Abstraction – Part 1

Lecture #5 of Principles of Model Checking

Joost-Pieter Katoen

Software Modeling and Verification Group

affiliated to University of Twente, Formal Methods and Tools

University of Twente, September 29, 2010

Content of this lecture

- Bisimulation
  - definition, properties, quotient, CTL* equivalence
- Bisimulation minimisation
  - partition refinement, efficiency improvement, complexity
- Simulation
  - pre-order, simulation equivalence, properties, $\forall$CTL* equivalence
- Checking simulation
  - basic idea of algorithm

Content of this lecture

⇒ Bisimulation

- definition, properties, quotient, CTL^* equivalence

- Bisimulation minimisation
  - partition refinement, efficiency improvement, complexity
- Simulation
  - pre-order, simulation equivalence, properties, $\forall CTL^*$ equivalence
- Checking simulation
  - basic idea of algorithm

Abstraction

Reduce (a huge) TS to (a small) $\widehat{TS}$ prior or during model checking

Relevant issues:

- What is the formal relationship between TS and $\widehat{TS}$?
- Can $\widehat{TS}$ be obtained algorithmically and efficiently?
- Which logical fragment (of LTL, CTL, CTL*) is preserved?
- And in what sense?
  - "strong" preservation: positive and negative results carry over
  - "weak" preservation: only positive results carry over
  - "match": logic equivalence coincides with formal relation

Bisimulation

$\mathcal{R} \subseteq S \times S$ is a *bisimulation* on *TS* if for any $(s_1, s_2) \in \mathcal{R}$:

- $L(s_1) = L(s_2)$
- if $s_1' \in \mathit{Post}(s_1)$ then there exists an $s_2' \in \mathit{Post}(s_2)$ with $(s_1', s_2') \in \mathcal{R}$
- if $s_2' \in \mathit{Post}(s_2)$ then there exists an $s_1' \in \mathit{Post}(s_1)$ with $(s_1', s_2') \in \mathcal{R}$

s_1 and s_2 are *bisimilar*, $s_1 \sim_{\mathit{TS}} s_2$, if $(s_1, s_2) \in \mathcal{R}$ for some bisimulation $\mathcal{R}$ for *TS*

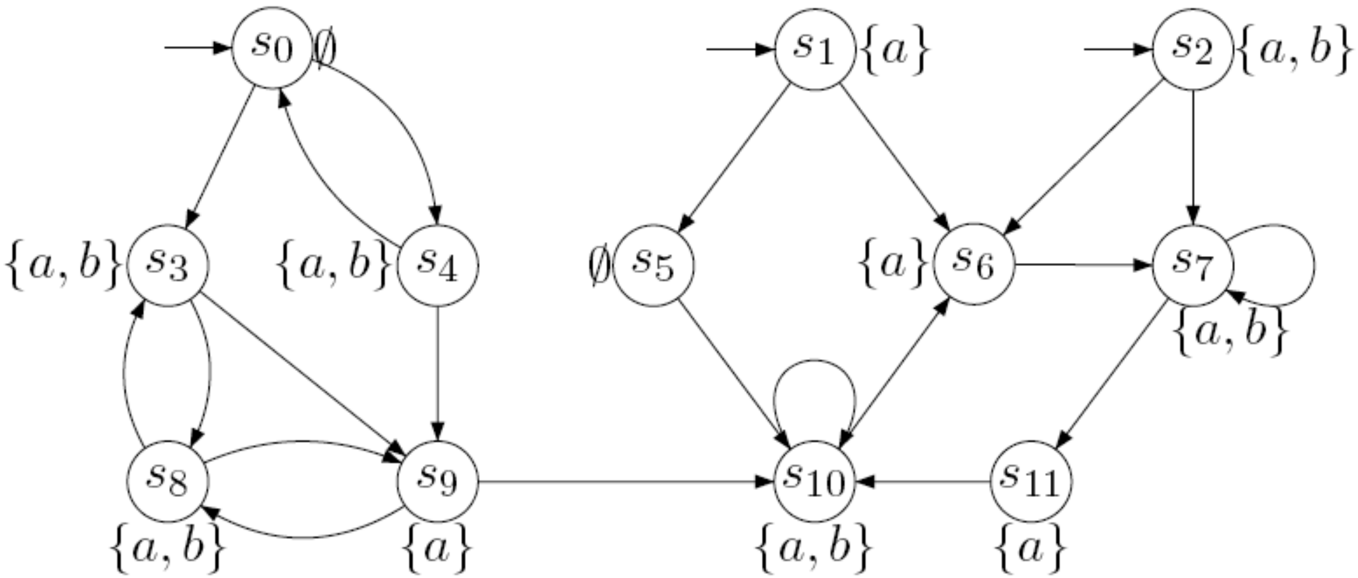
Bisimulation

$$\begin{array}{ccc} s_1 & \longrightarrow & s_1' \\ \mathcal{R} & & \\ s_2 & & \end{array} \quad \text{can be completed to} \quad \begin{array}{ccc} s_1 & \longrightarrow & s_1' \\ \mathcal{R} & & \color{red}{\mathcal{R}} \\ s_2 & \color{red}{\longrightarrow} & \color{red}{s_2'} \end{array}$$

and

$$\begin{array}{ccc} s_1 & & \\ \mathcal{R} & & \\ s_2 & \longrightarrow & s_2' \end{array} \quad \text{can be completed to} \quad \begin{array}{ccc} s_1 & \color{red}{\longrightarrow} & \color{red}{s_1'} \\ \mathcal{R} & & \color{red}{\mathcal{R}} \\ s_2 & \longrightarrow & s_2' \end{array}$$

Example



determine the bisimulation relation $\sim_{TS}$

Bisimulation on paths

For any bisimulation relation $\mathcal{R}$, whenever we have:

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

this can be completed to

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

proof: by induction on the length of a path

Bisimulation of transition systems

$TS_1 \sim TS_2$, if there exists a bisimulation $\mathcal{R}$ on $TS_1 \oplus TS_2$ such that:

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

Properties

$$TS_1 \sim TS_2 \quad \text{implies} \quad \mathit{Traces}(TS_1) = \mathit{Traces}(TS_2)$$

$$TS_1 \sim TS_2 \quad \text{implies} \quad TS_1 \models P \ \text{ iff } \ TS_2 \models P \text{ for any LT property } P$$

$$TS_1 \sim TS_2 \quad \text{implies} \quad TS_1 \models \varphi \ \text{ iff } \ TS_2 \models \varphi \text{ for any LTL formula } \varphi$$

Quotient transition system

Let $TS = (S, Act, \rightarrow, I, AP, L)$ and bisimulation $\mathcal{R} \subseteq S \times S$ be an *equivalence*

The *quotient* of TS under $\mathcal{R}$ is defined by:

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

where

- $S' = S/\mathcal{R} \;=\; \{\,[s]_{\mathcal{R}} \mid s \in S\,\}$ with $[s]_{\mathcal{R}} \;=\; \{\,s' \in S \mid (s, s') \in \mathcal{R}\,\}$
- $I' = \{\,[s]_{\mathcal{R}} \mid s \in I\,\}$
- $L'([s]_{\mathcal{R}}) = L(s)$
- $\rightarrow'$ is defined by: $\dfrac{s \xrightarrow{\alpha} s'}{[s]_{\mathcal{R}} \xrightarrow{\tau}{}' [s']_{\mathcal{R}}}$

note that $TS \;\sim\; TS/\mathcal{R}$ Why?

