Verification – Lecture 19
Symbolic Model Checking (2)

Bernd Finkbeiner – Sven Schewe
Rayna Dimitrova – Lars Kuhtz – Anne Proetzsch

Wintersemester 2007/2008

REVIEW

Ordered Binary Decision Diagram

- Binary decision diagram (OBDD) is a directed graph over $\langle X, < \rangle$ with:
  - each leaf v is labeled with a boolean value *val*$(v) \in \{ 0, 1 \}$
  - non-leaf v is labeled by a boolean variable *Var*$(v) \in X$
  - such that for each non-leaf v and vertex w:

$$w \in \{ \textit{left}(v), \textit{right}(v) \} \;\Rightarrow\; (\textit{Var}(v) < \textit{Var}(w) \;\vee\; w \text{ is a leaf})$$

⇒ An OBDD is acyclic

  - f_{B} for OBDD B is obtained as for BDTs

Shannon expansion

- Each boolean function $f : \mathbb{B}^n \longrightarrow \mathbb{B}$ can be written as:

$$f(x_1, \ldots, x_n) = (x_i \ \wedge \ f[x_i := 1]) \ \vee \ (\neg\, x_i \ \wedge \ f[x_i := 0])$$

  - where $f[x_i := 1]$ stands for $f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$
  - and $f[x_i := 0]$ is a shorthand for $f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$

- The boolean function $f_{\mathsf{B}}(v)$ represented by vertex v in BDT B is:
  - for v a leaf: $f_{\mathsf{B}}(v) = \mathit{val}(v)$
  - otherwise:

$$f_{\mathsf{B}}(v) = (\mathit{Var}(v) \ \wedge \ f_{\mathsf{B}}(\mathit{right}(v))) \ \vee \ (\neg\, \mathit{Var}(v) \ \wedge \ f_{\mathsf{B}}(\mathit{left}(v)))$$

- $f_{\mathsf{B}} = f_{\mathsf{B}}(v)$ where v is the root of B

Reduced OBDDs

OBDD B over $\langle X, < \rangle$ is called *reduced* iff:

1. for each leaf v, w: $(\mathit{val}(v) = \mathit{val}(w)) \ \Rightarrow \ v = w$

   $\Rightarrow$ identical terminal vertices are forbidden

2. for each non-leaf v: $\mathit{left}(v) \neq \mathit{right}(v)$

   $\Rightarrow$ non-leafs may not have identical children

3. for each non-leaf v, w:

$$(\mathit{Var}(v) = \mathit{Var}(w) \ \wedge \ \mathit{right}(v) \cong \mathit{right}(w) \ \wedge \ \mathit{left}(v) \cong \mathit{left}(w)) \ \Rightarrow \ v = w$$

   $\Rightarrow$ vertices may not have isomorphic sub-dags

Dynamic generation of ROBDDs

Main idea:

- Construct directly an ROBDD from a boolean expression
- Create vertices in depth-first search order
- On-the-fly reduction by applying hashing
  - on encountering a new vertex v, check whether:
  - an equivalent vertex w has been created (same label and children)
  - *left*(v) = *right*(v), i.e., vertex v is a “don’t care” vertex

ROBDDs are canonical

[Fortune, Hopcroft & Schmidt, 1978]

For ROBDDs B and B′ over $\langle X, < \rangle$ we have:
$(f_B = f_{B'})$ implies B and B′ are isomorphic

$\Rightarrow$ for a fixed variable ordering, any boolean function can be uniquely represented by an ROBDD (up to isomorphism)

