Master in Artificial Intelligence and Robotics (AIRO)
Electives in AI
Reasoning Agents

Fabio Patrizi

Sapienza University of Rome, Italy
patrizi@diag.uniroma1.it

A.Y. 2021-2022

Transition Systems

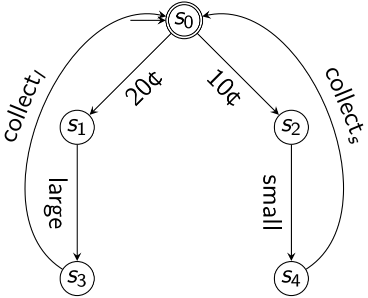
Transition Systems

Transition System

- Mathematical structure similar to graph
- Models states of the world and transitions among them
- Commonly used to reason about actions and world dynamics
- Typically, underlying model of some (compact) formalism
- Many variants

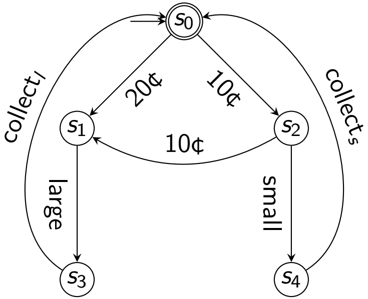
Transition Systems

Example: Vending Machine 1

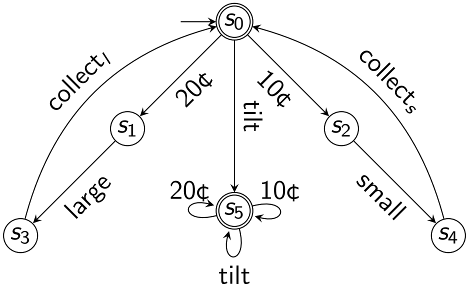


Transition Systems

Example: Vending Machine 2

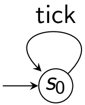


Example: Vending Machine 3



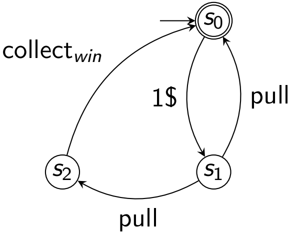
Transition Systems

Example: Non-terminating Process

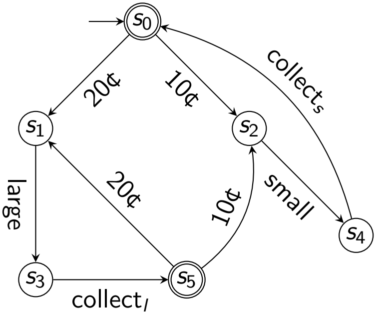


Transition Systems

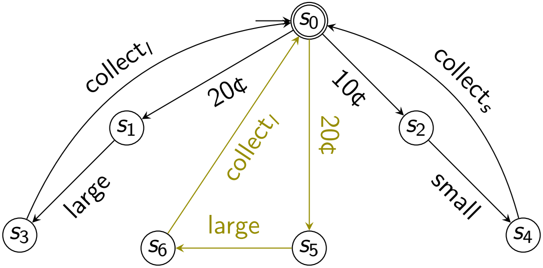
Example: Nondeterministic Domain



Example: Vending Machine 1 (variant α)



Example: Vending Machine 1 (variant β)



Transition Systems

Definition (Transition System)

$\mathcal{T} = (A, S, s_0, \rightarrow, F)$, where:

- A: set of *actions*
- S: set of *states*
- $s_0 \in S$: *initial* state
- $\rightarrow$: $S \times A \times S$: *transition relation* ($(s, a, s') \in \rightarrow$ denoted as $s \xrightarrow{a} s'$)
- $F \subseteq S$: set of *final* states

Variants (all include states and transitions):

- Finite/infinite actions/states
- No/single/many initial states
- Deterministic/nondeterministic actions/transitions
- No final states, **labelled states** (common in this course)

Labelled Transition Systems

Definition (Transition System)

$\mathcal{T} = (P, A, S, s_0, \rightarrow, \lambda)$, where:

- P: finite set of *propositions*
- A: set of *actions*
- S: set of *states*
- $s_0 \in S$: *initial* state
- $\rightarrow$: $S \times A \times S$: transition relation
- $\lambda : S \rightarrow 2^P$: labeling function

