Model Checking

Simulation and Stutter Trace Equivalence

[Baier & Katoen, Chapter 7.4–7.7]

Joost-Pieter Katoen and Tim Quatmann

Software Modeling and Verification Group

RWTH Aachen, SoSe 2022

Set-up

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

Overview

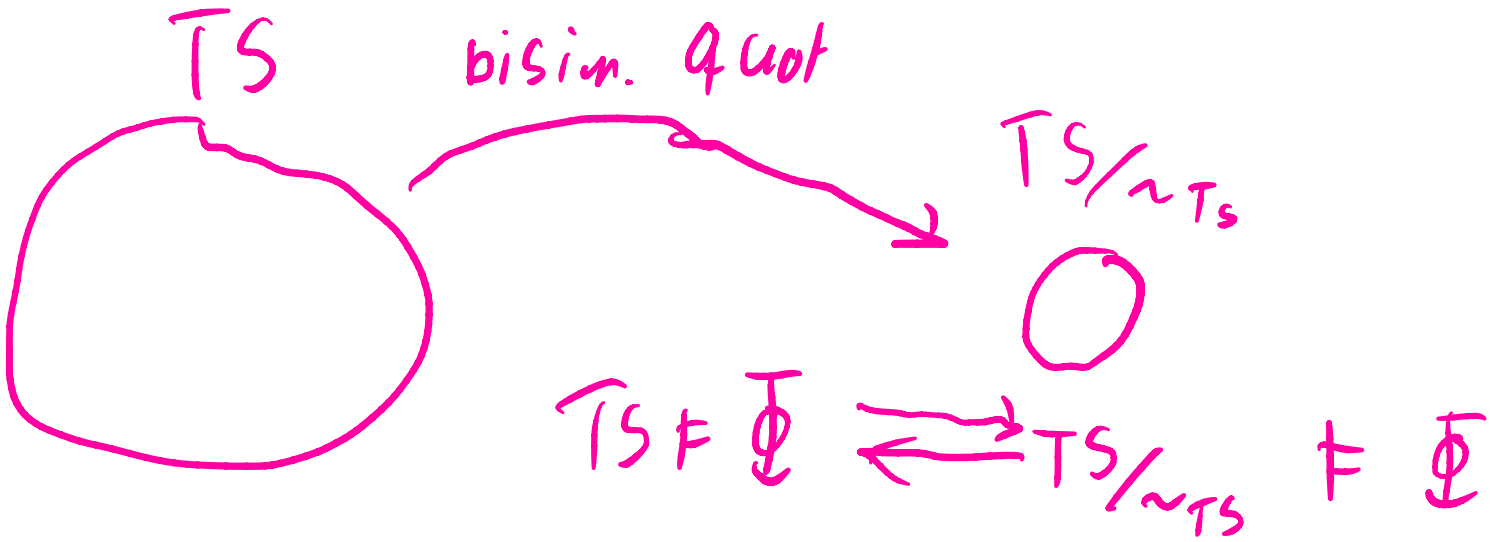
Previous Lecture: Bisimulation Quotienting

The quotient transition system $TS/\sim_{TS}$ for finite TS can be computed in $O(\log |S| \cdot |\rightarrow|)$ time.

partition refinement alg.

For every transition system TS it holds: $TS \sim TS/\sim_{TS}$.

- Hence, $TS \equiv_{\mathrm{CTL}^*} TS/\sim_{TS}$



Limitations of Bisimulation Quotienting

- The partition refinement algorithm does not work for infinite TS
- Even if TS is finite (but huge), computing $TS/{\sim_{TS}}$ might not be feasible
- $TS/{\sim_{TS}}$ can still be too large

Overview

Motivating Example

Program P over $x, y \in \mathbb{N}$

```
1 while x > 0 do
2 |   x−−;
3 |   y++
  if y is even then
4 |   return 1
  else
5 |   return 0
```

- State space S of induced TS_P is infinite
  - $S = \{\langle \ell, x{=}n, y{=}m \rangle \mid n, m \in \mathbb{N}\}$

Motivating Example

Program P over $x, y \in \mathbb{N}$

```
while x > 0 do
|   x−−;
|   y++
if y is even then
|   return 1
else
|   return 0
```

- State space S of induced TS_P is infinite
  - $S = \{\langle \ell, x{=}n, y{=}m \rangle \mid n, m \in \mathbb{N}\}$
- **Approach:** map concrete variable values to an abstract domain
  - $\mathrm{dom}_{\mathrm{abs}}(x) = \{\texttt{gzero}, \texttt{zero}\}$
  - $\mathrm{dom}_{\mathrm{abs}}(y) = \{\texttt{even}, \texttt{odd}\}$
- Consider abstraction function $f : S \to \hat{S}$

$$f(\langle \ell, x{=}n, y{=}m \rangle) = \begin{cases} \langle \ell, x{=}\texttt{gzero}, y{=}\texttt{even} \rangle & \text{if } x > 0 \wedge y \text{ is even} \\ \langle \ell, x{=}\texttt{gzero}, y{=}\texttt{odd} \rangle & \text{if } x > 0 \wedge y \text{ is } \cancel{\text{even}} \text{ odd} \\ \langle \ell, x{=}\texttt{zero}, y{=}\texttt{even} \rangle & \text{if } x = 0 \wedge y \text{ is } \cancel{\text{odd}} \text{ even} \\ \langle \ell, x{=}\texttt{zero}, y{=}\texttt{odd} \rangle & \text{if } x = 0 \wedge y \text{ is odd} \end{cases}$$

Motivating Example

Program P

```
while x > 0 do
    x−−;
    y++
if y is even then
    return 1
else
    return 0
```

Program P_{abstract}

```
while x = gzero do
    x := gzero or x := zero;
    if y = even then y := odd else y := even;
if y = even then
    return 1
else
    return 0
```

nondeterministic choice

$$\underbrace{TS_P}_{\text{infinite}} \mapsto \underbrace{\widehat{TS}_{f,P}}_{\text{finite}}$$

∀CTL*

$TS_P \not\models \Phi \leftarrow \widehat{TS}_{f,P} \not\models \Phi$

$TS \not\models \Phi \not\leftarrow TS_{f,P} \not\models \Phi$

Simulation Relation

Definition: simulation relation

Relation $\mathfrak{R} \subseteq S \times S$ is a simulation relation on TS if for any $(s_1, s_2) \in \mathfrak{R}$:

- $L(s_1) = L(s_2)$, and
- if $s_1' \in Post(s_1)$ then $(s_1', s_2') \in \mathfrak{R}$ for some $s_2' \in Post(s_2)$.