The importance of canonicity

- Absence of redundant vertices
  - if f_{B} does not depend on x_i, ROBDD B does not contain an x_i vertex
- Test for equivalence: $f(x_1, \ldots, x_n) \equiv g(x_1, \ldots, x_n)$?
  - generate ROBDDs B_f and B_g, and check isomorphism
- Test for validity: $f(x_1, \ldots, x_n) = 1$?
  - generate ROBDD B_f and check whether it only consists of a 1-leaf
- Test for implication: $f(x_1, \ldots, x_n) \rightarrow g(x_1, \ldots, x_n)$?
  - generate ROBDD $\mathrm{B}_f \wedge \neg \mathrm{B}_g$ and check if it just consist of a 0-leaf
- Test for satisfiability
  - f is satisfiable if and only if B_f is not just the 0-leaf

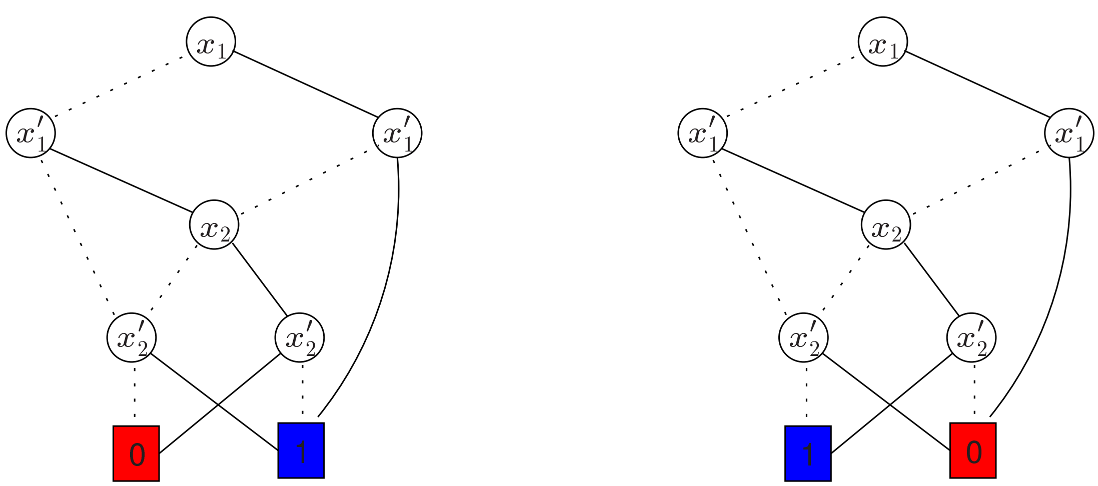
Variable ordering

- The size of the ROBDD depends on the variable ordering
- For some functions, very compact ROBDDs may be obtained
  - e.g., the even parity function
- Some boolean functions have linear and exponential ROBDDs
  - e.g., the addition function, or the stable function
- Some boolean functions only have polynomial ROBDDs
  - this holds, e.g., for symmetric functions (see next)
  - examples $f(\ldots) = x_1 \oplus \ldots \oplus x_n$, or $f(\ldots) = 1$ iff $\geqslant k$ variables x_i are true
- Some boolean functions only have exponential ROBDDs
  - this holds, e.g., for the multiplication function, cf. (Bryant, 1986)

Operations on ROBDDs

Algorithm	Inputs	Output ROBDD
REDUCE	B (not reduced)	B′ (reduced) with $f_{\mathsf{B}} = f_{\mathsf{B}'}$
NOT	B_f	$\mathsf{B}_{\neg f}$
APPLY	B_f, B_g, binary logical operator *op*	$\mathsf{B}_{f\ op\ g}$
RESTRICT	B_f, variable x, boolean value b	$\mathsf{B}_{f[x:=b]}$
RENAME	B_f, variables x and y	$\mathsf{B}_{f[x:=y]}$
EXISTS	B_f, variable x	$\mathsf{B}_{\exists x.\ f}$

