Verification

Lecture 16

Bernd Finkbeiner
Peter Faymonville
Michael Gerke

UNIVERSITÄT
DES
SAARLANDES

REVIEW: Bisimulation on states

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

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

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

$q_1 \sim_{TS} q_2$ if and only if $TS_{q_1} \sim TS_{q_2}$

Bisimulation vs. CTL* and CTL equivalence

Let TS be a finite transition system and s, s' states in TS

The following statements are equivalent:

(1) $s \sim_{TS} s'$

(2) s and s' are CTL-equivalent, i.e., $s \equiv_{CTL} s'$

(3) s and s' are CTL*-equivalent, i.e., $s \equiv_{CTL^*} s'$

this is proven in three steps: $\equiv_{CTL} \subseteq \sim \subseteq \equiv_{CTL^*} \subseteq \equiv_{CTL}$

important: equivalence is also obtained for any sub-logic containing $\neg$, $\wedge$ and X

REVIEW: An algorithm for bisimulation quotienting

Input: Transition system $(S, Act, \rightarrow, I, AP, L)$
Output: Bisimulation quotient

1. $\Pi = S/{\sim_{AP}}$ $\qquad (q, q') \in {\sim_{AP}} \Leftrightarrow L(q) = L(q')$
2. while some block $B \in \Pi$ is a splitter of Π $\qquad$ loop invariant: Π is coarser than $S/{\sim_{TS}}$
   2.1 pick a block B that is a splitter of Π
   2.2 $\Pi = \text{Refine}(\Pi, B)$
3. return Π

REVIEW: Simulation order on states

A simulation for $TS = (S, Act, \rightarrow, I, AP, L)$ is a binary relation $\mathcal{R} \subseteq S \times S$ such that for all $(q_1, q_2) \in \mathcal{R}$:

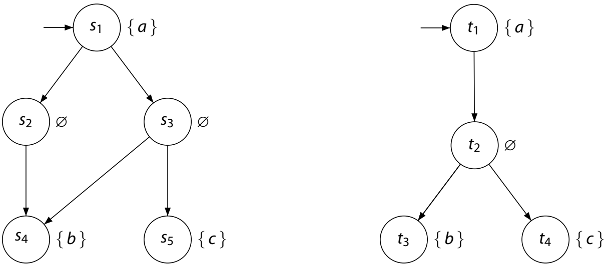
1. $L(q_1) = L(q_2)$
2. if $q_1' \in Post(q_1)$
   then there exists an $q_2' \in Post(q_2)$ with $(q_1', q_2') \in \mathcal{R}$

q_1 is simulated by q_2, denoted by $q_1 \preceq_{TS} q_2$,
if there exists a simulation $\mathcal{R}$ for TS with $(q_1, q_2) \in \mathcal{R}$

$$q_1 \preceq_{TS} q_2 \quad \text{if and only if} \quad TS_{q_1} \preceq TS_{q_2}$$

$$q_1 \simeq_{TS} q_2 \quad \text{if and only if} \quad q_1 \preceq_{TS} q_2 \text{ and } q_2 \preceq_{TS} q_1$$

Similar but not bisimilar



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

REVIEW: $\simeq$, $\forall$CTL*, and $\exists$CTL* equivalence

For finite transition system *TS* without terminal states:

$$\simeq_{TS} \;=\; \equiv_{\forall \mathrm{CTL}^*} \;=\; \equiv_{\forall \mathrm{CTL}} \;=\; \equiv_{\exists \mathrm{CTL}^*} \;=\; \equiv_{\exists \mathrm{CTL}}$$

REVIEW: Skeleton for simulation preorder checking

Require: finite transition system $TS = (S, Act, \rightarrow, I, AP, L)$ over AP
Ensure: simulation order $\preceq_{TS}$

$\mathcal{R} := \{ (q_1, q_2) \mid L(q_1) = L(q_2) \}$;

while $\mathcal{R}$ is not a simulation **do**
  choose $(q_1, q_2) \in \mathcal{R}$
    such that $(q_1, q_1') \in E$, but for all q_2' with $(q_2, q_2') \in E$, $(q_1', q_2') \notin \mathcal{R}$;
  $\mathcal{R} := \mathcal{R} \setminus \{ (q_1, q_2) \}$
end while
return $\mathcal{R}$