Observe:

- TSs with final states are a special case of labelled TSs:
  - $P = \{final\}$
  - s *final* iff $\lambda(s) = \{final\}$

Representation of TSs

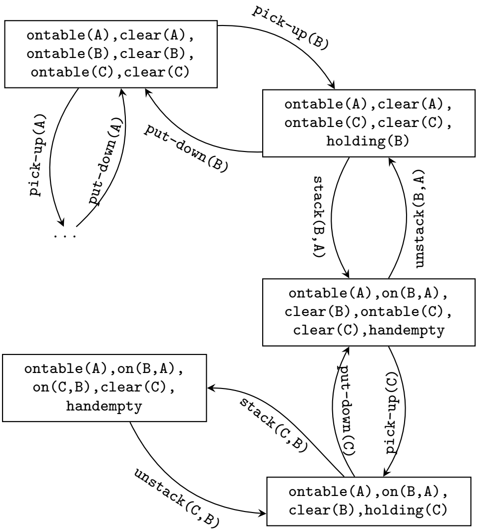
Typically represented compactly

```
(define (domain blocksworld)
  (:requirements :strips :typing)
  (:types block)
  (:predicates (on ?x - block ?y - block)
               (ontable ?x - block)
               (clear ?x - block)
               (handempty)
               (holding ?x - block)
               )

  (:action pick-up
             :parameters (?x - block)
             :precondition (and (clear ?x) (ontable ?x) (handempty))
             :effect
             (and (not (ontable ?x))
                   (not (clear ?x))
                   (not (handempty))
                   (holding ?x)))

  (:action put-down
             :parameters (?x - block)
             :precondition (holding ?x)
             :effect
             (and (not (holding ?x))
                   (clear ?x)
                   (handempty)
                   (ontable ?x)))

  (:action stack
             :parameters (?x - block ?y - block)
             :precondition (and (holding ?x) (clear ?y))
             :effect
             (and (not (holding ?x))
                   (not (clear ?y))
                   (clear ?x)
                   (handempty)
                   (on ?x ?y)))
  (:action unstack
             :parameters (?x - block ?y - block)
             :precondition (and (on ?x ?y) (clear ?x) (handempty))
             :effect
             (and (holding ?x)
                   (clear ?y)
                   (not (clear ?x))
                   (not (handempty))
                   (not (on ?x ?y))))
)
```



The Reachability Relation

- Many reasoning tasks are related to *Reachability*, e.g.:
  - check whether a *goal* state s (i.e., with desired label) is *reachable* from initial state s_0, i.e., there exists a path from s_0 to s
  - classical planning: find path from initial state s_0 to some goal state
  - nondeterministic planning: check whether set of reachable states from s_0 includes goal states only

- Fundamental Problem:
  - For every state $s \in S$, compute set of states *reachable* from s
  - Formalized as computation of *Reachability Relation*

Reachability-like

Definition (Reachability-like Relation)

- Let TS $\mathcal{T} = (P, A, S, s_0, \to, \lambda)$
- $R \subseteq S \times S$ is a *reachability-like* relation (over $\mathcal{T}$) if:
  - $(s, s) \in R$, for all $s \in S$
  - if (for some $s \in S$ and $a \in A$) $s \xrightarrow{a} s'$ and $(s', s'') \in R$ then $(s, s'') \in R$

Observe:

1. for all s' reachable from s, it is the case that $(s, s') \in R$
2. however, for some s, R may contain (s, s') with s' unreachable from s
3. how to exclude such unreachable states?