Negation



negation amounts to interchange the 0- and 1-leaf

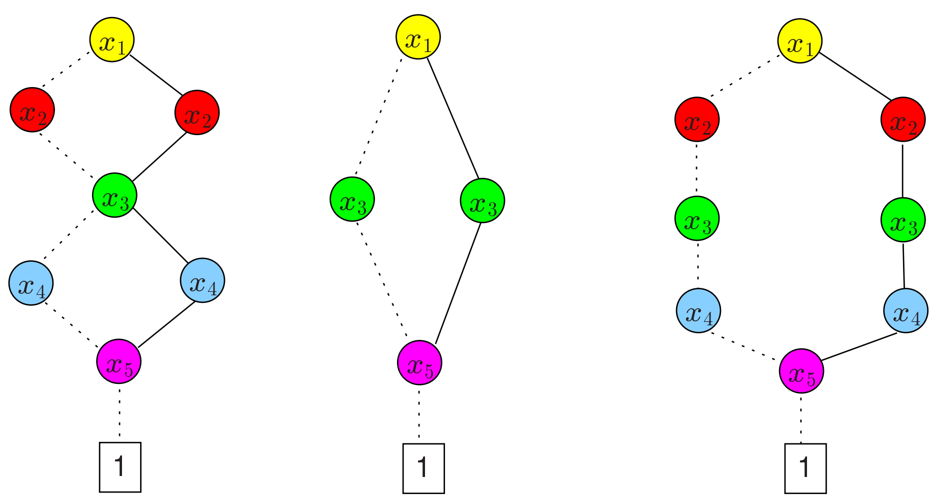
APPLY

- Shannon expansion for binary operations:

$$f \text{ } op \text{ } g \quad = \quad (x_1 \wedge (f[x_1 := 1] \text{ } op \text{ } g[x_1 := 1])) \vee (\neg x_1 \wedge (f[x_1 := 0] \text{ } op \text{ } g[x_1 := 0]))$$

- A top-down evaluation scheme using the Shannon's expansion:
  - let v be the variable highest in the ordering occurring in B_f orB_g
  - split the problem into subproblems for $v := 0$ and $v := 1$, and solve recursively
  - at the leaves, apply the boolean operator *op* directly
  - reduce afterwards to turn the resulting OBDD into an ROBDD
- Efficiency gain is obtained by *dynamic programming*
  - the time complexity of constructing the ROBDD of $B_{f \text{ } op \text{ } g}$ is in $\mathcal{O}(|B_f| \cdot |B_g|)$

Conjunction



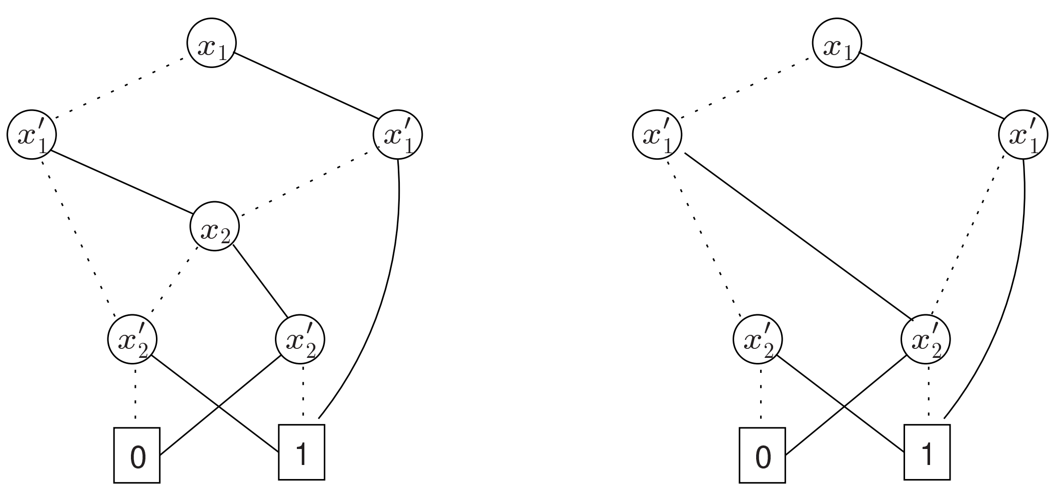
(a) (b) (c)

performing APPLY($\wedge$, B_{left}, B_{middle}), i.e., compute $f_{B_{left}} \wedge f_{B_{middle}}$