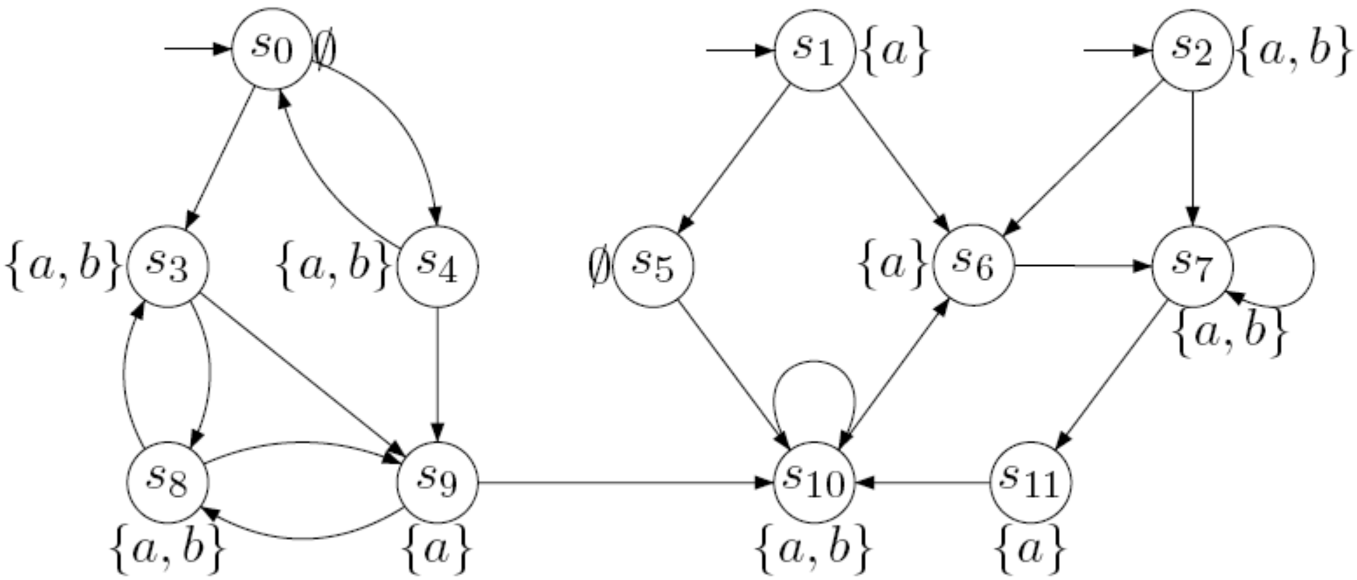
Coarsest bisimulation

$\sim_{TS}$ is a bisimulation, an equivalence,

and the coarsest bisimulation for TS

The quotient under $\sim_{TS}$ is the smallest
under any bisimulation relation

Example



determine the (coarsest) bisimulation quotient $TS/_{\sim_{TS}}$

The simplified bakery algorithm

Process 1:

$\ldots\ldots$

while true $\{$

$\ldots\ldots$

n_1 : $x_1 := x_2 + 1;$

w_1 : **wait until**$(x_2 = 0 \,||\, x_1 < x_2)\ \{$

c_1 : $\ldots$ critical section $\ldots\}$

$x_1 := 0;$

$\ldots\ldots$

$\}$

Process 2:

$\ldots\ldots$

while true $\{$

$\ldots\ldots$

n_2 : $x_2 := x_1 + 1;$

w_2 : **wait until**$(x_1 = 0 \,||\, x_2 < x_1)\ \{$

c_2 : $\ldots$ critical section $\ldots\}$

$x_2 := 0;$

$\ldots\ldots$

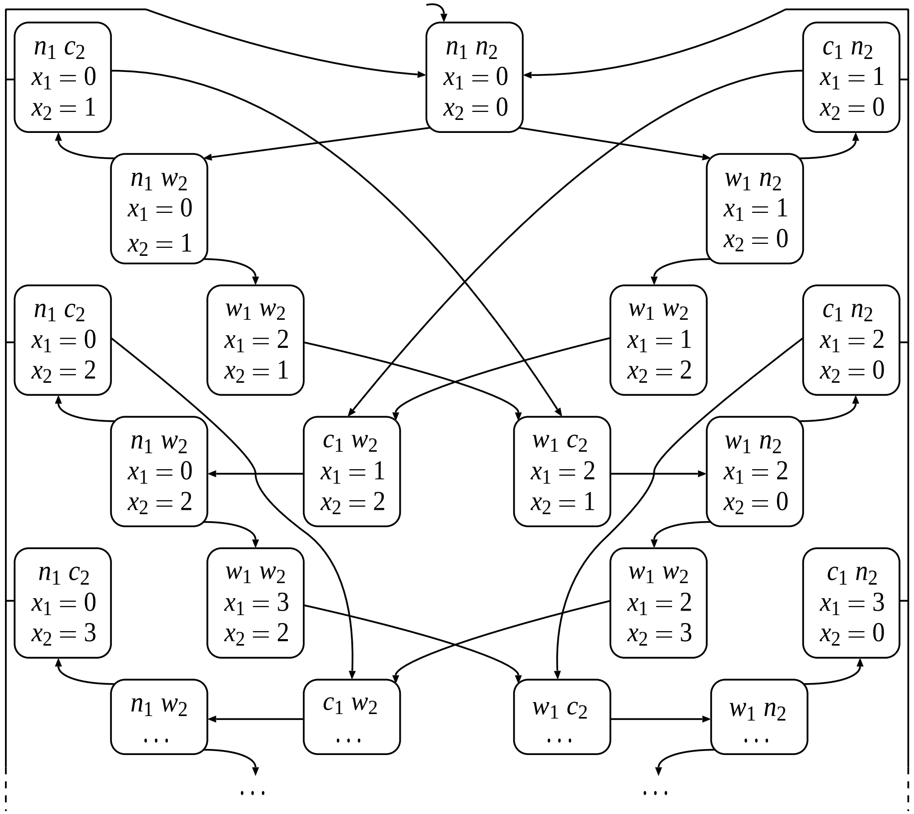
$\}$

this algorithm can be applied to arbitrarily many processes

Example run of bakery algorithm

process P_1	process P_2	x_1	x_2	effect
n_1	n_2	0	0	P_1 requests access to critical section
w_1	n_2	1	0	P_2 requests access to critical section
w_1	w_2	1	2	P_1 enters the critical section
c_1	w_2	1	2	P_1 leaves the critical section
n_1	w_2	0	2	P_1 requests access to critical section
w_1	w_2	3	2	P_2 enters the critical section
w_1	c_2	3	2	P_2 leaves the critical section
w_1	n_2	3	0	P_2 requests access to critical section
w_1	w_2	3	4	P_2 enters the critical section
...	...	..	..	...

Bakery algorithm as transition system



infinite state space due to possible unbounded increase of counters

Bisimulation

Function f maps a reachable state of TS_{Bak} onto an abstract one in TS_{Bak}^{abs}

Let $s = \langle \ell_1, \ell_2, x_1 = b_1, x_2 = b_2 \rangle$ be a state of TS_{Bak} with $\ell_i \in \{ n_i, w_i, c_i \}$ and $b_i \in \mathbb{N}$