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

$$Q \times Q \supseteq \mathcal{R}_0 \supsetneqq \mathcal{R}_1 \supsetneqq \mathcal{R}_2 \supsetneqq \ldots \supsetneqq \mathcal{R}_n = \preceq$$

Checking trace equivalence

Let TS_1 and TS_2 be finite transition systems over AP. Then:

1. The problem whether

$$Traces_{fin}(TS_1) = Traces_{fin}(TS_2) \quad \text{is PSPACE-complete}$$

2. The problem whether

$$Traces(TS_1) = Traces(TS_2) \quad \text{is PSPACE-complete}$$

Overview implementation relations

	bisimulation equivalence	simulation order	trace equivalence
preservation of temporal-logical properties	CTL^* CTL	$\forall CTL^* / \exists CTL^*$ $\forall CTL / \exists CTL$	LTL
checking equivalence	PTIME	PTIME	PSPACE-complete
graph minimization	PTIME	PTIME	---

Motivation: Stutter Equivalence

- Bisimulation, simulation and trace equivalence are <u>strong</u>
  - each transition $s \to s'$ must be matched by a transition of a related state
  - for comparing models at different abstraction levels, this is too fine
  - consider e.g., modeling an abstract action by a sequence of concrete actions
- Idea: allow for sequences of "invisible" actions
  - 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?

Stuttering equivalence

- $s \to s'$ in transition system TS is a stutter step if $L(s) = L(s')$
  - stutter steps do not affect the state labels of successor states
- Paths π_1 and π_2 are stuttering equivalent, denoted $\pi_1 \cong \pi_2$:
  - if there exists an infinite sequence $A_0 A_1 A_2 \ldots$ with $A_i \subseteq AP$ and
  - natural numbers $n_0, n_1, n_2, \ldots, m_0, m_1, m_2, \ldots \geq 1$ such that:

$$
\begin{aligned}
trace(\pi_1) &= \underbrace{A_0 \ldots A_0}_{n_0\text{-times}} \underbrace{A_1 \ldots A_1}_{n_1\text{-times}} \underbrace{A_2 \ldots A_2}_{n_2\text{-times}} \ldots \\
trace(\pi_2) &= \underbrace{A_0, \ldots, A_0}_{m_0\text{-times}} \underbrace{A_1 \ldots A_1}_{m_1\text{-times}} \underbrace{A_2 \ldots A_2}_{m_2\text{-times}} \ldots
\end{aligned}
$$

$\pi_1 \cong \pi_2$ if their traces only differ in their stutter steps

i.e., if both their traces are of the form $A_0^+ A_1^+ A_2^+ \ldots$ for $A_i \subseteq AP$

Stutter trace equivalence

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

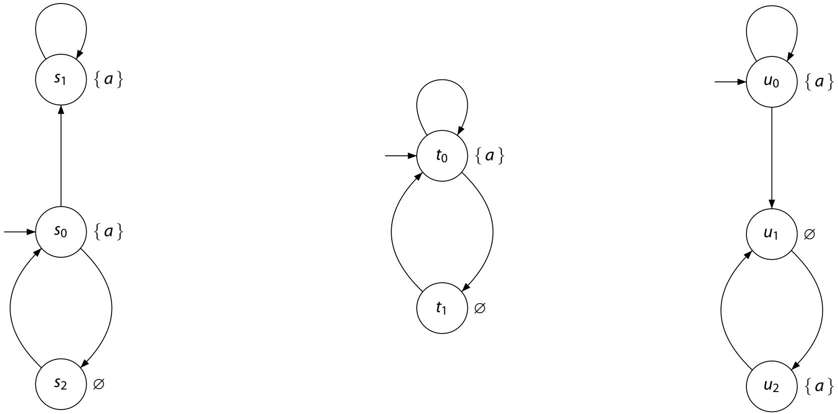
$$TS_1 \cong TS_2 \quad \text{if and only if} \quad TS_1 \sqsubseteq TS_2 \text{ and } TS_2 \sqsubseteq TS_1$$

where $\sqsubseteq$ is defined by:

$$TS_1 \sqsubseteq TS_2 \quad \text{iff} \quad \forall \sigma_1 \in Traces(TS_1)\ (\exists \sigma_2 \in Traces(TS_2).\ \sigma_1 \cong \sigma_2\ )$$

clearly: $Traces(TS_1) = Traces(TS_2)$ implies $TS_1 \cong TS_2$, but not always the reverse