Algorithm RESTRICT(B, x, b)

- For each vertex v labeled with variable x:
  - if $b = 1$ then redirect incoming edges to *right*(v)
  - if $b = 0$ then redirect incoming edges to *left*(v)
  - remove vertex v, and (if necessary) reduce (only above v)

RESTRICT



performing RESTRICT(B, x_2, 1): replace x_2 by constant 1

Exists

- Existential quantification over x_i:

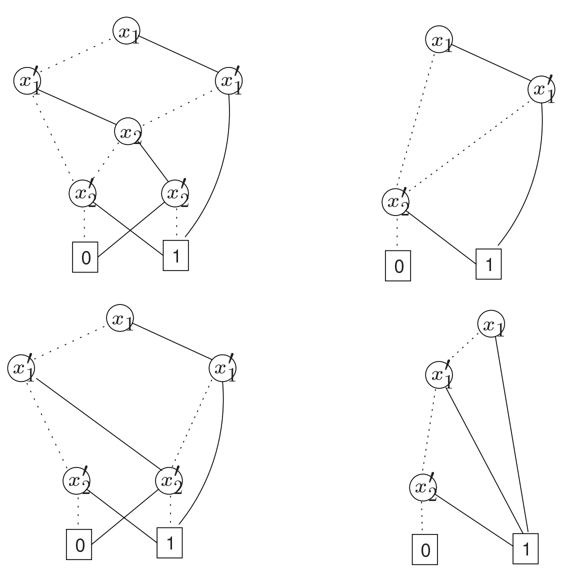
$$\exists x_i.\, f(x_1, \ldots, x_n) = f[x_i := 1] \;\vee\; f[x_i := 0]$$

- Naive realization: APPLY$(\vee, \text{RESTRICT}(\mathsf{B}_f, x_i, 1), \text{RESTRICT}(\mathsf{B}_f, x_i, 0))$

- Efficiency gain:
  - observe that RESTRICT$(\mathsf{B}_f, x_i, 1)$ and RESTRICT$(\mathsf{B}_f, x_i, 0)$ are equal up to x_i
  - . . . the resulting ROBDD also has the same structure up to x_i
  - replace each node labeled with x_i by the result of applying $\vee$ on its children

- This can easily be generalized to $\exists x_1. \ldots \exists x_k.\, f(x_1, \ldots x_n)$

A more involved example



ROBBDs B_f (left up), $\mathsf{B}_{f[x_2:=0]}$ (right up), $\mathsf{B}_{f[x_2:=1]}$ (left down), and $\mathsf{B}_{\exists x_2.\, f}$ (right down)

Operations on ROBDDs

Algorithm	Output	Time complexity	Space complexity
REDUCE	B' (reduced) with $f_B = f_{B'}$	$\mathcal{O}(\lvert B_f\rvert \cdot \log \lvert B_f\rvert)$	$\mathcal{O}(\lvert B_f\rvert)$
NOT	$B_{\neg f}$	$\mathcal{O}(\lvert B_f\rvert)$	$\mathcal{O}(\lvert B_f\rvert)$
APPLY	$B_{f\ op\ g}$	$\mathcal{O}(\lvert B_f\rvert \cdot \lvert B_g\rvert)$	$\mathcal{O}(\lvert B_f\rvert \cdot \lvert B_g\rvert)$
RESTRICT	$B_{f[x:=b]}$	$\mathcal{O}(\lvert B_f\rvert)$	$\mathcal{O}(\lvert B_f\rvert)$
RENAME	$B_{f[x:=y]}$	$\mathcal{O}(\lvert B_f\rvert)$	$\mathcal{O}(\lvert B_f\rvert)$
EXISTS	$B_{\exists x.\ f}$	$\mathcal{O}(\lvert B_f\rvert^2)$	$\mathcal{O}(\lvert B_f\rvert^2)$