State s_2 simulates s_1, written $s_1 \preceq_{TS} s_2$ if $(s_1, s_2) \in \mathfrak{R}$ for some simulation relation $\mathfrak{R}$ on TS.

$TS_1 \preceq TS_2$ iff $\forall s_1 \in I_1 . \exists s_2 \in I_2 . s_1 \preceq_{TS_1 \oplus TS_2} s_2$.

$\preceq_{TS}$ is a preorder and the coarsest simulation for TS.

transitive, reflexive

Visually

$$\begin{array}{ccc} s_1 & \rightarrow & s_1' \\ \mathfrak{R} & & \\ s_2 & & \end{array} \quad \text{can be completed to} \quad \begin{array}{ccc} s_1 & \rightarrow & s_1' \\ \mathfrak{R} & & \mathfrak{R} \\ s_2 & \rightarrow & s_2' \end{array}$$

Visually

$s_1 \rightarrow s_1'$, s_2, $\mathfrak{R}$ can be completed to $s_1 \rightarrow s_1'$, $s_2 \rightarrow s_2'$, $\mathfrak{R}$, $\mathfrak{R}$

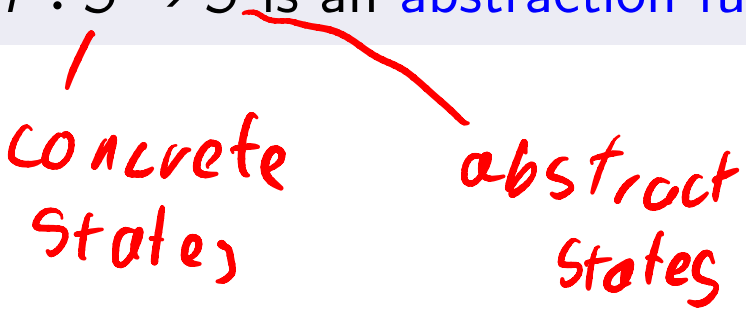
but not necessarily:

s_1, $s_2 \rightarrow s_2'$, $\mathfrak{R}$ can be completed to $s_1 \rightarrow s_1'$, $s_2 \rightarrow s_2'$, $\mathfrak{R}$, $\mathfrak{R}$

Abstraction Function

Definition: abstraction function

$f : S \to \hat{S}$ is an abstraction function if $f(s) = f(s') \Rightarrow L(s) = L(s')$.



Abstraction Function

Definition: abstraction function

$f : S \to \hat{S}$ is an abstraction function if $f(s) = f(s') \Rightarrow L(s) = L(s')$.

S are "concrete" states and $\hat{S}$ are "abstract" states, mostly $|\hat{S}| \ll |S|$

Abstraction Function

Definition: abstraction function

$f : S \to \hat{S}$ is an abstraction function if $f(s) = f(s') \Rightarrow L(s) = L(s')$.

S are "concrete" states and $\hat{S}$ are "abstract" states, mostly $|\hat{S}| \ll |S|$
Abstraction functions are useful for:

- data abstraction: abstract from values of program or control variables

  $x = 23 \mapsto x = \text{gzero}$

  f : concrete data domain $\to$ abstract data domain

- predicate abstraction: use predicates over the program variables

  f : state $\to$ valuations of the predicates

- localization reduction: program variables are visible or invisible

  f : all variables $\to$ visible variables

Abstract Transition System

Definition: abstract transition system

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

$$TS_f = (\hat{S}, Act, \rightarrow_f, I_f, AP, L_f), \quad \text{the abstraction of } TS \text{ under } f$$

where

- $\rightarrow_f$ is defined by: $\dfrac{s \xrightarrow{\alpha} s'}{f(s) \xrightarrow{\alpha}_f f(s')}$
- $I_f = \{ f(s) \mid s \in I \}$ and $L_f(f(s)) = L(s)$.

$TS \trianglelefteq TS_f$

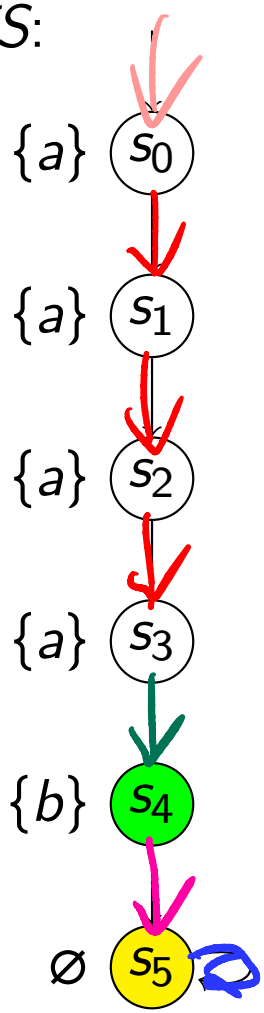
The relation $\mathfrak{R} = \{ (s, f(s)) \mid s \in S \}$ is a simulation for (TS, TS_f).

Proof.

By checking all conditions of a simulation relation. Straightforward. □

Example

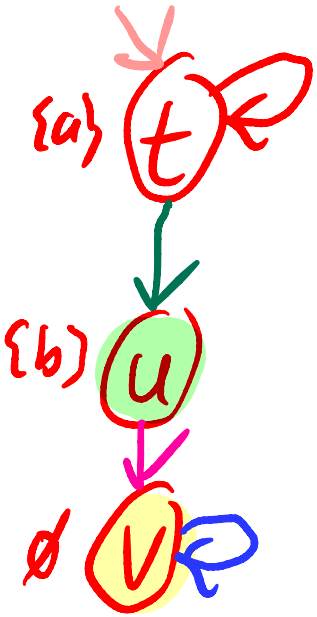
TS:



abstract states

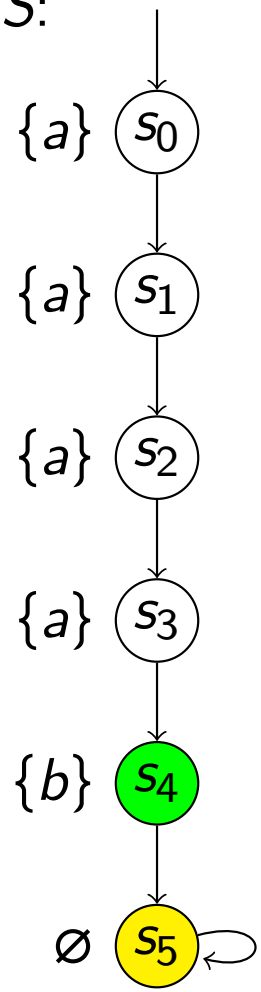
$$f(s_i) = \begin{cases} t & \text{if } i \in \{0,1,2,3\} \\ u & \text{if } i = 4 \\ v & \text{if } i = 5 \end{cases}$$