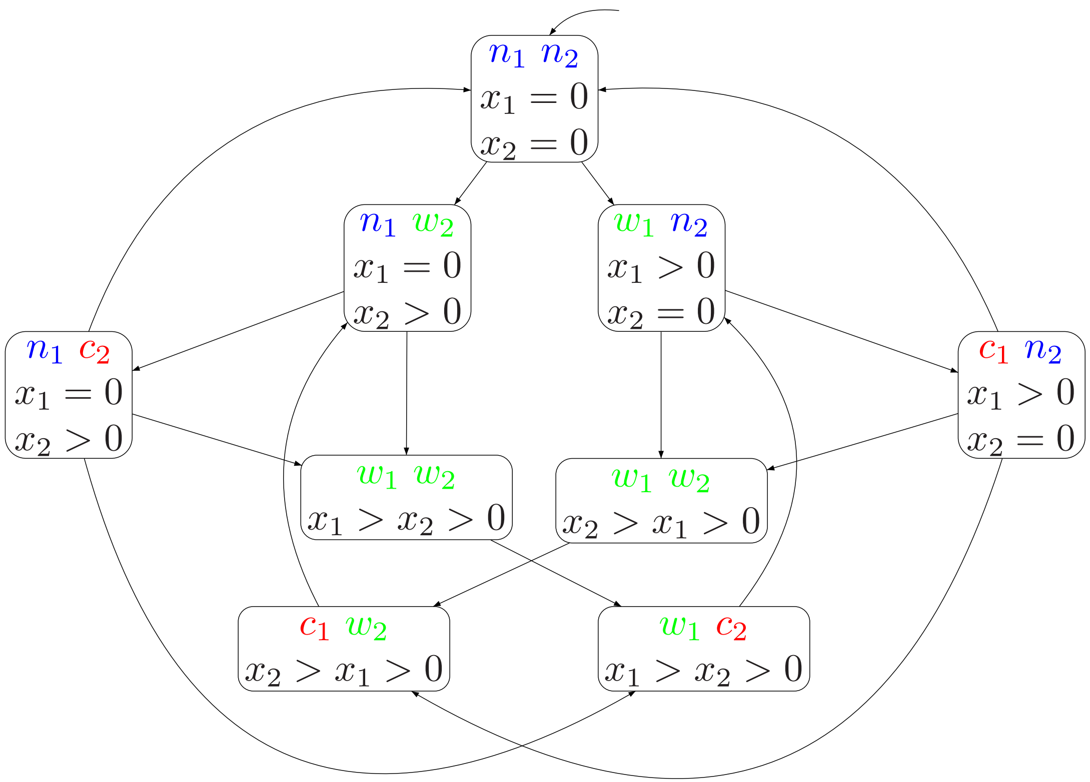
Then:

$$f(s) = \begin{cases} \langle \ell_1, \ell_2, x_1 = 0, x_2 = 0 \rangle & \text{if } b_1 = b_2 = 0 \\ \langle \ell_1, \ell_2, x_1 = 0, x_2 > 0 \rangle & \text{if } b_1 = 0 \text{ and } b_2 > 0 \\ \langle \ell_1, \ell_2, x_1 > 0, x_2 = 0 \rangle & \text{if } b_1 > 0 \text{ and } b_2 = 0 \\ \langle \ell_1, \ell_2, x_1 > x_2 > 0 \rangle & \text{if } b_1 > b_2 > 0 \\ \langle \ell_1, \ell_2, x_2 > x_1 > 0 \rangle & \text{if } b_2 > b_1 > 0 \end{cases}$$

It follows: $\mathcal{R} = \{ (s, f(s)) \mid s \in S \}$ is a bisimulation for $(TS_{Bak}, TS_{Bak}^{abs})$

for any subset of $AP = \{ noncrit_i, wait_i, crit_i \mid i = 1, 2 \}$

Bisimulation quotient



bisimulation quotient under $\sim_{TS}$ for $AP = \{ \, crit_1, crit_2, wait_1, wait_2 \, \}$

Syntax of CTL*

CTL* *state-formulas* are formed according to:

$$\Phi ::= \text{true} \;\Big|\; a \;\Big|\; \Phi_1 \wedge \Phi_2 \;\Big|\; \neg\Phi \;\Big|\; \exists\varphi$$

where $a \in AP$ and φ is a path-formula

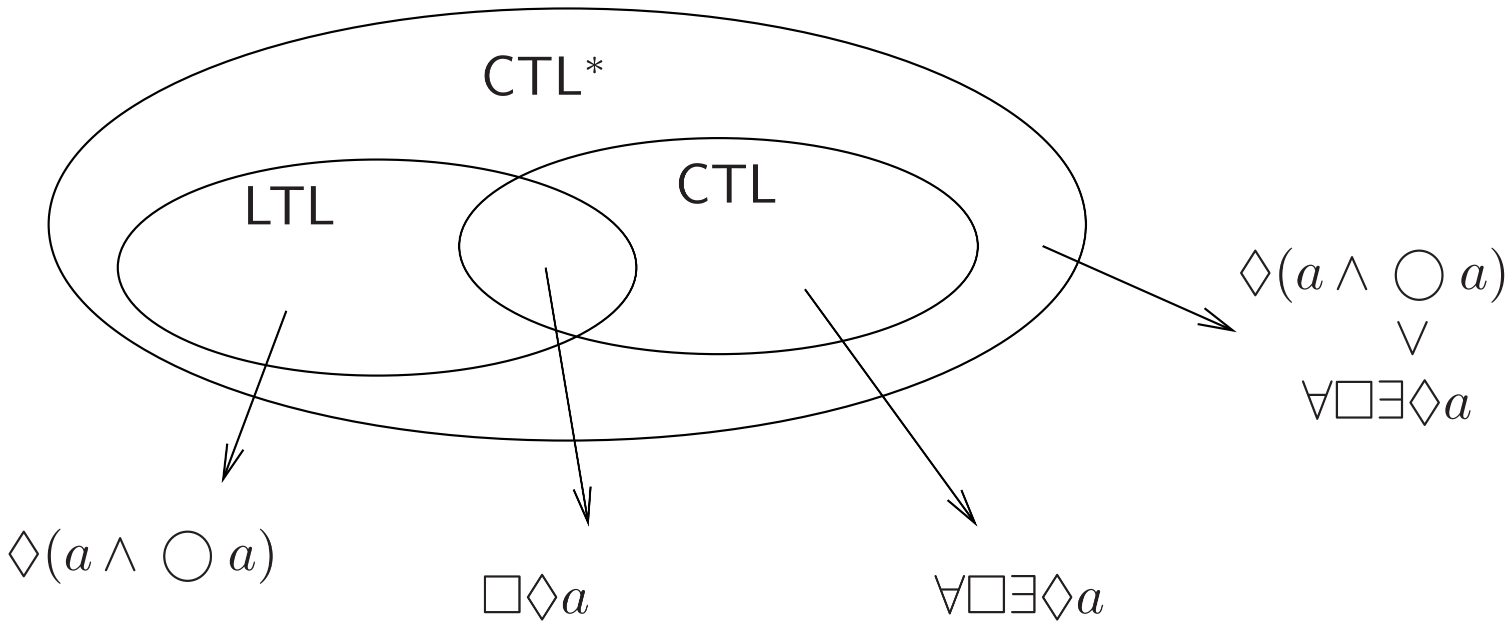
CTL* *path-formulas* are formed according to the grammar:

$$\varphi ::= \Phi \;\Big|\; \varphi_1 \wedge \varphi_2 \;\Big|\; \neg\varphi \;\Big|\; \bigcirc\varphi \;\Big|\; \varphi_1 \,\mathsf{U}\, \varphi_2$$

where Φ is a state-formula, and φ, φ_1 and φ_2 are path-formulas

in CTL*: $\forall\varphi \;=\; \neg\exists\neg\varphi$. This does not hold in CTL!