operations are only efficient if f and g have compact ROBDD representations

Computing *Sat*(Φ) symbolically

Input: CTL-formula Φ in ENF
Output: ROBDD $B_{Sat(\Phi)}$

switch(Φ):

true	:	**return** CONST(1);
false	:	**return** CONST(0);
x_i	:	**return** ROBDD B_f for $f(x_1, \ldots, x_n) = x_i$;
$\neg\Psi$	:	**return** NOT(*bddSat*(Ψ))
$\Phi_1 \wedge \Phi_2$	:	**return** APPLY($\wedge$, *bddSat*(Φ_1), *bddSat*(Φ_2))
$\exists\bigcirc\Psi$	:	**return** *bddEX*(Ψ);
$\exists(\Phi_1 \mathsf{U} \Phi_2)$	:	**return** *bddEU*(Φ_1, Φ_2)
$\exists\Box\Psi$	:	**return** *bddEG*(Ψ)

end switch

Boolean Transition Systems

- finite set of boolean variables: V
- initial condition θ: boolean function over V
- transitions represented by transition relation: boolean function ρ over $V \cup V'$
  - V: values in present state
  - V': values in next state
- Atomic propositions *AP* $= V$.

The next-step operator

$$\textit{Sat}(\bigcirc\Phi) \;=\; \{\, q \in Q \mid \exists q'.\, (q,q') \in E \text{ and } q' \in \textit{Sat}(\Phi)\,\}$$

Input: CTL-formula Φ in ENF
Output: ROBDD $\mathrm{B}_{Sat(\bigcirc\Phi)}$

B $:=$ *bddSat*(Φ); (* *Sat*(Φ) *)
B $:=$ RENAME$(\mathrm{B}, x_1, \ldots, x_n, x'_1, \ldots, x'_n)$;
B $:=$ APPLY$(\wedge, \mathrm{B}_\rho, \mathrm{B})$; (* *Pre*(*Sat*$(\Phi)$) *)
return EXISTS$(\mathrm{B}, x'_1, \ldots, x'_n)$

Existential until

Input: CTL-formulas Φ, Ψ in ENF
Output: ROBDD $B_{Sat(\exists(\Phi \mathsf{U} \Psi))}$

---

var N, P, B : *ROBDD*;
N := *bddSat*(Ψ);
P := CONST(0);
B := *bddSat*(Φ);
while (N $\neq$ P) **do**
  P := N; (* T_i *)
  N := RENAME(N, $x_1, \ldots, x_n, x'_1, \ldots, x'_n$);
  N := APPLY($\wedge$, B_ρ, N); (* *Pre*(T_i) *)
  N := EXISTS(N, $x'_1, \ldots, x'_n$);
  N := APPLY($\wedge$, N, B); (* *Pre*(T_i) $\cap$ *Sat*(Φ) *)
  N := APPLY($\vee$, P, N); (* $T_{i+1} = T_i \cup \ldots\ldots$ *)
od
return N