TS_f:



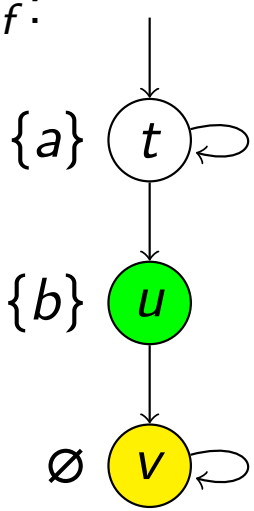
Example

TS:



$$f(s_i) = \begin{cases} t & \text{if } i \in \{0,1,2,3\} \\ u & \text{if } i = 4 \\ v & \text{if } i = 5 \end{cases}$$

TS_f:



Simulation Equivalence

Definition: simulation equivalence

Transition systems TS_1 and TS_2 are simulation equivalent, denoted $TS_1 \simeq TS_2$ if $TS_1 \preceq TS_2$ and $TS_2 \preceq TS_1$.

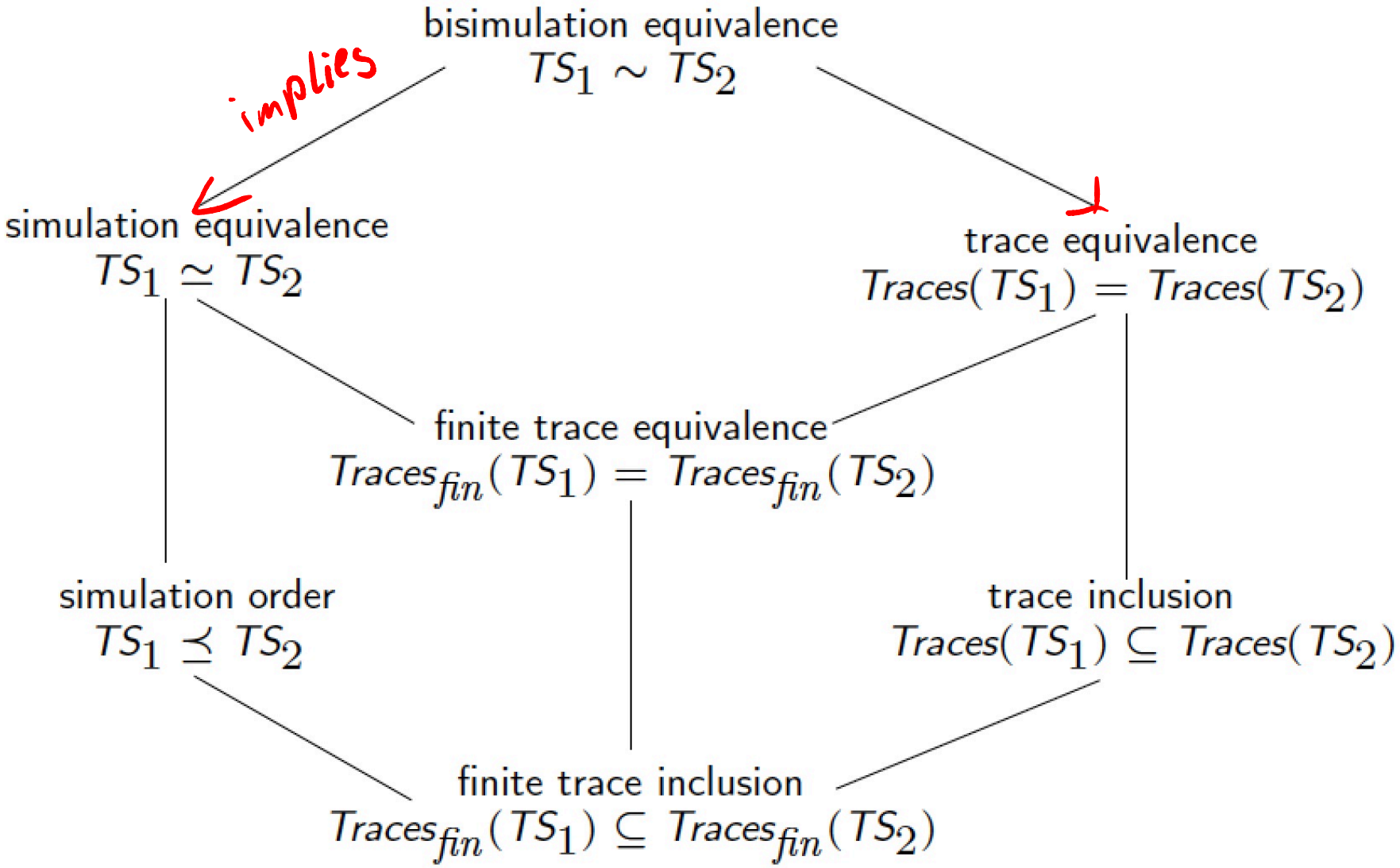
$TS_2 \simeq TS_1$

$TS_1 \sim TS_2 \not\Rightarrow TS_1 \simeq TS_2$

1. Bisimilarity implies simulation equivalence; not the converse.
2. Simulation equivalence implies finite trace equivalence; not the converse.
3. For AP-deterministic[1] transition systems: simulation, bisimulation and trace equivalence coincide.

[1] TS is AP-deterministic if all initial states are labelled differently, and this also applies to all direct successors of any state in TS.