Relationship between LTL, CTL and CTL*



CTL* equivalence

States s_1 and s_2 in *TS* (over *AP*) are CTL*-equivalent:

$$s_1 \equiv_{\mathsf{CTL}^*} s_2 \quad \text{if and only if} \quad (s_1 \models \Phi \;\text{ iff }\; s_2 \models \Phi)$$

for all CTL* state formulas over *AP*

$$TS_1 \equiv_{\mathsf{CTL}^*} TS_2 \quad \text{if and only if} \quad (TS_1 \models \Phi \;\text{ iff }\; TS_2 \models \Phi)$$

for any sublogic of CTL*, logical equivalence is defined analogously

Bisimulation vs. CTL* and CTL equivalence

For any finitely branching transition system *TS* and s, s' states in *TS*:

$$s \sim_{TS} s' \quad \text{iff} \quad s \equiv_{\mathsf{CTL}} s' \quad \text{iff} \quad s \equiv_{\mathsf{CTL}^*} s' \quad \text{iff} \quad s \equiv_{\mathsf{CTL}\setminus_{\mathsf{U}}} s'$$

this is proven in three steps: $\equiv_{\mathsf{CTL}} \;\subseteq\; \sim_{TS} \;\subseteq\; \equiv_{\mathsf{CTL}^*} \;\subseteq\; \equiv_{\mathsf{CTL}}$

Corollary

For any finitely branching transition systems TS and TS':

$$TS \sim TS' \quad \text{if and only if} \quad TS \equiv_{\mathrm{CTL}} TS' \quad \text{if and only if} \quad TS \equiv_{\mathrm{CTL}^*} TS'$$

$\Rightarrow$ prior to model-check CTL-formula Φ, first minimize TS wrt. $\sim$

Content of this lecture

- Bisimulation
  - definition, properties, quotient, CTL* equivalence

⇒ Bisimulation minimisation
  - partition refinement, efficiency improvement, complexity

- Simulation
  - pre-order, simulation equivalence, properties, $\forall$CTL* equivalence

- Checking simulation
  - basic idea of algorithm

Partitions

- A partition $\Pi = \{ B_1, \ldots, B_k \}$ of S satisfies:
  - B_i is non-empty; B_i is called a *block*
  - $B_i \cap B_j = \varnothing$ for all i, j with $i \neq j$
  - $B_1 \cup \ldots \cup B_k = S$
- $C \subseteq S$ is a *super-block* of partition Π of S if

$$C = B_{i_1} \cup \ldots \cup B_{i_l} \quad \text{for } B_{i_j} \in \Pi \text{ for } 0 < j \leqslant l$$

- Partition Π is *finer than* partition Π' if:

$$\forall B \in \Pi.\ (\exists B' \in \Pi'.\ B \subseteq B')$$

$\Rightarrow$ each block of Π' equals the disjoint union of a set of blocks in Π
- Π is strictly finer than Π' if it is finer than Π' and $\Pi \neq \Pi'$

Partitions and equivalences

- $\mathcal{R}$ is an equivalence on $S \quad \Rightarrow \quad S/\mathcal{R}$ is a partition of S
- Partition $\Pi = \{ B_1, \ldots, B_k \}$ of S induces the equivalence relation

$$\mathcal{R}_\Pi = \{ (s,t) \mid \exists B_i \in \Pi.\, s \in B_i \;\wedge\; t \in B_i \}$$

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

$\Rightarrow$ there is a one-to-one relationship between partitions and equivalences

Skeleton for bisimulation checking

from now on, we assume that *TS* is finite

- Iteratively compute a partition of S
- Initially: Π_0 equals $\Pi_{AP} = \{ (s,t) \in S \times S \mid L(s) = L(t) \}$
- Repeat until no change: $\boxed{\Pi_{i+1} := \mathit{Refine}(\Pi_i)}$
  - loop invariant: Π_i is coarser than $S/\sim$ and finer than $\{ S \}$
- Return Π_i
  - termination: $S \times S \supseteq \mathcal{R}_{\Pi_0} \supsetneq \mathcal{R}_{\Pi_1} \supsetneq \mathcal{R}_{\Pi_2} \supsetneq \ldots \supsetneq \mathcal{R}_{\Pi_i} = \sim_{TS}$
  - time complexity: maximally $|\, S \,|$ iterations needed (why?)

this is a partition-refinement algorithm

Theorem

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

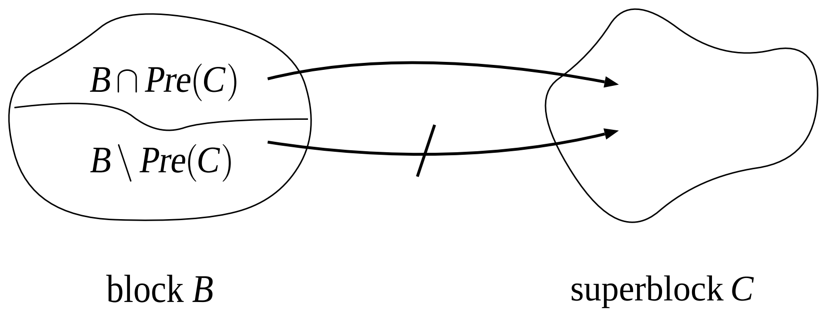
   (i) Π is finer than the initial partition Π_{AP}, and

   (ii) $B \cap Pre(C) = \varnothing$ or $B \subseteq Pre(C)$ for all $B, C \in \Pi$

2. If (ii) holds for Π, then it holds for all $B \in \Pi$ and all superblocks C of Π

The refinement operator

- Let: $\mathit{Refine}(\Pi, C) = \bigcup_{B \in \Pi} \mathit{Refine}(B, C)$ for C a superblock of Π
  - where $\mathit{Refine}(B, C) = \left\{ B \cap \mathit{Pre}(C),\ B \setminus \mathit{Pre}(C) \right\} \setminus \{\varnothing\}$



- Basic properties:
  - for Π finer than Π_{AP} and coarser than $S/{\sim}$:

    $\mathit{Refine}(\Pi, C)$ is finer than Π and $\mathit{Refine}(\Pi, C)$ is coarser than $S/{\sim}$
  - Π is strictly coarser than $S/{\sim}$ if and only if there exists a *splitter* for Π

Splitters

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

- C is a splitter of Π if for some $B \in \Pi$:

$$B \cap \mathit{Pre}(C) \neq \varnothing \;\wedge\; B \setminus \mathit{Pre}(C) \neq \varnothing$$

- Block B is stable wrt. C if

$$B \cap \mathit{Pre}(C) = \varnothing \;\wedge\; B \setminus \mathit{Pre}(C) = \varnothing$$

- Π is stable wrt. C if any $B \in \Pi$ is stable wrt. C

Algorithm skeleton

Input: finite transition system *TS* over *AP* with state space S
Output: bisimulation quotient space $S/\sim$

$\Pi := \Pi_{AP}$;
while there 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 Π *)
od
return Π

Which splitter to take?

How to determine a splitter for partition Π_{i+1}?

1. Simple strategy: $\mathcal{O}(|S| \cdot M)$

   use any block of Π_i as splitter candidate

2. Advanced strategy: $\mathcal{O}(\log |S| \cdot M)$

   use only "smaller" blocks of Π_i as splitter candidates
   and apply "simultaneous" refinement

Advanced strategy

- Not necessary to refine with respect to *all* blocks $C \in \Pi_{old}$

$\Rightarrow$ Consider only the "smaller" subblocks of a previous refinement

- Step i: refine C' into $C_1 = C' \cap \mathit{Pre}(D)$ and $C_2 = C' \setminus \mathit{Pre}(D)$

- Step $i{+}1$: use the *smallest* $C \in \{ C_1, C_2 \}$ as splitter
  - let C be such that $|C| \leqslant |C'|/2$, thus $|C| \leqslant |C' \setminus C|$
  - combine the refinement steps with respect to C and $C' \setminus C$