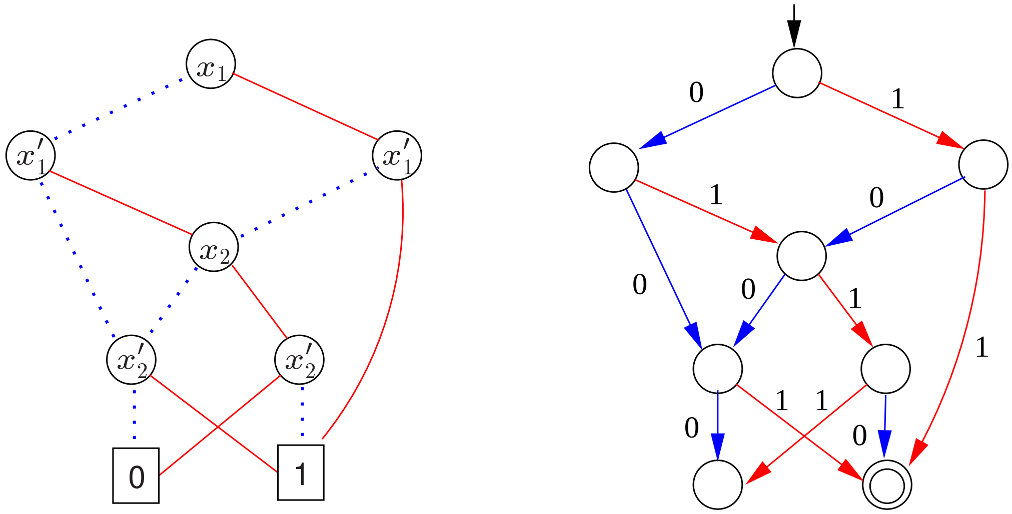
Possibly always

Input: CTL-formula Φ in ENF
Output: ROBDD $B_{Sat(\exists\Box\Phi)}$

---

var N, P, B : *ROBDD*;
B := *bddSat*(Φ);
N := B;
P := CONST(0);
while (N $\neq$ P) **do**
  P := N; (* T_i *)
  N := RENAME(N, $x_1, \ldots, x_n, x'_1, \ldots, x'_n$);
  N := APPLY($\wedge$, B_ρ, N); (* *Pre*(T_i) *)
  N := EXISTS(N, $x'_1, \ldots, x'_n$);
  N := APPLY($\wedge$, N, B); (* *Pre*(T_i) $\cap$ *Sat*(Φ) *)
  N := APPLY($\wedge$, P, N); (* $T_{i+1} = T_i \cap \ldots\ldots$ *)
od
return N

OBDDs versus deterministic automata



each OBDD B *is a deterministic automaton* A_{B} *with* $f_{\mathsf{B}}^{-1}(1) = L(A_{\mathsf{B}})$

Analogies between ROBDDs and deterministic automata

- For language L, a minimized automaton is unique up to isomorphism
  - for a given variable ordering $<$, and function f, an ROBDD is unique upto $\cong$
- $L = L'$? can be checked by verifying isomorphism of their automata
  - $f = f'$? for boolean functions can be checked by verifying $\mathsf{B}_f \cong \mathsf{B}_{f'}$

  $\Rightarrow$ in both cases, efficient algorithms do exist for this
- $L \neq \varnothing$? $\equiv$ is there a reachable accept state?
  - is f satisfiable? $\equiv$ its ROBDD has a reachable leaf 1
- Union, intersection, and complementation on det. automata is efficient
  - disjunction, conjunction, and negation on ROBDDs are efficient

Implementation relations

- A *binary relation* on transition systems
  - when does a transition systems correctly implements another?
- Important for system *synthesis*
  - stepwise *refinement* of a system specification S into an "implementation" S'
- Important for system *analysis*
  - use the implementation relation as a means for *abstraction*
  - replace $S \models \varphi$ by $S' \models \varphi$ where $|S'| \ll |S|$ such that:

$$S \models \varphi \text{ iff } S' \models \varphi \quad \text{or} \quad S' \models \varphi \;\Rightarrow\; S \models \varphi$$

$\Rightarrow$ Focus on state-based *bisimulation* and *simulation*

- logical characterization: which logical formulas are preserved by bisimulation?

Bisimulation equivalence

Let $S_i = (Q_i, Q_{0,i}, E_i, L_i)$, $i{=}1,2$, be two state graphs over *AP*.