Example


s_1
$\{a\}$
s_0
$\{a\}$
s_2
$\varnothing$
t_0
$\{a\}$
t_1
$\varnothing$
u_0
$\{a\}$
u_1
$\varnothing$
u_2
$\{a\}$

The X 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 for $b \in B \setminus A$:

$$\sigma_1 \cong \sigma_2 \quad \text{but} \quad \sigma_1 \models \mathrm{X}\,b \quad \text{and} \quad \sigma_2 \not\models \mathrm{X}\,b.$$

⇒ a logical characterization of $\cong$ can only be obtained by omitting X

in fact, it turns out that this is the only modal operator that is not preserved by $\cong$!

Stutter trace and $\text{LTL}_{\setminus X}$ equivalence

For traces σ_1 and σ_2 over 2^{AP} it holds:

$$\sigma_1 \cong \sigma_2 \;\Rightarrow\; (\sigma_1 \models \varphi \text{ if and only if } \sigma_2 \models \varphi)$$

for any $\text{LTL}_{\setminus X}$ formula φ over AP

$\text{LTL}_{\setminus X}$ denotes the class of LTL formulas without the next step operator X

Stutter trace and $LTL_{\setminus X}$ equivalence

For transition systems TS_1, TS_2 over AP (without terminal states):

(a) $TS_1 \cong TS_2$ implies $TS_1 \equiv_{LTL_{\setminus X}} TS_2$

(b) if $TS_1 \sqsubseteq TS_2$ then for any $LTL_{\setminus X}$ formula φ: $TS_2 \models \varphi$ implies $TS_1 \models \varphi$

Stutter insensitivity

- LT property P is stutter-insensitive if $[\sigma]_{\cong} \subseteq P$, for any $\sigma \in P$
  - P is stutter insensitive if it is closed under stutter equivalence
- For any stutter-insensitive LT property P:

$$TS_1 \cong TS_2 \quad \text{implies} \quad TS_1 \models P \text{ iff } TS_2 \models P$$

- Moreover: $TS_1 \sqsubseteq TS_2$ and $TS_2 \models P$ implies $TS_1 \models P$
- For any $LTL_{\setminus X}$ formula φ, LT property $Words(\varphi)$ is stutter insensitive
  - but: some stutter insensitive LT properties cannot be expressed in $LTL_{\setminus X}$
  - for LTL formula φ with $Words(\varphi)$ stutter insensitive:

$$\text{there exists } \psi \in LTL_{\setminus X} \text{ such that } \psi \equiv_{LTL} \varphi$$

Stutter bisimulation

$$\begin{array}{ccc} s_1 & \approx & s_2 \\ \downarrow & & \\ s_1' & & \end{array}$$

(with $s_1 \not\approx s_1'$)

can be completed to

$$\begin{array}{ccc} s_1 & \approx & s_2 \\ & & \downarrow \\ s_1 & \approx & u_1 \\ & & \downarrow \\ s_1 & \approx & u_2 \\ & & \downarrow \\ & & \vdots \\ & & \downarrow \\ s_1 & \approx & u_n \\ \downarrow & & \downarrow \\ s_1' & \approx & s_2' \end{array}$$

Stutter bisimulation

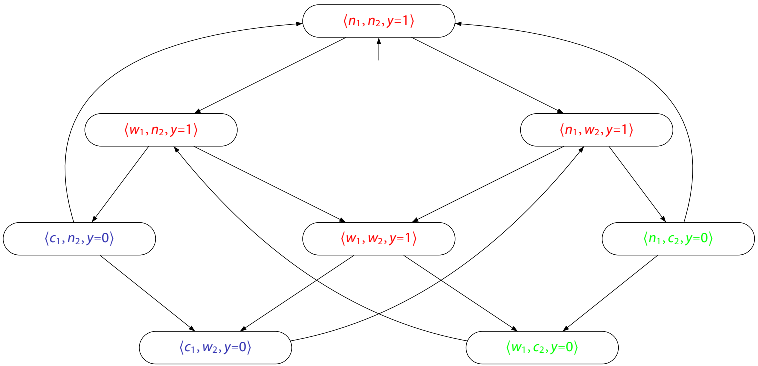
Let $TS = (S, Act, \rightarrow, I, AP, L)$ be a transition system and $\mathcal{R} \subseteq S \times S$