Overview


bisimulation equivalence
$TS_1 \sim TS_2$
implies
simulation equivalence
$TS_1 \simeq TS_2$
trace equivalence
$Traces(TS_1) = Traces(TS_2)$
finite trace equivalence
$Traces_{fin}(TS_1) = Traces_{fin}(TS_2)$
simulation order
$TS_1 \preceq TS_2$
trace inclusion
$Traces(TS_1) \subseteq Traces(TS_2)$
finite trace inclusion
$Traces_{fin}(TS_1) \subseteq Traces_{fin}(TS_2)$

Logical Characterisation for $\preceq_{TS}$

simulation preorder

Negation of formulas is problematic as $\preceq_{TS}$ is not symmetric:

- Let **L** be a fragment of CTL* which is closed under negation

  $\Phi \in L \Rightarrow \neg\Phi \in L$

- And assume **L** weakly matches $\preceq_{TS}$, that is:

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

  $s_1 \not\models \Phi \Rightarrow s_2 \not\models \Phi$

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

  $$s_1 \models \Phi \implies s_1 \not\models \underbrace{\neg\Phi}_{\in L} \implies s_2 \not\models \underbrace{\neg\Phi}_{\in L} \implies s_2 \models \Phi.$$

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

Universal Fragment of CTL*

no $\neg\Phi$ for $\Phi \neq a$

Definition: universal fragment of CTL*

no $\exists$

$\forall$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:

dual dual

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

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

$\forall$CTL does not contain (general) negation and no existential path quantifier

Universal CTL* Contains LTL

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

for LTL

Definition: positive normal form (Lecture 7)

The LTL-formula φ is in positive normal form (PNF) if it is of the form:

$$\varphi ::= \text{true} \mid \text{false} \mid a \mid \neg a \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2 \mid \varphi_1 \,\mathsf{R}\, \varphi_2.$$

As $\Box\varphi \equiv \text{false}\,\mathsf{R}\,\varphi$, $\Box\varphi$ is in PNF; $\Diamond\varphi \equiv \text{true}\,\mathsf{U}\,\varphi$ is in PNF too.

Simulation and CTL

if $s \preceq_{TS} s'$ and $s' \models \Phi$ then $s \models \Phi$

Theorem: Simulation equivalence, CTL and and CTL*

Let TS be a finitely branching[2] transition system and s, s' states in TS. The following statements are equivalent:

1. $s \preceq_{TS} s'$
2. for any $\forall$CTL*-formula Φ: $s' \models \Phi$ implies $s \models \Phi$
3. for any $\forall$CTL-formula Φ: $s' \models \Phi$ implies $s \models \Phi$
4. for any $\forall$CTL$\backslash_{U,\,R}$-formula Φ: $s' \models \Phi$ implies $s \models \Phi$

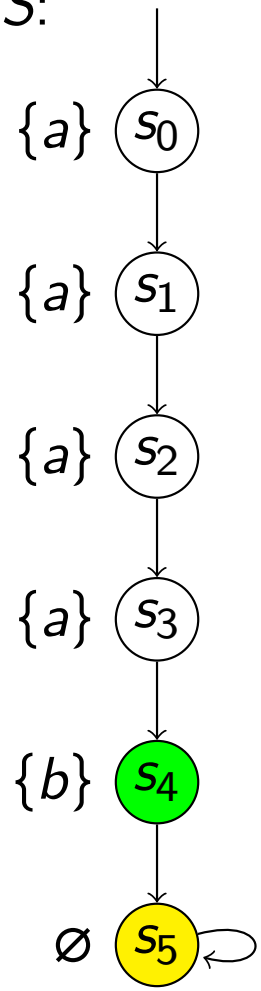
Proof.

Along similar lines as the proof for the corresponding theorem for bisimilarity and CTL*, CTL and CTL$^-$-equivalence. □

[2] This means that every state has only finitely many direct successors.

Example

TS:

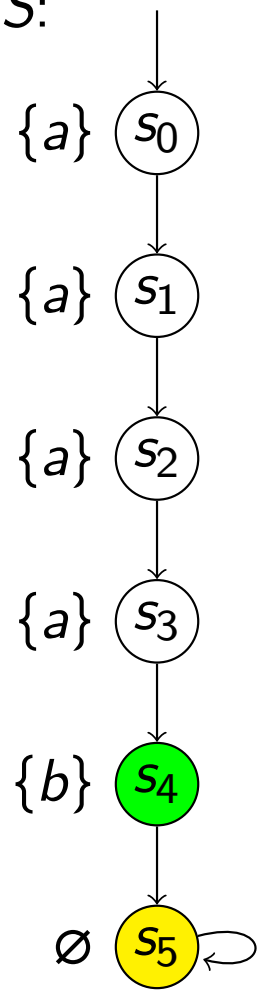


$$f(s_i) = \begin{cases} t & \text{if } i \in \{0,1,2,3\} \\ u & \text{if } i = 4 \\ v & \text{if } i = 5 \end{cases}$$

Example

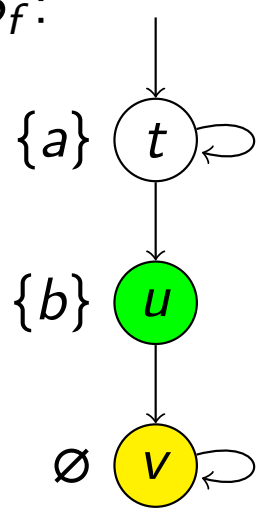
$TS \trianglelefteq TS_f$

TS:



$$f(s_i) = \begin{cases} t & \text{if } i \in \{0,1,2,3\} \\ u & \text{if } i = 4 \\ v & \text{if } i = 5 \end{cases}$$

TS_f:



$TS_f \models \forall b\, R(a \vee b)$

$\Downarrow$

$TS \models \forall b\, R(a \vee b)$

$TS_f \not\models \forall a\, U\, b$

but $TS \models \forall a\, U\, b$

Overview

Algorithm Skeleton

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 $s_1 \to s_1'$ and $\forall s_2'.\ s_2 \to s_2'$ 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:

$$S \times S \supsetneq \mathfrak{R}_0 \supsetneq \mathfrak{R}_1 \supsetneq \mathfrak{R}_2 \supsetneq \ldots \supsetneq \mathfrak{R}_n = \preceq_{TS}$$

Algorithm

---

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

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

(* $Sim(s) = Sim_{TS}(s)$ for any s *)

return $\{ (s_1, s_2) \mid s_2 \in Sim(s_1) \}$

$Sim_{\mathfrak{R}}(s) = \{ s' \mid (s, s') \in \mathfrak{R} \}$, the upward closure of s under $\mathfrak{R}$

$$\varnothing \supseteq Sim_{\mathfrak{R}_0}(s) \supseteq Sim_{\mathfrak{R}_1}(s) \supseteq \ldots \supseteq Sim_{\mathfrak{R}_n}(s) = Sim_{\preceq_{TS}}(s)$$

Time complexity

Number of States

The time complexity of computing $\preceq_{TS}$ is $O(M \cdot N^2)$.