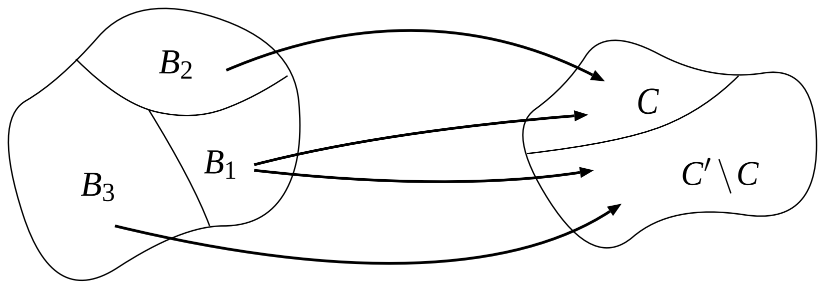
- $\mathit{Refine}(\Pi, C, C' \setminus C) = \mathit{Refine}\Big(\mathit{Refine}(\Pi, C),\ C' \setminus C \Big)$ where $|C| \leqslant |C' \setminus C|$
  - the decomposed blocks are stable with respect to C and $C' \setminus C$

The new refinement operator

- Let: $\mathit{Refine}(\Pi, C, C' \setminus C) \;=\; \bigcup_{B \in \Pi} \mathit{Refine}(B, C, C' \setminus C)$
  - where $\mathit{Refine}(B, C, C' \setminus C) \;=\; \{\, B_1, B_2, B_3 \,\} \setminus \{\, \varnothing \,\}$ with:

$$\begin{array}{lcll} B_1 & = & B \cap \mathit{Pre}(C) \cap \mathit{Pre}(C' \setminus C) & \text{to both } C \text{ and } C \setminus C' \\ B_2 & = & (B \cap \mathit{Pre}(C)) \setminus \mathit{Pre}(C' \setminus C) & \text{only to } C \\ B_3 & = & (B \cap \mathit{Pre}(C' \setminus C)) \setminus \mathit{Pre}(C) & \text{only to } C' \setminus C \end{array}$$

$\Rightarrow$ blocks B_1, B_2, B_3 are stable with respect to C and $C' \setminus C$



block B

Improved partition-refinement algorithm

Input: finite transition system *TS* with state space S
Output: bisimulation quotient space $S/\sim$

$\Pi_{old} := \{ S \}$;
$\Pi := \mathit{Refine}(\Pi_{AP}, S)$;

(* loop invariant: Π is coarser than $S/\sim$ and finer than Π_{AP} and Π_{old}, *)
(* and Π is stable with respect to any block in Π_{old} *)

repeat
  choose block $C' \in \Pi_{old} \setminus \Pi$ and block $C \in \Pi$ with $C \subseteq C'$ and $|C| \leqslant \frac{|C'|}{2}$;
  $\Pi_{old} := \Pi$;
  $\Pi := \mathit{Refine}(\Pi, C, C' \setminus C)$;
until $\Pi = \Pi_{old}$
return Π

Content of this lecture

- Bisimulation
  - definition, properties, quotient, CTL^* equivalence
- Bisimulation minimisation
  - partition refinement, efficiency improvement, complexity

$\Rightarrow$ Simulation
  - pre-order, simulation equivalence, properties, $\forall\mathrm{CTL}^*$ equivalence
- Checking simulation
  - basic idea of algorithm

Simulation relation

- $\mathcal{R} \subseteq S \times S$ is a simulation relation on *TS* if for any $(s_1, s_2) \in \mathcal{R}$:
  - $L(s_1) = L(s_2)$
  - if $s_1' \in Post(s_1)$ then there exists an $s_2' \in Post(s_2)$ with $(s_1', s_2') \in \mathcal{R}$
- s_2 simulates s_1, written $s_1 \preceq_{TS} s_2$
  - if $(s_1, s_2) \in \mathcal{R}$ for some simulation relation $\mathcal{R}$ on *TS*
- $TS_1 \preceq TS_2$ iff $\forall s_1 \in I_1. \exists s_2 \in I_2. \, s_1 \preceq_{TS_1 \oplus TS_2} s_2$

Facts: $\preceq_{TS}$ is a preorder and the coarsest simulation for *TS*

Simulation order

s_1	$\longrightarrow$	s_1'		s_1	$\longrightarrow$	s_1'
$\mathcal{R}$			can be completed to	$\mathcal{R}$		$\mathcal{R}$
s_2				s_2	$\longrightarrow$	s_2'

but <u>not</u> necessarily:

s_1				s_1	$\longrightarrow$	s_1'
$\mathcal{R}$			can be completed to	$\mathcal{R}$		$\mathcal{R}$
s_2	$\longrightarrow$	s_2'		s_2	$\longrightarrow$	s_2'

Abstraction function

- $f : S \longrightarrow \widehat{S}$ is an *abstraction function* if $f(s) = f(s') \;\Rightarrow\; L(s) = L(s')$
  - S is a set of concrete states and $\widehat{S}$ a set of abstract states, i.e. $|\widehat{S}| \ll |S|$
- Abstraction functions are useful for:
  - data abstraction: abstract from values of program or control variables

    $$f \;:\; \text{concrete data domain} \longrightarrow \text{abstract data domain}$$

  - predicate abstraction: use predicates over the program variables

    $$f \;:\; \text{state} \longrightarrow \text{valuations of the predicates}$$

  - localization reduction: partition program variables into visible and invisible

    $$f \;:\; \text{all variables} \longrightarrow \text{visible variables}$$

Abstract transition system

For $TS = (S, Act, \rightarrow, I, AP, L)$ and abstraction function $f : S \rightarrow \widehat{S}$ let:

$$TS_f \;=\; (\widehat{S}, Act, \rightarrow_f, I_f, AP, L_f), \qquad \text{the } \textit{abstraction} \text{ of } TS \text{ under } f$$

where

- $\rightarrow_f$ is defined by: $$\frac{s \xrightarrow{\alpha} s'}{f(s) \xrightarrow{\alpha}_f f(s')}$$
- $I_f = \{\, f(s) \mid s \in I \,\}$
- $L_f(f(s)) = L(s)$; for $s \in \widehat{S} \setminus f(S)$, labeling is undefined

Abstract transition system

For $TS = (S, Act, \rightarrow, I, AP, L)$ and abstraction function $f : S \rightarrow \widehat{S}$ let:

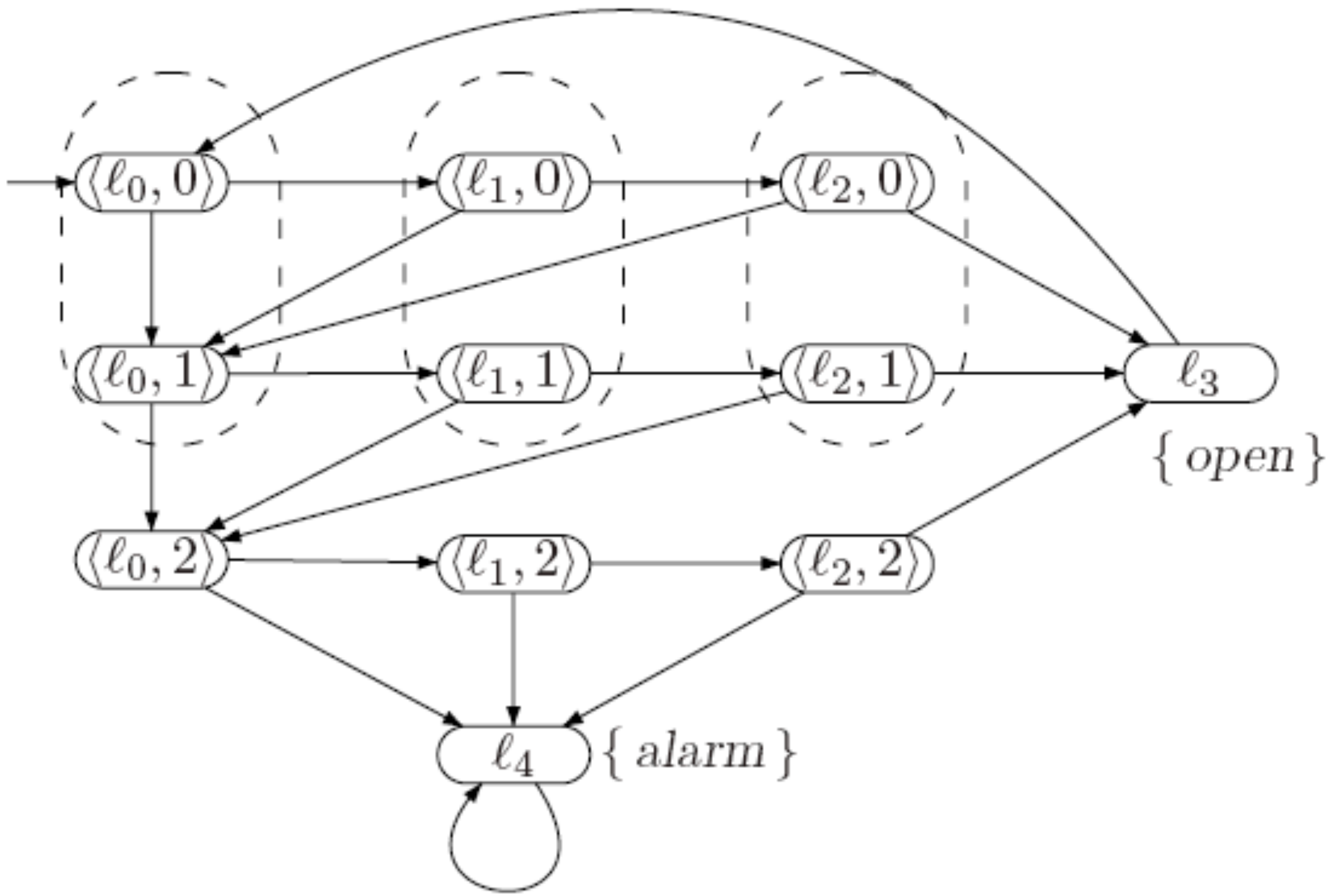
$$TS_f \;=\; (\widehat{S}, Act, \rightarrow_f, I_f, AP, L_f), \qquad \text{the } \textit{abstraction} \text{ of } TS \text{ under } f$$

where

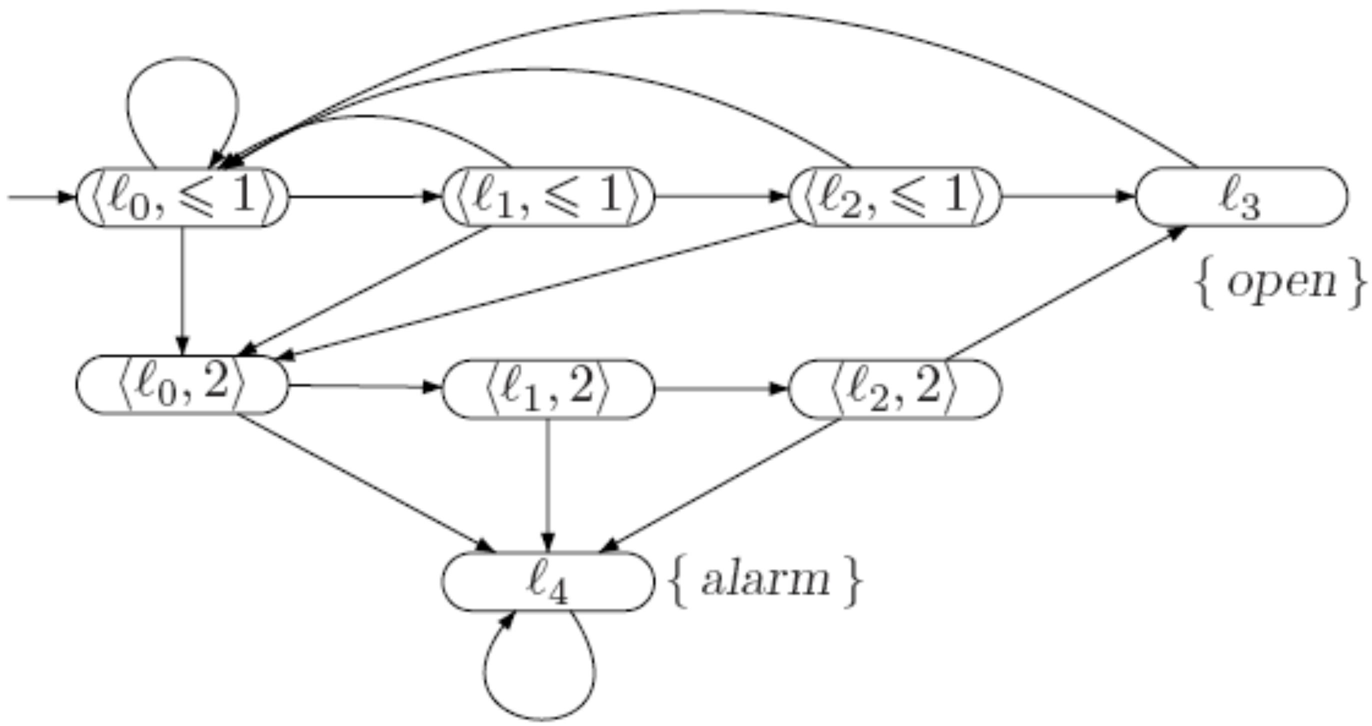
- $\rightarrow_f$ is defined by: $$\frac{s \xrightarrow{\alpha} s'}{f(s) \xrightarrow{\alpha}_f f(s')}$$
- $I_f = \{ f(s) \mid s \in I \}$
- $L_f(f(s)) = L(s)$; for $s \in \widehat{S} \setminus f(S)$, labeling is undefined

$\mathcal{R} = \{ (s, f(s)) \mid s \in S \}$ *is a simulation for* (TS, TS_f)

Abstraction example



Abstraction example



Simulation equivalence

TS_1 and TS_2 are *simulation equivalent*, denoted $TS_1 \simeq TS_2$,

if $TS_1 \preceq TS_2$ and $TS_2 \preceq TS_1$

Simulation quotient

For $\mathit{TS} = (S, \mathit{Act}, \rightarrow, I, \mathit{AP}, L)$ and simulation equivalence $\simeq\ \subseteq S \times S$ let