$\mathcal{R}$ is a stutter-bisimulation for TS if for all $(s_1, s_2) \in \mathcal{R}$:

1. $L(s_1) = L(s_2)$
2. if $s_1' \in Post(s_1)$ with $(s_1, s_1') \notin \mathcal{R}$, then there exists a finite path fragment $s_2\, u_1\, \ldots\, u_n\, s_2'$ with $n \geq 0$ and $(s_2, u_i) \in \mathcal{R}$ and $(s_1', s_2') \in \mathcal{R}$
3. if $s_2' \in Post(s_2)$ with $(s_2, s_2') \notin \mathcal{R}$, then there exists a finite path fragment $s_1\, v_1\, \ldots\, v_n\, s_1'$ with $n \geq 0$ and $(s_1, v_i) \in \mathcal{R}$ and $(s_1', s_2') \in \mathcal{R}$

s_1, s_2 are stutter-bisimulation equivalent, denoted $s_1 \approx_{TS} s_2$, if there exists a stutter bisimulation $\mathcal{R}$ for TS with $(s_1, s_2) \in \mathcal{R}$

Example



$\mathcal{R}$ inducing the following partitioning of the state space is a stutter bisimulation:

$$\{\{\langle n_1, n_2\rangle, \langle n_1, w_2\rangle, \langle w_1, n_2\rangle, \langle w_1, w_2\rangle\}, \{\langle c_1, n_2\rangle, \langle c_1, w_2\rangle\}, \{\langle n_1, c_2\rangle, \langle w_1, c_2\rangle\}\}$$

In fact, this is the coarsest stutter bisimulation, i.e., $\mathcal{R}$ equals $\approx_{TS}$

Stutter-bisimilar transition systems

Let $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP, L_i)$, $i = 1, 2$, be transition systems over AP
A stutter bisimulation for (TS_1, TS_2) is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ such that:

1. $\mathcal{R}$ and $\mathcal{R}^{-1}$ are stutter-bisimulations for $TS_1 \oplus TS_2$, and
2. $\forall s_1 \in I_1. (\exists s_2 \in I_2. (s_1, s_2) \in \mathcal{R})$ and
   $\forall s_2 \in I_2. (\exists s_1 \in I_1. (s_1, s_2) \in \mathcal{R})$.

TS_1 and TS_2 are stutter-bisimulation equivalent (stutter-bisimilar, for short), denoted $TS_1 \approx TS_2$, if there exists a stutter bisimulation for (TS_1, TS_2)

Stutter bisimulation quotient

For $TS = (S, Act, \rightarrow, I, AP, L)$ and stutter bisimulation $\approx_{TS} \subseteq S \times S$ let

$TS/\approx^{div} = (S', \{\tau\}, \rightarrow', I', AP, L')$, be the quotient of TS under $\approx_S$

where

- $S' = S/\approx_S = \{ [q]_{\approx_S} \mid q \in S \}$ with $[q]_{\approx_S} = \{ q' \in S \mid q \approx_S q' \}$
- $I' = \{ [q]_{\approx_S} \mid q \in I \}$
- $\rightarrow'$ is defined by: $$\frac{s \xrightarrow{\alpha} s' \text{ and } s \not\approx s'}{[s]_\approx \xrightarrow{\tau}{}' [s']_\approx}$$
- $L'([q]_{\approx_S}) = L(q)$

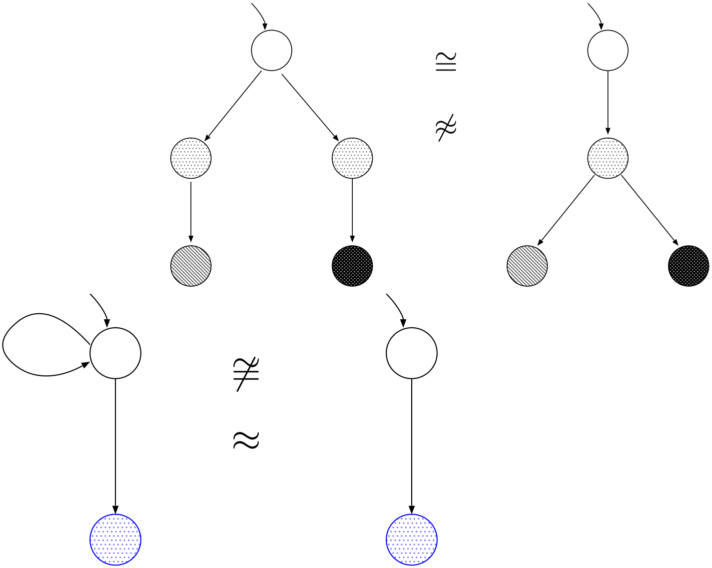
note that (a) no self-loops occur in $TS/\approx_S$ and (b) $TS \approx_S TS/\approx_S$