Number of transitions

Proof.

In the worst case, there are N^2 iterations as their are N^2 pairs of states. For each pair of states in the worst case all transitions have to be examined. □

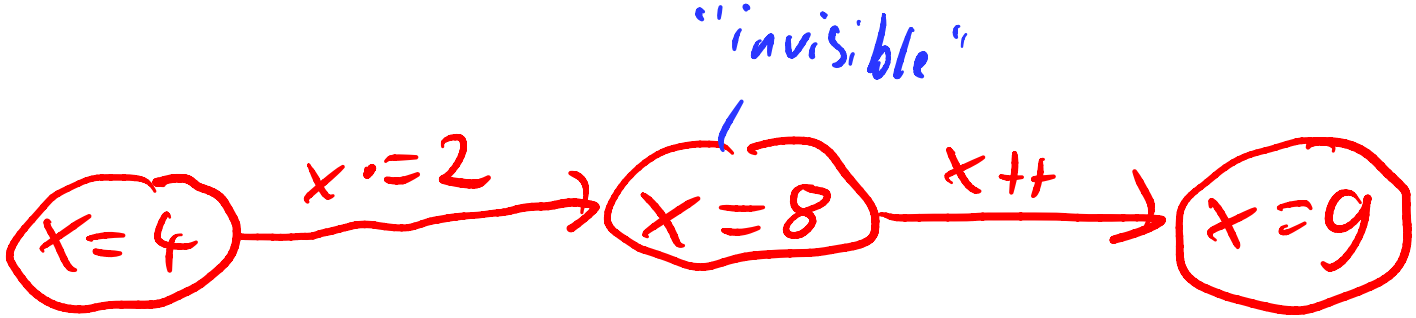
The best known algorithm[3] has complexity $O(M \cdot N)$. It removes several pairs in each iteration at a time and uses efficient data structures for the sets $Sim_{\mathfrak{R}}(s)$.

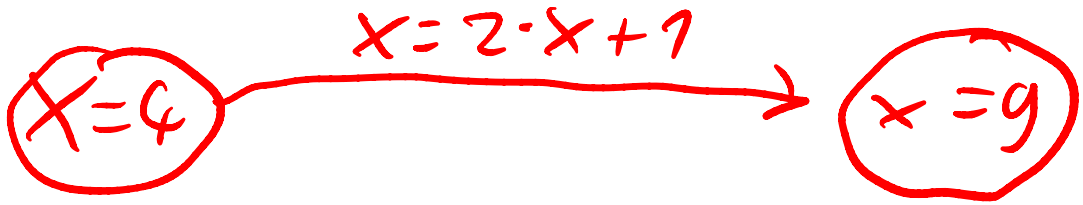
[3] Due to Henzinger, Henzinger and Kopke.

Overview

Motivation

- Bisimulation, simulation and trace equivalence are strong
  - each transition $s \to s'$ must be matched by a transition
  - for comparing models at different abstraction levels, this is too fine
  - e.g., modeling an abstract action by a sequence of concrete actions





Motivation

- Bisimulation, simulation and trace equivalence are strong
  - each transition $s \to s'$ must be matched by a transition
  - for comparing models at different abstraction levels, this is too fine
  - e.g., modeling an abstract action by a sequence of concrete actions

- Idea: allow for sequences of "invisible" transitions
  - each transition $s \to s'$ must be matched by a path fragment of a related state
  - matching means: ending in a state related to s', and all previous states invisible

Motivation

- Bisimulation, simulation and trace equivalence are strong
  - each transition $s \to s'$ must be matched by a transition
  - for comparing models at different abstraction levels, this is too fine
  - e.g., modeling an abstract action by a sequence of concrete actions

- Idea: allow for sequences of "invisible" transitions
  - each transition $s \to s'$ must be matched by a path fragment of a related state
  - matching means: ending in a state related to s', and all previous states invisible

- Abstraction of such internal computations yields coarser quotients
  - but: what kind of properties are preserved?
  - but: can such quotients still be obtained efficiently?
  - but: how to treat infinite internal computations?

Stutter Equivalence

Definition: stutter step

Transition $s \to s'$ in transition system TS is a stutter step if $L(s) = L(s')$.

Stutter Equivalence

Definition: stutter step

Transition $s \to s'$ in transition system TS is a stutter step if $L(s) = L(s')$.

Definition: stutter equivalence

Paths π_1 and π_2 are stutter equivalent, denoted $\pi_1 \equiv_{sttrace} \pi_2$ whenever

$$trace(\pi_1) \text{ and } trace(\pi_2) \text{ are both of the form } {A_0}^+ {A_1}^+ {A_2}^+ \ldots$$

for $A_i \subseteq AP$.

Stutter Equivalence

Definition: stutter step

Transition $s \to s'$ in transition system TS is a stutter step if $L(s) = L(s')$.

Definition: stutter equivalence

Paths π_1 and π_2 are stutter equivalent, denoted $\pi_1 \equiv_{sttrace} \pi_2$ whenever

$$trace(\pi_1) \text{ and } trace(\pi_2) \text{ are both of the form } {A_0}^+ {A_1}^+ {A_2}^+ \ldots$$