A *bisimulation* for (S_1, S_2) is a binary relation $\mathcal{R} \subseteq Q_1 \times Q_2$ such that:

1. $\forall q_1 \in Q_{0,1}\, \exists q_2 \in Q_{0,2}.\, (q_1, q_2) \in \mathcal{R}$ and
   $\forall q_2 \in Q_{0,2}\, \exists q_1 \in Q_{0,1}.\, (q_1, q_2) \in \mathcal{R}$
2. for all states $q_1 \in Q_1$, $q_2 \in Q_2$ with $(q_1, q_2) \in \mathcal{R}$ it holds:
   (a) $L_1(q_1) = L_2(q_2)$
   (b) if $q_1' \in$ *Successors*(q_1) then there exists $q_2' \in$ *Successors*(q_2) with $(q_1', q_2') \in \mathcal{R}$
   (c) if $q_2' \in$ *Successors*(q_2) then there exists $q_1' \in$ *Successors*(q_1) with $(q_1', q_2') \in \mathcal{R}$

S_1 and S_2 are bisimilar, denoted $S_1 \sim S_2$, if there exists a bisimulation for (S_1, S_2)

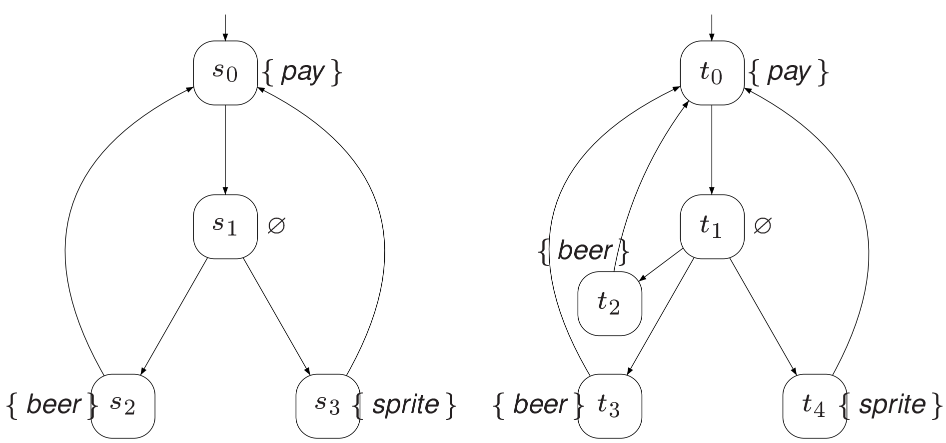
Bisimulation equivalence

$$\begin{array}{ccc} q_1 & \rightarrow & q_1' \\ \mathcal{R} & & \\ q_2 & & \end{array} \quad \text{can be completed to} \quad \begin{array}{ccc} q_1 & \rightarrow & q_1' \\ \mathcal{R} & & \mathcal{R} \\ q_2 & \rightarrow & q_2' \end{array}$$

and

$$\begin{array}{ccc} q_1 & & \\ \mathcal{R} & & \\ q_2 & \rightarrow & q_2' \end{array} \quad \text{can be completed to} \quad \begin{array}{ccc} q_1 & \rightarrow & q_1' \\ \mathcal{R} & & \mathcal{R} \\ q_2 & \rightarrow & q_2' \end{array}$$

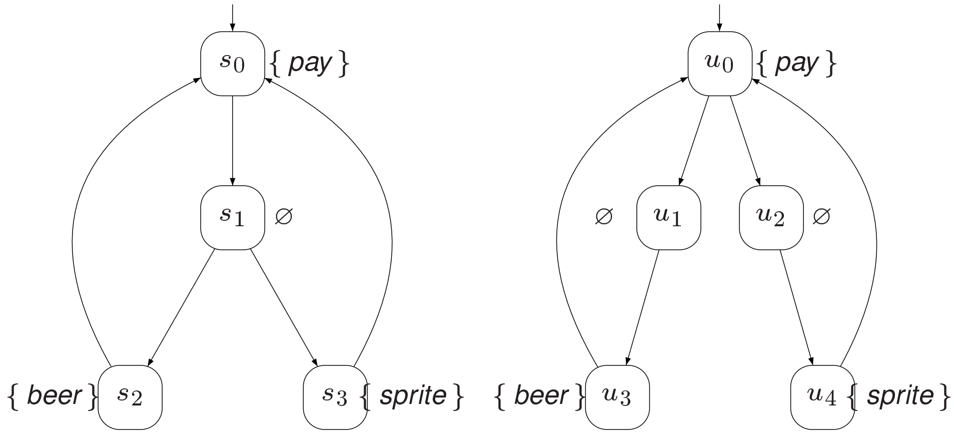
Example (1)



