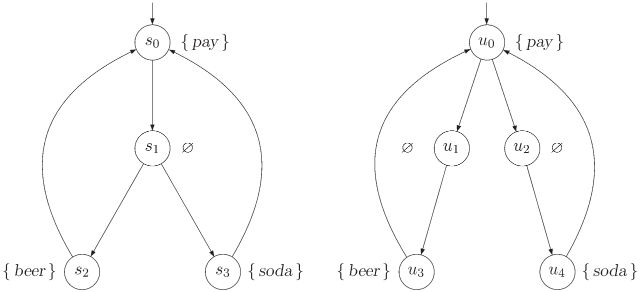
Then, $\mathcal{T}' \preceq \mathcal{T}$ and $\mathcal{T} \preceq \mathcal{T}'$ using

$$\mathcal{R} = \{(u_0, s_0), (u_1, s_1), (u_2, s_1), (u_3, s_2), (u_4, s_3)\}$$

and $\mathcal{R}' = \{(s_0, u_0), (s_1, u_1), (s_2, u_3), (s_3, u_3)\}$

$\Longrightarrow$ **Blackboard proof.**

Example



Beverage vending machines [BK08].

Then, $\mathcal{T}' \preceq \mathcal{T}$ and $\mathcal{T} \preceq \mathcal{T}'$ using

$$\mathcal{R} = \{(u_0, s_0), (u_1, s_1), (u_2, s_1), (u_3, s_2), (u_4, s_3)\}$$

and $\mathcal{R}' = \{(s_0, u_0), (s_1, u_1), (s_2, u_3), (s_3, u_3)\}$

⚠ **Error in book: $\mathcal{R}^{-1}$ does not work for $\mathcal{T} \preceq \mathcal{T}'$ $\Longrightarrow$ exercise.**

Properties

Simulation is a preorder

For a fixed set AP of propositions, the simulation relation $\preceq$ is reflexive and transitive.

- Reflexivity: $\mathcal{T} \preceq \mathcal{T}$.
- Transitivity: $\mathcal{T} \preceq \mathcal{T}' \wedge \mathcal{T}' \preceq \mathcal{T}'' \Longrightarrow \mathcal{T} \preceq \mathcal{T}''$.

$\Longrightarrow$ **Exercise.**

Abstraction (1/4)

Concept

Let $\mathcal{T}$ be a TS.

- If $\mathcal{T}'$ is obtained from $\mathcal{T}$ by removing transitions (e.g., resolving non-determinism), then $\mathcal{T}' \preceq \mathcal{T}$.

  $\hookrightarrow$ $\mathcal{T}'$ is a **refinement** of $\mathcal{T}$.

- If $\mathcal{T}'$ is obtained from $\mathcal{T}$ by **abstraction**, then $\mathcal{T} \preceq \mathcal{T}'$.

Abstraction: idea

Represent a set of concrete states (with identical labels) using a unique abstract state, through an abstraction function $f \colon S \to \widehat{S}$.

Abstraction function

$f \colon S \to \widehat{S}$ is an abstraction function if

$$f(s) = f(s') \Longrightarrow L(s) = L(s').$$

Abstraction (2/4)

Usefulness

- From concrete states S to abstract states $\widehat{S}$ s.t. $|\widehat{S}| \lll |S|$.
  - $\hookrightarrow$ Goal: more efficient model checking.
- Useful for data abstraction, predicate abstraction, localization reduction.

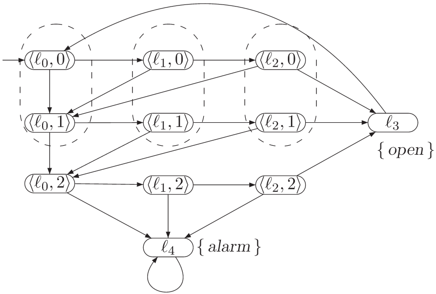
$\Longrightarrow$ **See book for formal discussion.**

Here, example of an automatic door opener.