Reachability

Definition (Reachability relation)

- $R \subseteq S \times S$ is a *reachability* relation (over $\mathcal{T}$) if:
  - $(s, s') \in R$ iff $(s, s') \in R'$ *for all* reachability-like relations R'

Observe:

1. A reachability relation R is also a reachability-like relation
2. A reachability relation R is *the smallest* reachability-like relation

Equivalent inductive definition

Definition (Reachability relation)

- Let TS $\mathcal{T} = (P, A, S, s_0, \rightarrow, \lambda)$
- The reachability relation of a TS $\mathcal{T}$ is the smallest relation $R \subseteq S \times S$ s.t.:
  - $(s, s) \in R$, for all $s \in S$
  - if $s \xrightarrow{a} s'$ and $(s', s'') \in R$ then $(s, s'') \in R$

Computing Reachability

- On finite TSs, reachability relation easily computable

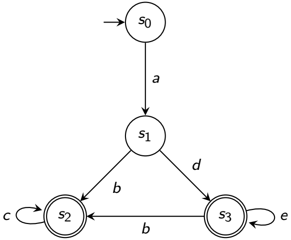
Algorithm ComputeReachability

Input: Transition system $\mathcal{T} = (P, A, S, s_0, \rightarrow, \lambda)$
Output: Reachability relation of $\mathcal{T}$

$R := \emptyset$
$R' := \{(s, s) \mid s \in S\}$
while$(R \neq R')\{$
  $R := R'$
  $R' := R' \cup \{(s, s'') \mid s \xrightarrow{a} s' \text{ and } (s', s'') \in R'\}$
$\}$
return R

- *Least fixpoint* computation by approximates, starting from empty set

Computing Reachability

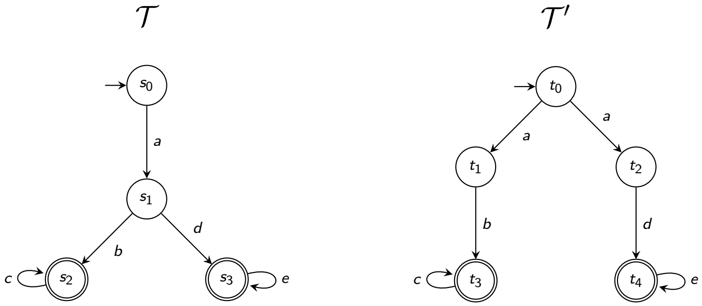


$$
\begin{aligned}
R^0 &= \emptyset \\
R^1 &= \{(s_0, s_0), (s_1, s_1), (s_2, s_2), (s_3, s_3)\} \\
R^2 &= R^1 \cup \{(s_0, s_1), (s_1, s_2), (s_1, s_3), (s_3, s_2)\} \\
R^3 &= R^2 \cup \{(s_0, s_2), (s_0, s_3)\} \\
R^4 &= R^3
\end{aligned}
$$

$$
\begin{gathered}
R = \{(s_0, s_0), (s_0, s_1), (s_0, s_2), (s_0, s_3) \\
(s_1, s_1), (s_1, s_2), (s_1, s_3), \\
(s_2, s_2), (s_3, s_2), (s_3, s_3)\}
\end{gathered}
$$

Transition Systems vs. Automata

- Automata: define *language*, i.e., set of (finite) strings
- Transition Systems: define *behaviors*, i.e., strings but also *choices*



- Same language: $abc^* + ade^*$
- Different choices:
  - $\mathcal{T}$: two choices after a
  - $\mathcal{T}'$: no choice after a
- Thus, different behaviors

Equivalent Transition Systems

- Intuition: two TSs are *equivalent*, or *bisimilar* if from their initial states *exactly* the same behaviors start
  - same strings (considering labelings, if present)
  - same choices
- Formalized through notions of
  - *Bisimulation*
  - *Bisimilarity*

Bisimulation

Definition (Bisimulation relation)

- Let $\mathcal{T} = (P, A, S, s_0, \rightarrow, \lambda)$ and $\mathcal{T}' = (P, A, T, t_0, \rightarrow', \lambda')$ be two (possibly the same) TSs
- $B \subseteq S \times T$ is a *bisimulation* relation (over $\mathcal{T}$) if $(s, t) \in B$ implies:
  - $\lambda(s) = \lambda'(t)$
  - for all actions $a \in A$:
    - if $s \xrightarrow{a} s'$ then, for some t', $t \xrightarrow{a} t'$ and $(s', t') \in B$
    - if $t \xrightarrow{a} t'$ then, for some s', $s \xrightarrow{a} s'$ and $(s', t') \in B$

Observe:

1. if two states (s, t) are in a bisimulation relation B they give raise to same behaviors
2. however, there may exist pairs not in B generating same behaviors
3. how to include such pairs?