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

is a bisimulation for (S_1, S_2) where $AP = \{ \mathit{pay}, \mathit{beer}, \mathit{sprite} \}$

Example (2)



$S_1 \not\sim S_3$ for $AP = \{ pay, beer, sprite \}$

But: $\{ (s_0, u_0), (s_1, u_1), (s_1, u_2), (s_2, u_3), (s_2, u_4), (s_3, u_3), (s_3, u_4) \}$

is a bisimulation for (S_1, S_3) for $AP = \{ pay, drink \}$

$\sim$ is an equivalence

For any transition systems S, S_1, S_2 and S_3 over AP:

$S \sim S$ (reflexivity)

$S_1 \sim S_2$ implies $S_2 \sim S_1$ (symmetry)

$S_1 \sim S_2$ and $S_2 \sim S_3$ implies $S_1 \sim S_3$ (transitivity)

Bisimulation on paths

Whenever we have:

$$\begin{array}{ccccccccc} s_0 & \to & s_1 & \to & s_2 & \to & s_3 & \to & s_4 \ldots\ldots \\ \mathcal{R} \\ t_0 \end{array}$$

this can be completed to

$$\begin{array}{ccccccccc} s_0 & \to & s_1 & \to & s_2 & \to & s_3 & \to & s_4 \ldots\ldots \\ \mathcal{R} & & \mathcal{R} & & \mathcal{R} & & \mathcal{R} & & \mathcal{R} \\ t_0 & \to & t_1 & \to & t_2 & \to & t_3 & \to & t_4 \ldots\ldots \end{array}$$

proof: by induction on index i of state s_i

Bisimulation vs. trace equivalence

$$S_1 \sim S_2 \quad \text{implies} \quad \mathit{Traces}(S_1) = \mathit{Traces}(S_2)$$

bisimilar transition systems thus satisfy the same LT properties!

Bisimulation on states

$\mathcal{R} \subseteq S \times S$ is a *bisimulation* on *S* if for any $(q_1, q_2) \in \mathcal{R}$:

- $L(q_1) = L(q_2)$
- if $q_1' \in$ *Successors*(q_1) then there exists an $q_2' \in$ *Successors*(q_2) with $(q_1', q_2') \in \mathcal{R}$
- if $q_2' \in$ *Successors*(q_2) then there exists an $q_1' \in$ *Successors*(q_1) with $(q_1', q_2') \in \mathcal{R}$

q_1 and q_2 are *bisimilar*, $q_1 \sim_S q_2$, if $(q_1, q_2) \in \mathcal{R}$ for some bisimulation $\mathcal{R}$ for *S*

$q_1 \sim_S q_2$ if and only if $S_{q_1} \sim S_{q_2}$

Coarsest bisimulation

$\sim_S$ is an equivalence and the coarsest bisimulation for *S*

Quotient state graph

For $S = (Q, Q_0, E, L)$ and bisimulation $\sim_S \subseteq S \times S$ on S let

$$S/\sim_S = (Q', Q'_0, E', L') \quad \text{be the } \textit{quotient} \text{ of } S \text{ under } \sim_S$$

where

- $Q' = S/\sim_S = \{ [q]_\sim \mid q \in Q \}$ with $[q]_\sim = \{ q' \in Q \mid q \sim_S q' \}$
- $Q'_0 = \{ [q]_\sim \mid q \in Q_0 \}$
- $E' = \{([q]_\sim, [q']_\sim) \mid (q, q') \in E\}$
- $L'([q]_\sim) = L(q)$

note that $S \sim S/\sim_S$ Why?