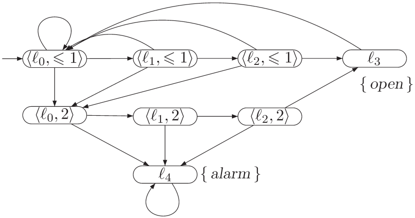
- ▷ Three-digit code, two errors allowed before alarm.

Abstraction (3/4)

Example: automatic door opener (1/2)



Automatic door opener [BK08].



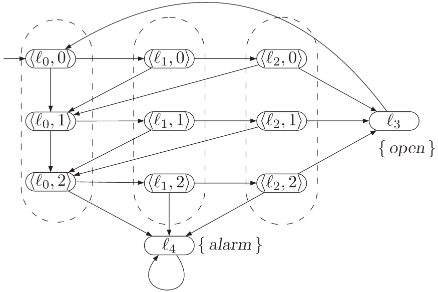
Abstract TS [BK08].

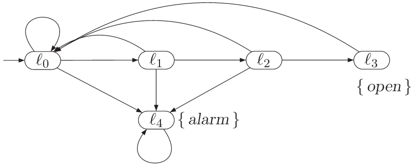
First abstraction: group by number of errors $\{\leq 1, 2\}$.

By construction, $\mathcal{T} \preceq \mathcal{T}_f$.

Abstraction (4/4)

Example: automatic door opener (2/2)





Abstract TS [BK08].

Automatic door opener [BK08].

Second abstraction: complete abstraction of the number of errors.

$\hookrightarrow$ Coarser abstraction $\Longrightarrow$ smaller TS.

By construction, $\mathcal{T} \preceq \mathcal{T}_f$.

Simulation equivalence

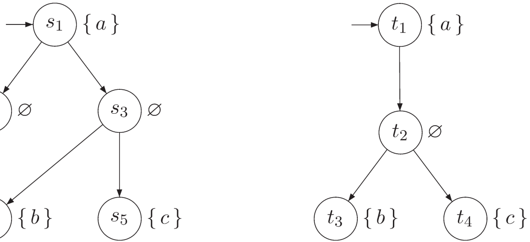
Definition: simulation equivalence

TSs $\mathcal{T}_1$ and $\mathcal{T}_2$ are simulation-equivalent, or *similar*, denoted $\mathcal{T}_1 \simeq \mathcal{T}_2$, if $\mathcal{T}_1 \preceq \mathcal{T}_2$ and $\mathcal{T}_2 \preceq \mathcal{T}_1$.

Simulation is **coarser** than bisimulation:

$$\begin{array}{c} \mathcal{T}_1 \simeq \mathcal{T}_2 \\ \not\Downarrow \Uparrow \\ \mathcal{T}_1 \sim \mathcal{T}_2 \end{array}$$

Example



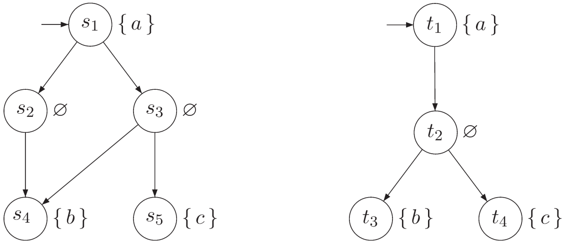
Similar but not bisimilar TSs [BK08].

$\mathcal{T}_1 \simeq \mathcal{T}_2$

- $\mathcal{T}_1 \preceq \mathcal{T}_2$: $\mathcal{R}_1 = \{(s_1, t_1), (s_2, t_2), (s_3, t_2), (s_4, t_3), (s_5, t_4)\}$.
- $\mathcal{T}_2 \preceq \mathcal{T}_1$: $\mathcal{R}_2 = \{(t_1, s_1), (t_2, s_3), (t_3, s_4), (t_4, s_5)\}$.