Stutter trace and stutter bisimulation

For transition systems TS_1 and TS_2 over AP:

- Known fact: $TS_1 \sim TS_2$ implies $Traces(TS_1) = Traces(TS_2)$
- But not: $TS_1 \approx TS_2$ implies $TS_1 \cong TS_2$!
- So:
  - bisimilar transition systems are trace equivalent
  - but stutter-bisimilar transition systems are not always stutter trace-equivalent!
- Why? Stutter paths!
  - stutter bisimulation does not impose any constraint on such paths
  - but $\cong$ requires the existence of a stuttering equivalent trace

Stutter trace and stutter bisimulation are incomparable



Stutter bisimulation does not preserve LTL$_{\setminus X}$

$TS_{left} \approx TS_{right}$ but $TS_{left} \not\models \text{F}\,a$ and $TS_{right} \models \text{F}\,a$

stutter-trace inclusion:

$TS_1 \sqsubseteq TS_2$ iff $\forall \sigma_1 \in Traces(TS_1)\ \exists \sigma_2 \in Traces(TS_2).\ \sigma_1 \cong \sigma_2$

stutter-trace equivalence:

$TS_1 \cong TS_2$ iff $TS_1 \sqsubseteq TS_2$ and $TS_2 \sqsubseteq TS_1$

stutter-bisimulation equivalence:

$TS_1 \approx TS_2$ iff there exists a stutter-bisimulation for (TS_1, TS_2)

stutter-bisimulation equivalence with divergence:

$TS_1 \approx^{div} TS_2$ iff there exists a divergence-sensitive stutter bisimulation for (TS_1, TS_2)

Divergence sensitivity

- Stutter paths are paths that only consist of stutter steps
  - no restrictions are imposed on such paths by stutter bisimulation
  - $\Rightarrow$ stutter trace-equivalence ($\trianglelefteq$) and stutter bisimulation ($\approx$) are incomparable
  - $\Rightarrow$ $\approx$ and $LTL_{\setminus X}$ equivalence are incomparable
- Stutter paths diverge: they never leave an equivalence class
- Remedy: only relate divergent states or non-divergent states
  - divergent state = a state that has a stutter path
  - $\Rightarrow$ relate states only if they either both have stutter paths or none of them
- This yields divergence-sensitive stutter bisimulation ($\approx^{div}$)
  - $\Rightarrow$ $\approx^{div}$ is strictly finer than $\trianglelefteq$ (and $\approx$)
  - $\Rightarrow$ $\approx^{div}$ and $CTL^*_{\setminus X}$ equivalence coincide

Divergence sensitivity

Let TS be a transition system and $\mathcal{R}$ an equivalence relation on S

- s is $\mathcal{R}$-divergent if there exists an infinite path fragment

  $s\, s_1\, s_2 \ldots \in Paths(s)$ such that $(s, s_j) \in \mathcal{R}$ for all $j > 0$
  - s is $\mathcal{R}$-divergent if there is an infinite path starting in s that only visits $[s]_{\mathcal{R}}$
- $\mathcal{R}$ is divergence sensitive if for any $(s_1, s_2) \in \mathcal{R}$:

$$s_1 \text{ is } \mathcal{R}\text{-divergent implies } s_2 \text{ is } \mathcal{R}\text{-divergent}$$

  - $\mathcal{R}$ is divergence-sensitive if in any $[s]_{\mathcal{R}}$ either all or none of the states are $\mathcal{R}$-divergent

Divergence-sensitive stutter bisimulation

s_1, s_2 in TS are divergent stutter-bisimilar, denoted $s_1 \approx_{TS}^{div} s_2$, if:

$\exists$ divergent-sensitive stutter bisimulation $\mathcal{R}$ on TS such that $(s_1, s_2) \in \mathcal{R}$

$\approx_{TS}^{div}$ is an equivalence, the coarsest divergence-sensitive stutter bisimulation for TS

and the union of all divergence-sensitive stutter bisimulations for TS

Quotient transition system under $\approx^{div}$

For $TS = (S, Act, \to, I, AP, L)$ and divergent-sensitive stutter bisimulation $\approx^{div} \subseteq S \times S$,

$TS/\approx^{div} = (S', \{\tau\}, \to', I', AP, L')$ is the quotient of TS under $\approx^{div}$

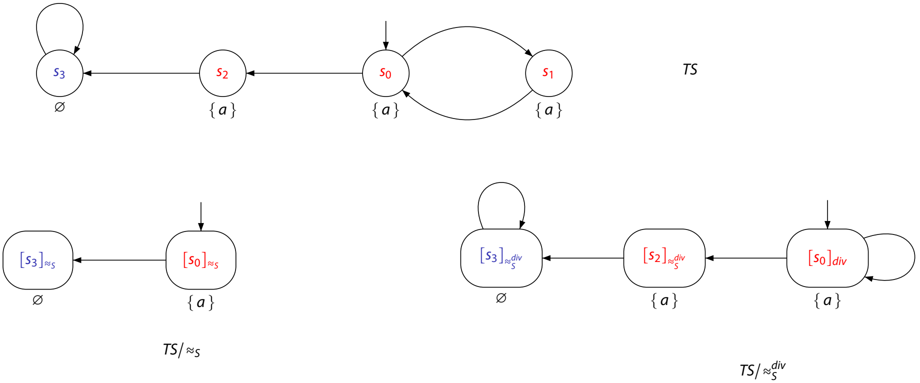
where

- S', I' and L' are defined as usual (for eq. classes $[s]_{div}$ under $\approx^{div}$)
- $\to'$ is defined by:

$$\frac{s \xrightarrow{\alpha} s' \ \wedge\ s \not\approx^{div} s'}{[s]_{div} \xrightarrow{\tau}'_{div} [s']_{div}} \quad \text{and} \quad \frac{s \text{ is } \approx^{div}\text{-divergent}}{[s]_{div} \xrightarrow{\tau}'_{div} [s]_{div}}$$

note that $TS \approx^{div} TS/\approx^{div}$

Example



s_3
s_2
s_0
s_1
$\varnothing$
$\{a\}$
$\{a\}$
$\{a\}$
TS
$[s_3]_{\approx_S}$
$[s_0]_{\approx_S}$
$\varnothing$
$\{a\}$
$TS/\approx_S$
$[s_3]_{\approx_S^{div}}$
$[s_2]_{\approx_S^{div}}$
$[s_0]_{div}$
$\varnothing$
$\{a\}$
$\{a\}$
$TS/\approx_S^{div}$

$\approx^{div}$ on paths

For infinite path fragments $\pi_i = s_{0,i}\, s_{1,i}\, s_{2,i} \ldots$, $i = 1, 2$, in TS:

$$\pi_1 \;\approx^{div}_{TS}\; \pi_2$$

if and only if there exists an infinite sequence of indexes

$$0 = j_0 < j_1 < j_2 < \ldots \quad \text{and} \quad 0 = k_0 < k_1 < k_2 < \ldots$$

with:

$$s_{j,1} \;\approx^{div}_{TS}\; s_{k,2} \text{ for all } j_{r-1} \leq j < j_r \text{ and } k_{r-1} \leq k < k_r \text{ with } r = 1, 2, \ldots .$$

Comparing paths by $\approx^{div}$

Let $TS = (S, Act, \rightarrow, I, AP, L)$, $s_1, s_2 \in S$. Then:

$$s_1 \approx_{TS}^{div} s_2 \quad \text{implies} \quad \forall \pi_1 \in Paths(s_1).\ \left(\exists \pi_2 \in Paths(s_2).\ \pi_1 \approx_{TS}^{div} \pi_2\right)$$