Bisimilarity relation

Definition (Bisimilarity relation)

- Let $\mathcal{T} = (P, A, S, s_0, \rightarrow, \lambda)$ and $\mathcal{T}' = (P, A, T, t_0, \rightarrow', \lambda')$ be two (possibly the same) TSs
- $B \subseteq S \times T$ is a *bisimilarity* relation between $\mathcal{T}$ and $\mathcal{T}'$, if:
    - $(s, t) \in B$ iff $(s, t) \in B'$ for some bisimulation relation B'

Observe:

1. A bisimilarity relation B is also a bisimulation relation
2. A bisimilarity relation B is *the largest* bisimulation relation between (the states of) $\mathcal{T}$ and $\mathcal{T}'$

Bisimilarity relation

Equivalent co-inductive definition

Definition (Bisimilarity relation)

- Let $\mathcal{T} = (P, A, S, s_0, \rightarrow, \lambda)$ and $\mathcal{T}' = (P, A, T, t_0, \rightarrow', \lambda')$ be two (possibly the same) TSs
- The *bisimilarity* relation between $\mathcal{T}$ and $\mathcal{T}'$ is the largest relation $B \subseteq S \times T$ s.t. $(s, t) \in B$ implies:
  - $\lambda(s) = \lambda'(t)$
  - for all actions $a \in A$:
    - if $s \xrightarrow{a} s'$ then, for some t', $t \xrightarrow{a} t'$ and $(s', t') \in B$
    - if $t \xrightarrow{a} t'$ then, for some s', $s \xrightarrow{a} s'$ and $(s', t') \in B$

Bisimilarity relation

Definition (Bisimilar states)

Two states s, t of two transition systems $\mathcal{T}$ and $\mathcal{T}'$ are said to be *bisimilar*, or *equivalent*, if $(s, t) \in B$, for B the bisimilarity relation between $\mathcal{T}$ and $\mathcal{T}'$.

Definition (Bisimilar transition systems)

Two transition systems $\mathcal{T}$ and $\mathcal{T}'$ are said to be *bisimilar*, or *equivalent*, if their respective initial states s_0 and t_0 are s.t. $(s_0, t_0) \in B$, for B the bisimilarity relation between $\mathcal{T}$ and $\mathcal{T}'$.

Computing Bisimilarity

- On finite TSs, bisimilarity relation easily computable

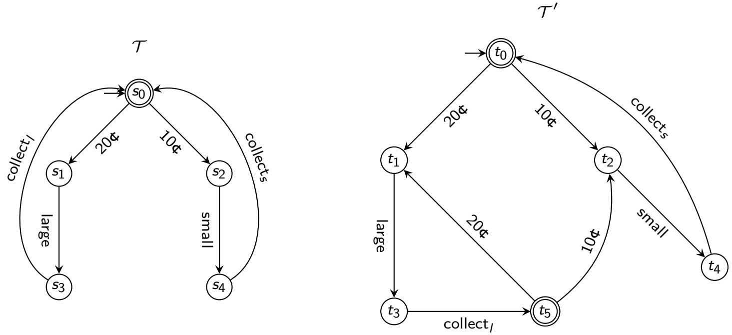
Algorithm ComputeBisimilarity

Input: TSs $\mathcal{T} = (P, A, S, s_0, \rightarrow, \lambda)$, $\mathcal{T}' = (P, A, T, t_0, \rightarrow', \lambda')$
Output: Bisimilarity relation between $\mathcal{T}$ and $\mathcal{T}'$

$B := S \times T$
$B' := B \setminus \{(s,t) \mid \lambda(s) \neq \lambda'(t)\}$
while$(B \neq B')\{$
  $B := B'$
  $B' := B' \setminus$
    $(\{(s,t) \mid \exists a, s'.s \xrightarrow{a} s' \text{ and } \nexists t'.t' \xrightarrow{a}' t \text{ and } (s',t') \in B'\} \cup$
    $\{(s,t) \mid \exists a, t'.t \xrightarrow{a}' t' \text{ and } \nexists s'.s' \xrightarrow{a} s' \text{ and } (s',t') \in B'\})$
$\}$
return B

- *Greatest fixpoint* computation by approximates, starting from total relation $S \times T$