$\Longrightarrow$ **Blackboard proof.**

Example



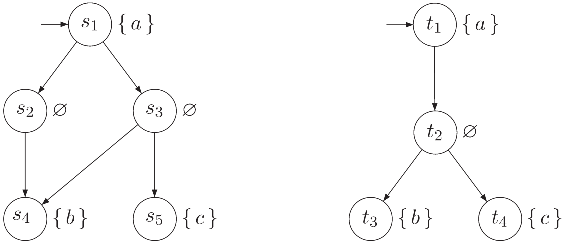
Similar but not bisimilar TSs [BK08].

$\mathcal{T}_1 \simeq \mathcal{T}_2$ but $\mathcal{T}_1 \not\sim \mathcal{T}_2$

- Only candidate to mimic s_2 is t_2 but $t_2 \rightarrow t_4$ cannot be mimicked by s_2.

$\Longrightarrow$ **Blackboard proof.**

Example



Similar but not bisimilar TSs [BK08].

$\mathcal{T}_1 \simeq \mathcal{T}_2$ but $\mathcal{T}_1 \not\sim \mathcal{T}_2$. The difference is that:

- For $\simeq$, we can use two $\neq$ relations $\mathcal{R}_1$ and $\mathcal{R}_2$.
- For $\sim$, we need to use **the same relation** in both directions!

Quotienting (1/3)

Idea

Idea

1. As for bisimulation, see simulation as a relation between states of a *single* TS.
2. Quotient the TS by this relation.
   - ▷ Obtain a smaller TS that preserves properties.
3. Model check the smaller TS.
   - ▷ More efficient! (quotienting is "cheap" in comparison to model checking)

Since simulation is coarser than bisimulation, the simulation quotient will be **a better abstraction**, i.e., $|S/\simeq| \leq |S/\sim|$.

Still, simulation only preserves **a smaller fragment of CTL**, while bisimulation preserves the whole logic.

$\Longrightarrow$ If sufficient, use the simulation quotient.

Quotienting (2/3)

Simulation on states

Definition: simulation preorder as a relation on states

Let $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$ be a TS. A **simulation** for $\mathcal{T}$ is a binary relation $\mathcal{R}$ on $S \times S$ s.t. for all $(s_1, s_2) \in \mathcal{R}$:

(1) $L(s_1) = L(s_2)$

(2) $s'_1 \in Post(s_1) \Longrightarrow (\exists\, s'_2 \in Post(s_2) \;\wedge\; (s'_1, s'_2) \in \mathcal{R})$.

States s_1 is simulated by s_2, or s_2 *simulates* s_1, denoted $s_1 \preceq_{\mathcal{T}} s_2$, if there exists a simulation $\mathcal{R}$ for $\mathcal{T}$ with $(s_1, s_2) \in \mathcal{R}$. States s_1 and s_2 are *similar*, denoted $s_1 \simeq_{\mathcal{T}} s_2$ if $s_1 \preceq_{\mathcal{T}} s_2$ and $s_2 \preceq_{\mathcal{T}} s_1$.

Remark: $\preceq_{\mathcal{T}}$ is the *coarsest simulation* for $\mathcal{T}$.

For simplicity, we write $\preceq$ and $\simeq$ for $\preceq_{\mathcal{T}}$ and $\simeq_{\mathcal{T}}$ in the following.

Quotienting (3/3)

Simulation quotient

Quotient

Let $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$ be a TS. The **simulation quotient** of $\mathcal{T}$ is defined by

$$\mathcal{T}/\!\simeq\; = (S/\!\simeq, \{\tau\}, \longrightarrow', I', AP, L')$$

where:

- $I' = \{[s]_{\simeq} \mid s \in I\}$,
- $s \xrightarrow{\alpha} s' \implies [s]_{\simeq} \xrightarrow{\tau}' [s']_{\simeq}$,
- $L'([s]_{\simeq}) = L(s)$.