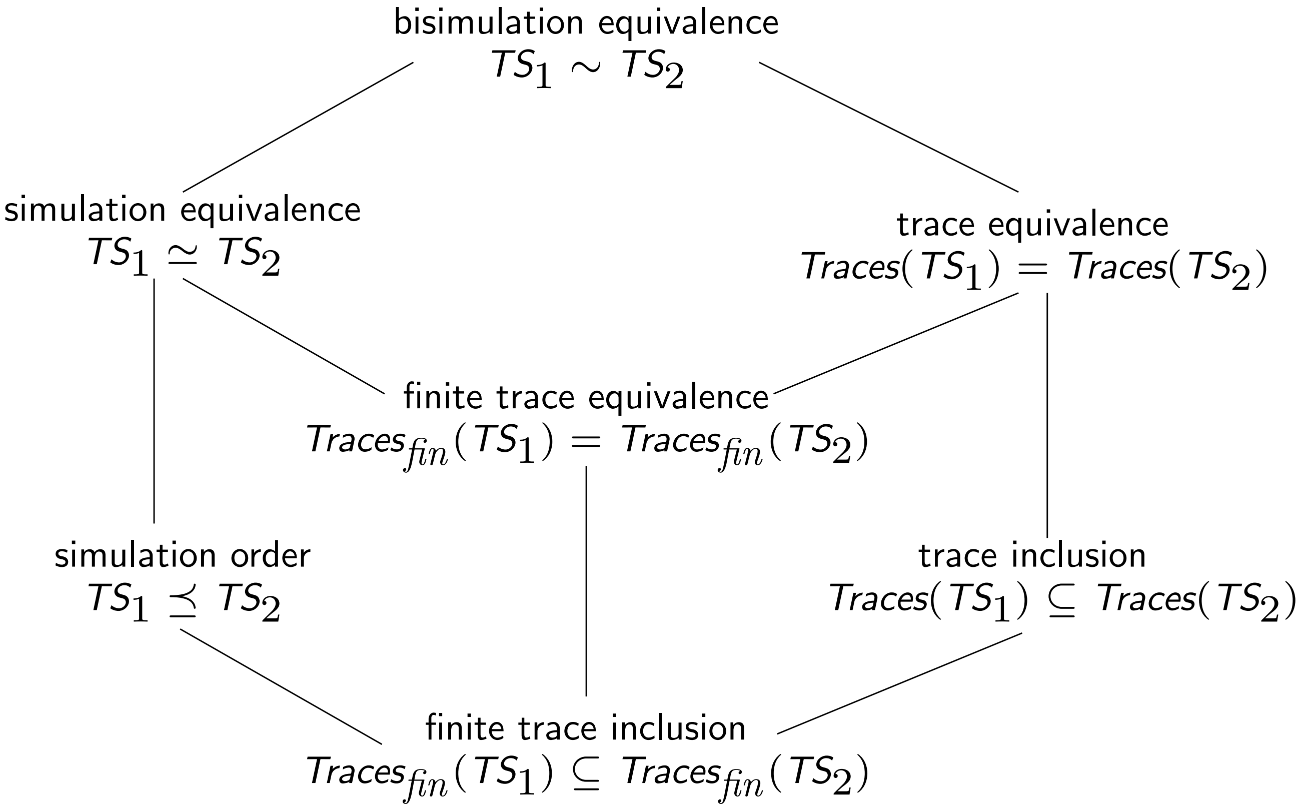
$$\mathit{TS}/\simeq\ =\ (S', \{\tau\}, \rightarrow', I', \mathit{AP}, L'), \qquad \text{the } \textit{quotient} \text{ of } \mathit{TS} \text{ under } \simeq$$

where

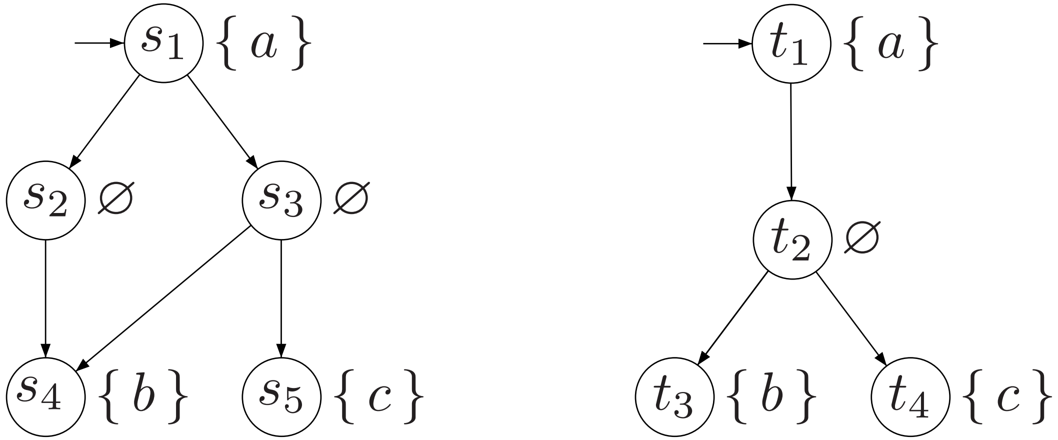
- $S' = S/\simeq = \{ [s]_\simeq \mid s \in S \}$ and $I' = \{ [s]_\simeq \mid s \in I \}$
- $\rightarrow'$ is defined by: $$\frac{s \xrightarrow{\alpha} s'}{[s]_\simeq \xrightarrow{\tau}{}' [s']_\simeq}$$
- $L'([s]_\simeq) = L(s)$

it follows that $\mathit{TS} \simeq \mathit{TS}/\simeq$

Trace, bisimulation, and simulation equivalence



Similar but not bisimilar



$TS_{left} \simeq TS_{right}$ but $TS_{left} \not\sim TS_{right}$

Simulation vs. trace equivalence

- $TS_1 \simeq TS_2$ implies $Traces_{fin}(TS_1) = Traces_{fin}(TS_2)$
- If TS_1 and TS_2 do not have terminal states:

$$TS_1 \preceq TS_2 \quad \text{implies} \quad Traces(TS_1) \subseteq Traces(TS_2)$$

- If TS_1 and TS_2 are *AP*-deterministic:

$$TS_1 \simeq TS_2 \quad \text{iff} \quad Traces(TS_1) = Traces(TS_2) \quad \text{iff} \quad TS_1 \sim TS_2$$

TS is *AP*-deterministic if there all initial states are labeled differently,
and this also applies to all direct successors of any state in *TS*

Logical characterization of $\preceq_{TS}$

- Negation of formulas is problematic as $\preceq_{TS}$ is not symmetric
- Let $\mathbf{L}$ be a fragment of CTL* which is closed under negation
- And assume $\mathbf{L}$ weakly matches $\preceq_{TS}$, that is:

$$s_1 \preceq_{TS} s_2 \quad \text{iff} \quad \text{for all state formulae } \Phi \text{ of } \mathbf{L}\text{: } s_2 \models \Phi \implies s_1 \models \Phi.$$

- Let $s_1 \preceq_{TS} s_2$. Then, for any state formula Φ of $\mathbf{L}$:

$$s_1 \models \Phi \implies s_1 \not\models \neg\Phi \implies s_2 \not\models \neg\Phi \implies s_2 \models \Phi.$$

- Hence, $s_2 \preceq_{TS} s_1$ which requires $\preceq_{TS}$ to be symmetric

Universal fragment of CTL*

$\forall$CTL* *state-formulas* are formed according to:

$$\Phi \ ::= \ \text{true} \ \Big| \ \text{false} \ \Big| \ a \ \Big| \ \neg a \ \Big| \ \Phi_1 \wedge \Phi_2 \ \Big| \ \Phi_1 \vee \Phi_2 \ \Big| \ \forall \varphi$$

where $a \in AP$ and φ is a path-formula

$\forall$CTL* *path-formulas* are formed according to:

$$\varphi \ ::= \ \Phi \ \Big| \ \bigcirc \varphi \ \Big| \ \varphi_1 \wedge \varphi_2 \ \Big| \ \varphi_1 \vee \varphi_2 \ \Big| \ \varphi_1 \, \mathsf{U} \, \varphi_2 \ \Big| \ \varphi_1 \, \mathsf{R} \, \varphi_2$$

where Φ is a state-formula, and φ, φ_1 and φ_2 are path-formulas

in $\forall$CTL, the only path operators are $\bigcirc \Phi$, $\Phi_1 \, \mathsf{U} \, \Phi_2$ and $\Phi_1 \, \mathsf{R} \, \Phi_2$

Universal CTL* contains LTL

For every LTL formula there exists an equivalent $\forall$CTL* formula

Simulation order and $\forall$CTL*

For any finitely branching transition system *TS* and s, s' states in *TS*:

(1) $s \preceq_{TS} s'$ iff

(2) for any $\forall$CTL*-formula Φ: $s' \models \Phi$ implies $s \models \Phi$ iff

(3) for any $\forall$CTL-formula Φ: $s' \models \Phi$ implies $s \models \Phi$ iff

(4) for any $\forall\text{CTL}\backslash_{\text{U, R}}$-formula Φ: $s' \models \Phi$ implies $s \models \Phi$