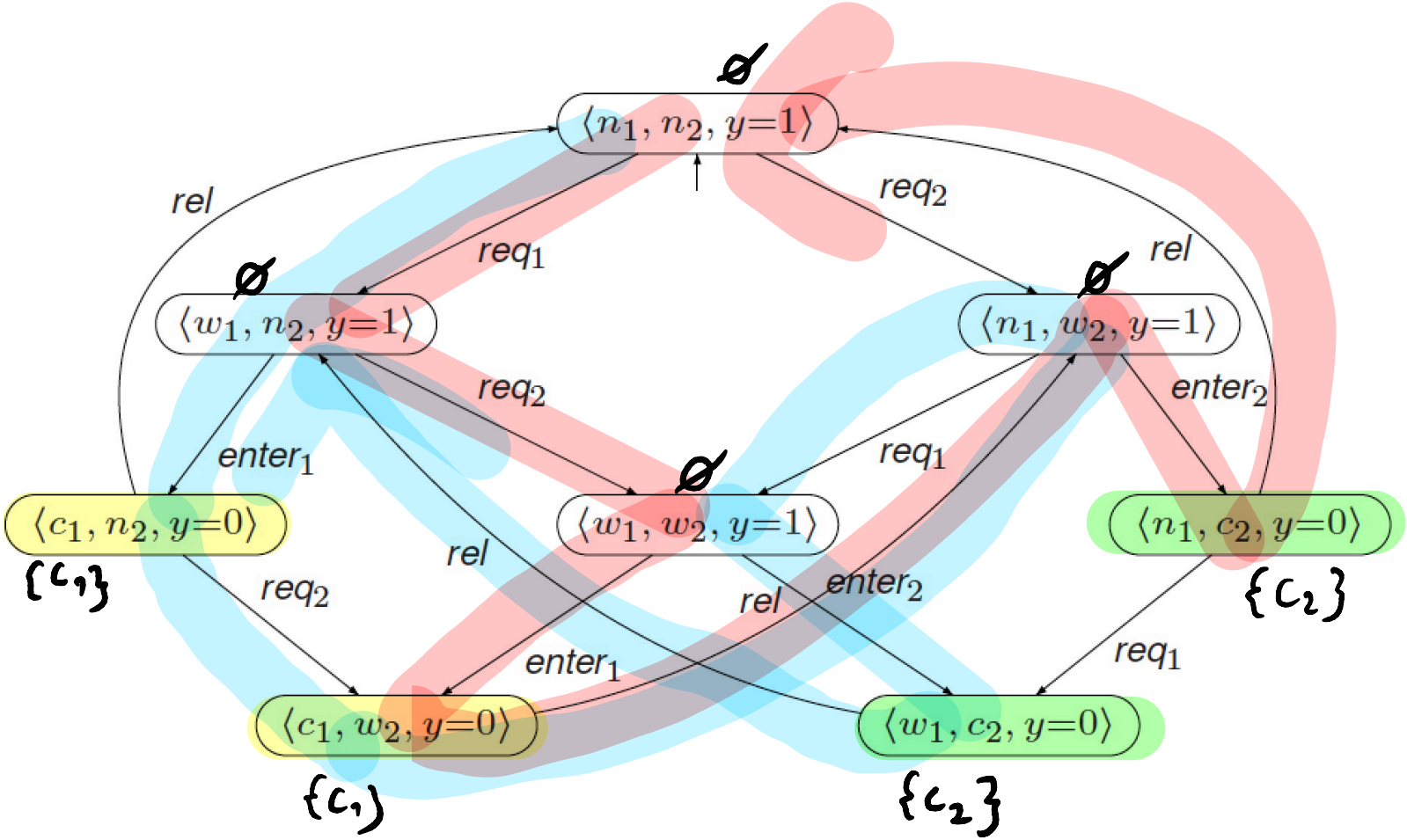
for $A_i \subseteq AP$.

For positive integers n_i and m_i:

$$trace(\pi_1) = \underbrace{A_0 \ldots A_0}_{n_0 \text{ times}} \underbrace{A_1 \ldots A_1}_{n_1 \text{ times}} \underbrace{A_2 \ldots A_2}_{n_2 \text{ times}} \ldots$$

$$trace(\pi_2) = \underbrace{A_0 \ldots A_0}_{m_0 \text{ times}} \underbrace{A_1 \ldots A_1}_{m_1 \text{ times}} \underbrace{A_2 \ldots A_2}_{m_2 \text{ times}} \ldots$$

Example


$\langle n_1, n_2, y{=}1 \rangle$
$\langle w_1, n_2, y{=}1 \rangle$
$\langle n_1, w_2, y{=}1 \rangle$
$\langle c_1, n_2, y{=}0 \rangle$
$\langle w_1, w_2, y{=}1 \rangle$
$\langle n_1, c_2, y{=}0 \rangle$
$\langle c_1, w_2, y{=}0 \rangle$
$\langle w_1, c_2, y{=}0 \rangle$
rel
req_1
req_2
$enter_1$
$enter_2$
$\{c_1\}$
$\{c_2\}$

Example

These infinite paths are stutter equivalent

$$\begin{aligned}
\pi_1 \quad = \quad & \langle n_1, n_2 \rangle \to \langle w_1, n_2 \rangle \to \langle w_1, w_2 \rangle \to \langle c_1, w_2 \rangle \to \langle n_1, w_2 \rangle \to \\
& \langle n_1, c_2 \rangle \to \langle n_1, n_2 \rangle \to \langle w_1, n_2 \rangle \to \langle w_1, w_2 \rangle \to \langle c_1, w_2 \rangle \to \ldots \\
\pi_2 \quad = \quad & \langle n_1, n_2 \rangle \to \langle w_1, n_2 \rangle \to \langle c_1, n_2 \rangle \to \langle c_1, w_2 \rangle \to \langle n_1, w_2 \rangle \to \\
& \langle w_1, w_2 \rangle \to \langle w_1, c_2 \rangle \to \langle w_1, n_2 \rangle \to \langle c_1, n_2 \rangle \to \ldots
\end{aligned}$$

Hence, $\pi_1 \equiv_{sttrace} \pi_2$, since for e.g., $AP = \{ crit_1, crit_2 \}$:

$$\begin{aligned}
trace(\pi_1) \quad = \quad & \varnothing^3 \{ crit_1 \} \varnothing \{ crit_2 \} \varnothing^3 \{ crit_1 \} \ldots \quad \text{and} \\
trace(\pi_2) \quad = \quad & \varnothing^2 (\{ crit_1 \})^2 \varnothing^2 \{ crit_2 \} \varnothing \{ crit_1 \} \ldots
\end{aligned}$$

Stutter Trace Equivalence

Definition: stutter trace equivalence

Transition systems TS_i over AP, i=1, 2, are stutter-trace equivalent:

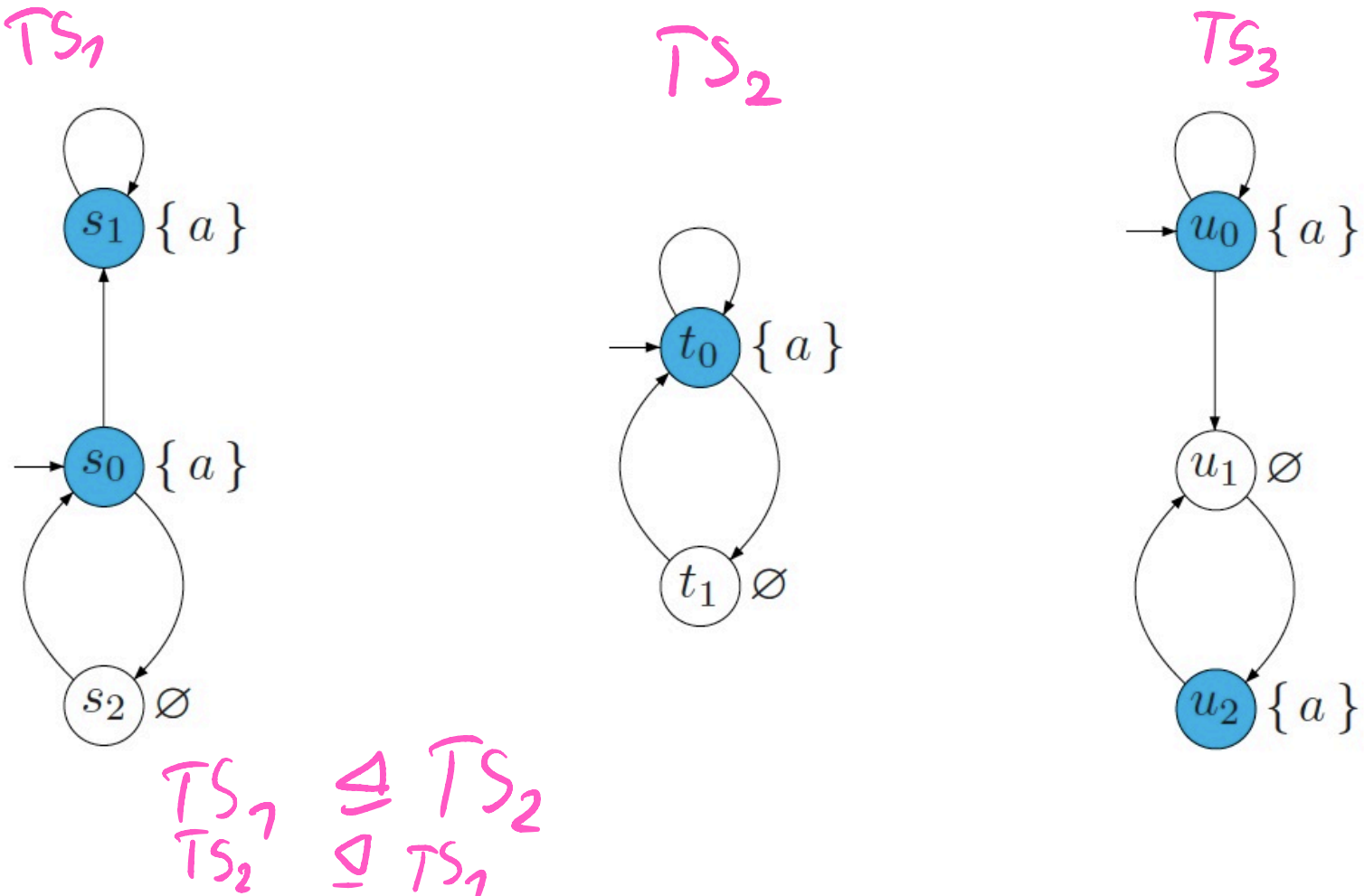
$$TS_1 \equiv_{sttrace} TS_2 \quad \text{if and only if} \quad TS_1 \trianglelefteq TS_2 \text{ and } TS_2 \trianglelefteq TS_1$$

where the stutter trace inclusion relation $\trianglelefteq$ is defined by:

$$TS_1 \trianglelefteq TS_2 \quad \text{iff} \quad \forall \sigma_1 \in \mathit{Traces}(TS_1) \left(\exists \sigma_2 \in \mathit{Traces}(TS_2). \ \sigma_1 \equiv_{sttrace} \sigma_2 \right)$$

Trace-equivalent transition systems are stutter trace-equivalent, but not the converse.

Example



$TS_1 \equiv_{sttrace} TS_2$, $\neg(TS_1 \trianglelefteq TS_3)$ and $\neg(TS_2 \trianglelefteq TS_3)$, but $TS_3 \trianglelefteq TS_2$ and $TS_3 \trianglelefteq TS_1$

The Next-Operator

Stuttering equivalence does not preserve the validity of next-formulas:

$\sigma_1 = A\,B\,B\,B\ldots$ and $\sigma_2 = A\,A\,A\,B\,B\,B\,B\ldots$ for $A, B \subseteq AP$ and $A \neq B$

$b \in B \setminus A$

$\sigma_1 \models \bigcirc b$ but $\sigma_1 \not\models \bigcirc b$

iff $b \in B$ as $b \notin A$

The Next-Operator

Stuttering equivalence does not preserve the validity of next-formulas:

$\sigma_1 = A\,B\,B\,B\ldots$ and $\sigma_2 = A\,A\,A\,B\,B\,B\,B\ldots$ for $A, B \subseteq AP$ and $A \neq B$

Then:

$$\sigma_1 \equiv_{sttrace} \sigma_2 \quad \text{but} \quad \sigma_1 \models \bigcirc b \quad \text{and} \quad \sigma_2 \not\models \bigcirc b \text{ for } b \in B \setminus A.$$

The Next-Operator

Stuttering equivalence does not preserve the validity of next-formulas:

$$\sigma_1 = A\,B\,B\,B\ldots \text{ and } \sigma_2 = A\,A\,A\,B\,B\,B\,B\ldots \text{ for } A, B \subseteq AP \text{ and } A \neq B$$

Then:

$$\sigma_1 \equiv_{sttrace} \sigma_2 \quad \text{but} \quad \sigma_1 \models \bigcirc b \quad \text{and} \quad \sigma_2 \not\models \bigcirc b \text{ for } b \in B \setminus A.$$

$\Rightarrow$ a logical characterisation of $\equiv_{sttrace}$ can only be obtained by omitting $\bigcirc$

in fact, this is the only modal operator that is not preserved by $\equiv_{sttrace}$

Stutter Trace and LTL Equivalence

LTC without the O operator

$\sigma_1 \equiv_{sttrace} \sigma_2$ (both over 2^{AP}) if and only if they are $\text{LTL}_{\setminus\bigcirc}$-equivalent, i.e., $\sigma_1 \vDash \varphi$ if and only if $\sigma_2 \vDash \varphi$ for every $\text{LTL}_{\setminus\bigcirc}$-formula φ.

Proof.