It is easily shown that $\mathcal{T} \simeq \mathcal{T}/\!\simeq$.

Algorithm for simulation preorder (1/4)

Goal

Goal

Given a TS $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$, compute the simulation preorder $\preceq_{\mathcal{T}}$ (the coarsest simulation).

- Can be used to compute $\mathcal{T}/\simeq$ (by looking at states s_1, s_2 such that $s_1 \preceq s_2$ and $s_2 \preceq s_1$).
- Can be used to check whether $\mathcal{T}_1 \simeq \mathcal{T}_2$ by computing $\mathcal{T}_1 \oplus \mathcal{T}_2/\simeq$ as for bisimulation.

Algorithm for simulation preorder (2/4)

Basic idea

Input: TS $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$
Output: simulation preorder $\preceq_{\mathcal{T}}$
$\quad\mathcal{R} := \{(s_1, s_2) \mid L(s_1) = L(s_2)\}$
$\quad$**while** $\mathcal{R}$ is not a simulation **do**
$\quad\quad$let $(s_1, s_2) \in \mathcal{R}$ s.t. $s_1 \rightarrow s_1' \wedge \nexists\, s_2'$ s.t. $\big(s_2 \rightarrow s_2' \wedge (s_1', s_2') \in \mathcal{R}\big)$
$\quad\quad\mathcal{R} := \mathcal{R} \setminus \{(s_1, s_2)\}$
$\quad$**return** $\mathcal{R}$

Intuitively, we start with the largest possible approximation (i.e., identical labels) and iteratively remove pairs of states that do not satisfy $s_1 \preceq s_2$ up to obtaining a proper simulation relation.

$\#$ iterations bounded by $|S|^2$:

$$S \times S \supseteq \mathcal{R}_0 \subsetneq \mathcal{R}_1 \supsetneq \ldots \supsetneq \mathcal{R}_n = \preceq_{\mathcal{T}}$$

Algorithm for simulation preorder (3/4)

Complexity

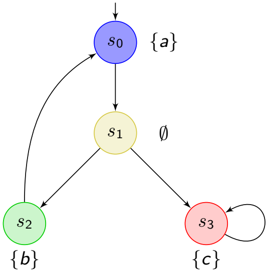
While straightforward implementation leads to $\mathcal{O}(M \cdot |S|^3)$, clever refinements reduce the complexity of the algorithm to $\mathcal{O}(M \cdot |S|)$.

$\Longrightarrow$ **See the book for more details.**

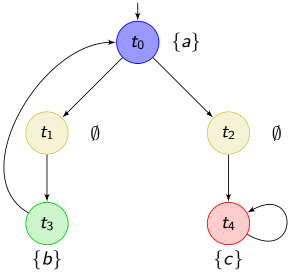
$\Longrightarrow$ **Blackboard illustration for two TSs.**

Algorithm for simulation preorder (4/4)

Illustration (summary)



TS $\mathcal{T}_1$



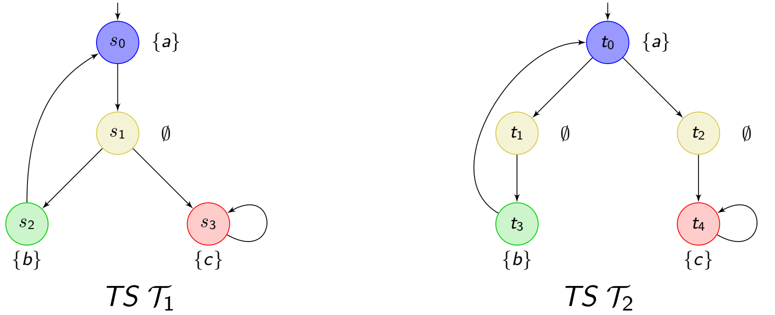
TS $\mathcal{T}_2$

$\mathcal{T}_1 \preceq \mathcal{T}_2$?