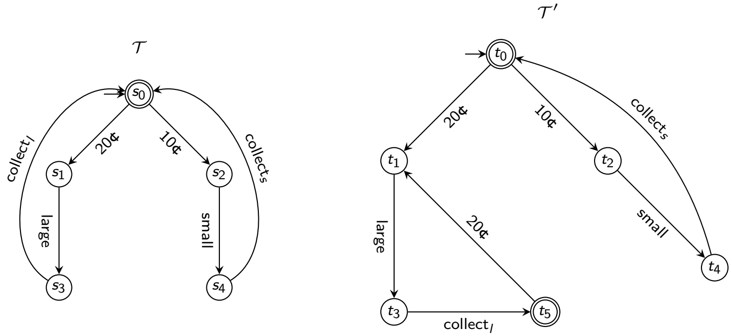
Computing Bisimilarity



1. $B^0 = \{(s_0, t_0), \ldots, (s_0, t_5), (s_1, t_0), \ldots, (s_1, t_5), (s_2, t_0), \ldots, (s_2, t_5), (s_3, t_0), \ldots, (s_3, t_5), (s_4, t_0), \ldots, (s_4, t_5)\}$
2. $B^1 = \{(s_0, t_0), (s_0, t_5), (s_1, t_1), \ldots, (s_1, t_4), (s_2, t_1), \ldots, (s_2, t_4), (s_3, t_1), \ldots, (s_3, t_4), (s_4, t_1), \ldots, (s_4, t_4)\}$
3. $B^2 = \{(s_0, t_0), (s_0, t_5), (s_1, t_1), (s_2, t_2), (s_3, t_3), (s_4, t_4)\}$
4. $B^3 = B^2$, then $B = B^3$

- Are $\mathcal{T}$ and $\mathcal{T}'$ bisimilar? Yes: $(s_0, t_0) \in B$

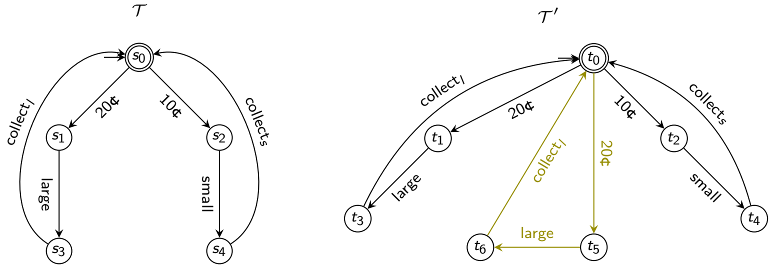
Computing Bisimilarity



1. $B^0 = \{(s_0, t_0), \ldots, (s_0, t_5), (s_1, t_0), \ldots, (s_1, t_5), (s_2, t_0), \ldots, (s_2, t_5), (s_3, t_0), \ldots, (s_3, t_5), (s_4, t_0), \ldots, (s_4, t_5)\}$
2. $B^1 = \{(s_0, t_0), (s_0, t_5), (s_1, t_1), \ldots, (s_1, t_4), (s_2, t_1), \ldots, (s_2, t_4), (s_3, t_1), \ldots, (s_3, t_4), (s_4, t_1), \ldots, (s_4, t_4)\}$
3. $B^2 = \{(s_0, t_0), (s_1, t_1), (s_2, t_2), (s_3, t_3), (s_4, t_4)\}$
4. $B^3 = \{(s_0, t_0), (s_1, t_1), (s_2, t_2), (s_4, t_4)\}$
5. $B^4 = \{(s_0, t_0), (s_2, t_2), (s_4, t_4)\}$
6. $B^5 = \{(s_2, t_2), (s_4, t_4)\}$
7. $B^6 = \{(s_4, t_4)\}$
8. $B^7 = \{\}$

- Are $\mathcal{T}$ and $\mathcal{T}'$ bisimilar? No: $(s_0, t_0) \notin B$

Computing Bisimilarity



- Are $\mathcal{T}$ and $\mathcal{T}'$ bisimilar?
- What are the bisimilar states within same TS?

Summary

- Transition Systems are behavioral models of systems
- Fundamental reasoning problems include computing reachability and bisimilarity
- Fixpoint computations required

References I