Content of this lecture

- Bisimulation
  - definition, properties, quotient, CTL* equivalence
- Bisimulation minimisation
  - partition refinement, efficiency improvement, complexity
- Simulation
  - pre-order, simulation equivalence, properties, $\forall$CTL* equivalence

$\Rightarrow$ Checking simulation
  - basic idea of algorithm

Skeleton for simulation preorder checking

Input: finite transition system *TS* over *AP* with state space S
Output: simulation order $\preceq_{TS}$

$\mathcal{R} := \{ (s_1, s_2) \mid L(s_1) = L(s_2) \}$;

while $\mathcal{R}$ is not a simulation **do**
  let $(s_1, s_2) \in \mathcal{R}$ such that $s_1 \longrightarrow s_1'$ and $\forall s_2'.\ s_2 \longrightarrow s_2'$ implies $(s_1', s_2') \notin \mathcal{R}$;
  $\mathcal{R} := \mathcal{R} \setminus \{ (s_1, s_2) \}$;
od
return $\mathcal{R}$

The number of iterations is bounded above by $|S|^2$, since:

$$S \times S \supseteq \mathcal{R}_0 \supsetneqq \mathcal{R}_1 \supsetneqq \mathcal{R}_2 \supsetneqq \ldots \supsetneqq \mathcal{R}_n = \preceq_{TS}$$

Algorithm to compute $\preceq$ (1)

---

for all $s_1 \in S$ **do**
  $\mathit{Sim}(s_1) := \{ s_2 \in S \mid L(s_1) = L(s_2) \}$; (* initialization *)
od

while $\exists (s_1, s_2) \in S \times \mathit{Sim}(s_1).\ \exists s_1' \in \mathit{Post}(s_1)$ with $\mathit{Post}(s_2) \cap \mathit{Sim}(s_1') = \varnothing$ **do**
  choose such a pair of states (s_1, s_2); (* $s_1 \not\preceq_{TS} s_2$ *)
  $\mathit{Sim}(s_1) := \mathit{Sim}(s_1) \setminus \{ s_2 \}$;
od
(* $\mathit{Sim}(s) = \mathit{Sim}_{TS}(s)$ for any s *)
return $\{ (s_1, s_2) \mid s_2 \in \mathit{Sim}(s_1) \}$

$\mathit{Sim}_{\mathcal{R}}(s) = \{ s' \mid (s, s') \in \mathcal{R} \}$, the upward closure of s under $\mathcal{R}$

$\varnothing \supseteq \mathit{Sim}_{\mathcal{R}_0}(s) \supseteq \mathit{Sim}_{\mathcal{R}_1}(s) \supseteq \ldots \supseteq \mathit{Sim}_{\mathcal{R}_n}(s) = \mathit{Sim}_{\preceq_{TS}}(s)$

Time complexity

Time complexity of computing $\prec_{TS}$ is $\mathcal{O}\left(M \cdot |S|^2\right)$

in each iteration a single pair is deleted; can we do better?

A simple observation

$$\begin{array}{ccc} s_1 & \longrightarrow & s_1' \\ \mathcal{R} & & \mathcal{R} \\ s_2 & \longrightarrow & s_2' \end{array}$$

- Assume: s_2' is the *only* successor of s_2 related to s_1' $(*)$
  - $Sim_{\mathcal{R}}(s_1') \cap Post(s_2) = \{ s_2' \}$ where $Sim_{\mathcal{R}}(s_1') = \{ s \in S \mid (s_1', s) \in \mathcal{R} \}$
- Removing (s_1', s_2') from $\mathcal{R}$ implies that $s_1 \not\preceq s_2$

$\Rightarrow$ (s_1, s_2) can thus also safely be removed from $\mathcal{R}$

- This applies to *all* direct predecessors of s_2' satisfying $(*)$

Algorithm to compute $\preceq$ (2)

Input: finite transition system *TS* over *AP* with state space S
Output: simulation order $\preceq_{TS}$

---

for all $s_1 \in S$ **do**
  $\mathit{Sim}_{old}(s_1) := S$;
  $\mathit{Sim}(s_1) := \{ s_2 \in S \mid L(s_1) = L(s_2) \}$;
od
while $(\exists s \in S \text{ with } \mathit{Sim}_{old}(s) \neq \mathit{Sim}(s))$ **do**
  choose s_1' such that $\mathit{Sim}_{old}(s_1') \neq \mathit{Sim}(s_1')$;
  $\mathit{Remove}(s_1') := \mathit{Pre}\Big(\mathit{Sim}_{old}(s_1')\Big) \setminus \mathit{Pre}\Big(\mathit{Sim}(s_1')\Big)$;   (* predecessors that $\not\preceq s_1'$ *)
  for all $s_1 \in \mathit{Pre}(s_1')$ **do**
    $\mathit{Sim}(s_1) := \mathit{Sim}(s_1) \setminus \mathit{Remove}(s_1')$;
  od
  $\mathit{Sim}_{old}(s_1') := \mathit{Sim}(s_1')$;
od
return $\{ (s_1, s_2) \mid s_2 \in \mathit{Sim}(s_1) \}$

Implementation details

- Introduce for any state s_1' the set *Remove*(s_1')
  - contains all states s_2 to be removed from *Sim*(s_1) for $s_1 \in Pre(s_1')$:

$$Remove(s_1') \;=\; Pre(Sim_{old}(s_1')) \setminus Pre(Sim(s_1'))$$

  $\Rightarrow$ the sets Sim_{old} are superfluous

  $\Rightarrow$ termination condition: $Remove(s_1') = \varnothing$ for all $s_1' \in S$

  - adapt the sets *Remove* on modifying *Sim*(s_1)

- Let $s_2 \in Remove(s_1')$ and $s_1 \in Pre(s_1')$
  - then $s_1 \longrightarrow s_1'$ but no transition $s_2 \longrightarrow s_2'$ with $s_2' \in Sim(s_1')$
  - then $s_1 \not\preceq s_2$, so s_2 can be removed from *Sim*(s_1):

  $\Rightarrow$ extend $Remove(s_1)$ with $s \in Pre(s_2)$ and $Post(s) \cap Sim(s_1) = \varnothing$

Algorithm to compute $\preceq$ (3)

```
for all s1 ∈ S do
  Sim(s1) := { s2 ∈ S | L(s1) = L(s2) };                                  (* initialization *)
  Remove(s1) := S \ Pre(Sim(s1));
od
                     (* loop invariant: Remove(s1') = Pre (Sim_old(s1')) \ Pre (Sim(s1')) *)
while (∃s1' ∈ S with Remove(s1') ≠ ∅) do
  choose s1' such that Remove(s1') ≠ ∅;
  for all s2 ∈ Remove(s1') do
    for all s1 ∈ Pre(s1') do
      if s2 ∈ Sim(s1) then
        Sim(s1) := Sim(s1) \ { s2 };                          (* s2 ∈ Sim_old(s1) \ Sim(s1) *)
        for all s ∈ Pre(s2) with Post(s) ∩ Sim(s1) = ∅ do
                                              (* s ∈ Pre (Sim_old(s1)) \ Pre(Sim(s1)) *)
          Remove(s1) := Remove(s1) ∪ { s };
        od
      fi
    od
  od
  Remove(s1') := ∅;                                           (* Sim_old(s1') := Sim(s1') *)
od
return { (s1, s2) | s2 ∈ Sim(s1) }
```

Time complexity

Time complexity of computing $\preceq_{TS}$ is $\mathcal{O}\Big(M \cdot |S|\Big)$

Summary

formal relation	trace equivalence	bisimulation	simulation
complexity	PSPACE-complete	$\mathcal{O}(M \cdot \log \lvert S \rvert)$	$\mathcal{O}(M \cdot \lvert S \rvert)$
logical fragment	LTL	CTL^*	$\forall\text{CTL}^*$
preservation	strong	strong match	weak match