On the black board. We prove the direction from left to right. □

Corollary

$TS_1 \equiv_{sttrace} TS_2$ if and only if $\left(TS_1 \equiv_{LTL\setminus\bigcirc} TS_2\right)$.

Show $\forall \sigma_1, \sigma_2 \in (2^{AP})^\omega$.

$\sigma_1 \equiv_{stutrace} \sigma_2 \Rightarrow (\forall \varphi \in LTL_{\setminus O} . \ \sigma_1 \models \varphi \text{ iff } \sigma_2 \models \varphi)$

Proof: by structural induction over φ

Let $A_0 A_1 A_2 \dots$ be s.t.

$\sigma_1 = A_0^{n_0} A_1^{n_1} A_2^{n_2} \dots$ for $n_i > 0$

$\sigma_2 = A_0^{m_0} A_1^{m_1} A_2^{m_2} \dots$ for $m_i > 0$

Ind. Base

- if $\varphi = a$ for $a \in AP$: $\sigma_1 \models a$ iff $a \in A_0$ iff $\sigma_2 \models a$
- if $\varphi = true$: clear ✓

Ind. step

- if $\varphi = \neg\varphi'$ ...
- if $\varphi = \varphi_1 \wedge \varphi_2$ ...
- if $\varphi = \varphi_1 \, U \, \varphi_2$

Assume $\sigma_1 \models \varphi$, thus

$\exists j \geq 0 \quad \sigma_1[j \dots] \models \varphi_2 \wedge \sigma_1[i \dots] \models \varphi_1$ for all $i < j$

$\sigma_1[j \dots]$ is of the form $A_r^+ A_{r+1}^+ A_{r+2}^+ \dots$ for $r \geq 0$ such that $\sum_{i=0}^{r-1} n_i < j < \sum_{i=0}^{r} n_i$

Since $\sigma_1 \equiv_{stutrace} \sigma_2$ it follows that for $k = 1 + \sum_{i=0}^{r-1} m_i$

$\sigma_2[k \dots]$ is of the form $A_r^+ A_{r+1}^+ A_{r+2}^+ \dots$

i.e. $\sigma_1[j \dots] \equiv_{stutrace} \sigma_2[k \dots]$

$\sigma_1[j\cdots] \equiv_{\text{sttrace}} \sigma_2[k\cdots]$

I.H. and $\sigma_1[j\cdots] \not\models \varphi_2$ yield $\sigma_2[k\cdots] \not\models \varphi_2$

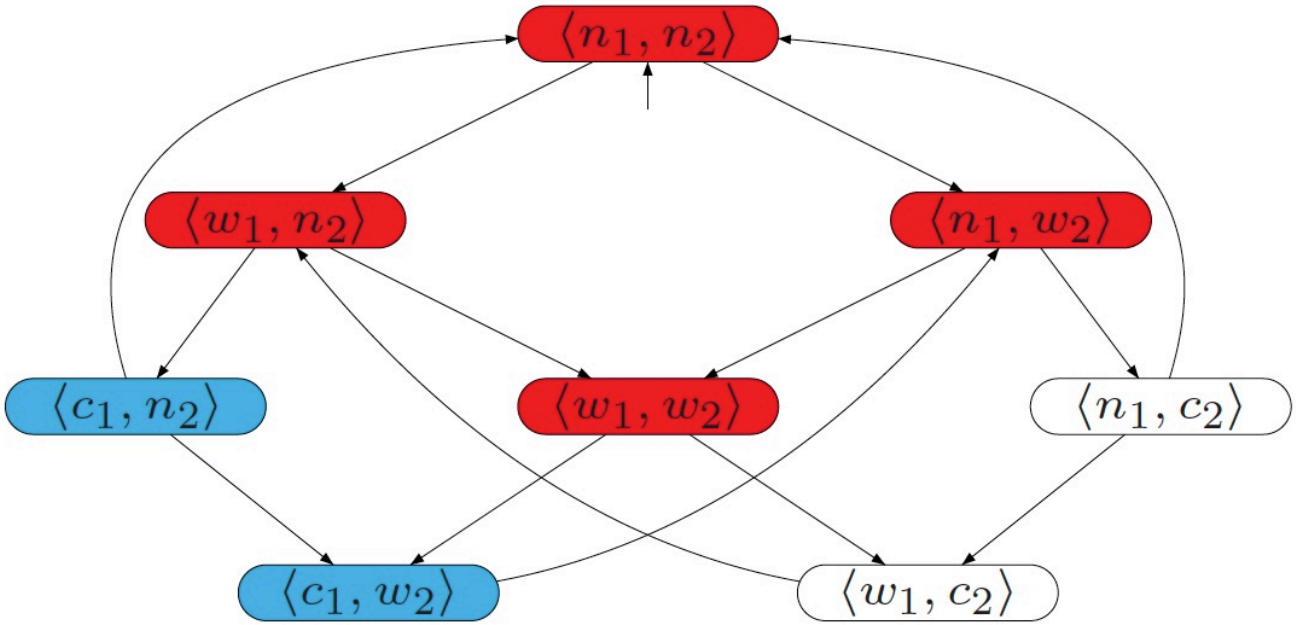
Similarly we can derive $\sigma_2[h\cdots] \not\models \varphi_1$

for all $h < k \Rightarrow \sigma_2 \not\models \underbrace{\varphi_1 \cup \varphi_2}_{=\varphi}$

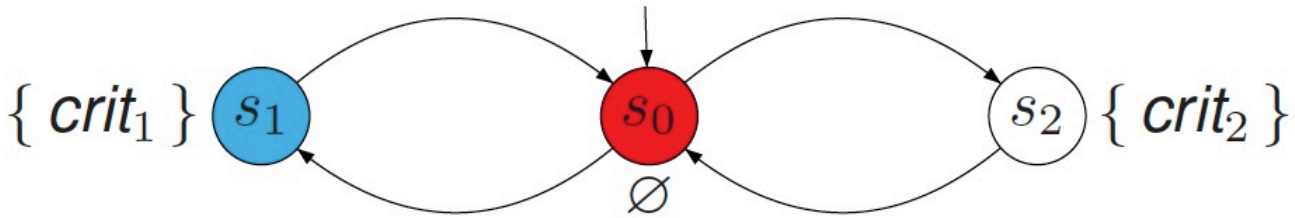
- $\sigma_2 \models \varphi$ implies $\sigma_1 \models \varphi$ holds similarly

□

Example



is stutter equivalent to:



Next Lecture

Monday June 27, 10:30