- $\mathcal{R}_0 = \{(s_0, t_0), (s_1, t_1), (s_1, t_2), (s_2, t_3), (s_3, t_4)\}$
- $\mathcal{R}_1 = \{(s_0, t_0), (s_1, t_2), (s_2, t_3), (s_3, t_4)\}$
- $\mathcal{R}_2 = \{(s_0, t_0), (s_2, t_3), (s_3, t_4)\}$, $\mathcal{R}_3 = \{(s_2, t_3), (s_3, t_4)\}$

Algorithm for simulation preorder (4/4)

Illustration (summary)



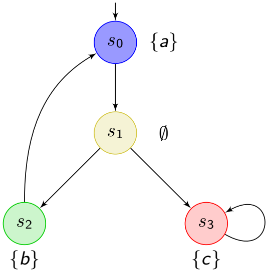
$\mathcal{T}_1 \preceq \mathcal{T}_2$?

- $\mathcal{R}_4 = \{(s_3, t_4)\} = \preceq$

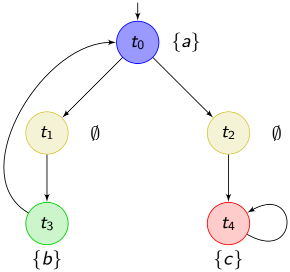
$$(s_0, t_0) \notin \preceq \implies \mathcal{T}_1 \not\preceq \mathcal{T}_2$$

Algorithm for simulation preorder (4/4)

Illustration (summary)



TS $\mathcal{T}_1$



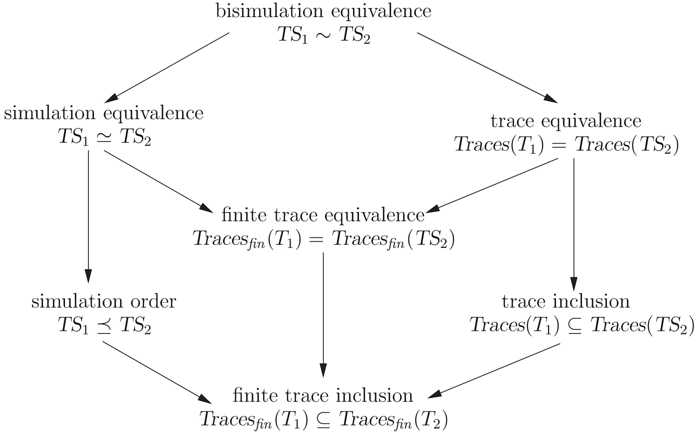
TS $\mathcal{T}_2$

$\mathcal{T}_2 \preceq \mathcal{T}_1$?

- $\mathcal{R}_0 = \{(t_0, s_0), (t_1, s_1), (t_2, s_1), (t_3, s_2), (t_4, s_3)\} = \preceq$

$$(t_0, s_0) \in \preceq \implies \mathcal{T}_2 \preceq \mathcal{T}_1$$

Relations between equivalences: summary



Relation between equivalences and preorders on TSs [BK08]: $\mathcal{R} \rightarrow \mathcal{R}'$ means that $\mathcal{R}$ is strictly finer than $\mathcal{R}'$ (i.e., it is more distinctive).

Other properties of simulation

If $\mathcal{T}_1$ and $\mathcal{T}_2$ do not have terminal states:

- $\mathcal{T}_1 \preceq \mathcal{T}_2 \implies \mathit{Traces}(\mathcal{T}_1) \subseteq \mathit{Traces}(\mathcal{T}_2)$;
- if $\mathcal{T}_2$ satisfies a linear-time property (LTL), then $\mathcal{T}_1$ also;
- if $\mathcal{T}_2$ satisfies a branching-time property expressible in $\forall$CTL or $\exists$CTL (i.e., strict fragments of CTL), then $\mathcal{T}_1$ also.

$\implies$ **See book for more.**

References I

C. Baier and J.-P. Katoen.
Principles of model checking.
MIT Press, 2008.