Stutter equivalence versus $\approx^{div}$

Let TS_1 and TS_2 be transition systems over AP. Then:

$$\underbrace{TS_1 \approx^{div} TS_2}_{\text{stutter-bisimulation equivalence with divergence}} \quad \text{implies} \quad \underbrace{TS_1 \cong TS_2}_{\text{stutter-trace equivalence}}$$

whereas the reverse implication does not hold in general

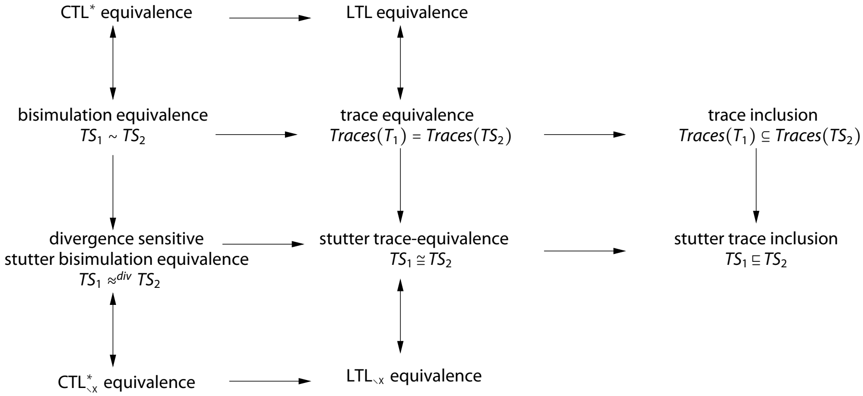
$\mathrm{CTL}^*_{\setminus X}$ equivalence and $\approx^{div}$

For finite transition systems *TS* without terminal states, and s_1, s_2 in *TS*:

$$s_1 \approx^{div}_{TS} s_2 \quad \text{iff} \quad s_1 \equiv_{\mathrm{CTL}^*_{\setminus X}} s_2 \quad \text{iff} \quad s_1 \equiv_{\mathrm{CTL}_{\setminus X}} s_2$$

divergent-sensitive stutter bisimulation coincides with $\mathrm{CTL}_{\setminus X}$ and $\mathrm{CTL}^*_{\setminus X}$ equivalence

Comparative semantics



Timed Automata

Time-critical systems

- Timing issues are of crucial importance for many systems, e.g.,
  - landing gear controller of an airplane, railway crossing, robot controllers
  - steel production controllers, communication protocols . . . . . .
- In time-critical systems correctness depends on:
  - not only on the logical result of the computation, but
  - also on the time at which the results are produced
- How to model timing issues:
  - discrete-time or continuous-time?

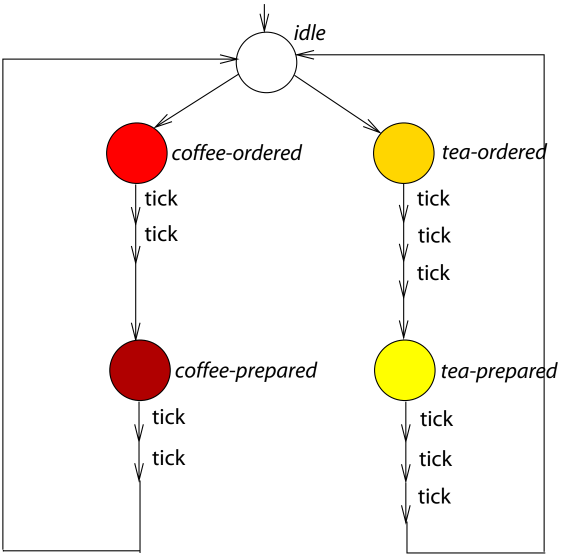
A discrete time domain

- Time has a discrete nature, i.e., time is advanced by discrete steps
  - time is modelled by naturals; actions can only happen at natural time values
  - a specific tick action is used to model the advance of one time unit
  - ⇒ delay between any two events is always a multiple of the minimal delay of one time unit
- Properties can be expressed in traditional temporal logic
  - the next-operator "measures" time
  - two time units after being red, the light is green: $\mathrm{G}\,(\mathit{red} \Rightarrow \mathrm{X}\,\mathrm{X}\,\mathit{green})$
  - within two time units after red, the light is green:

$$\mathrm{G}\,(\mathit{red} \Rightarrow (\mathit{green} \vee \mathrm{X}\,\mathit{green} \vee \mathrm{X}\,\mathrm{X}\,\mathit{green}))$$

- Main application area: synchronous systems, e.g., hardware

A discrete-time coffee machine

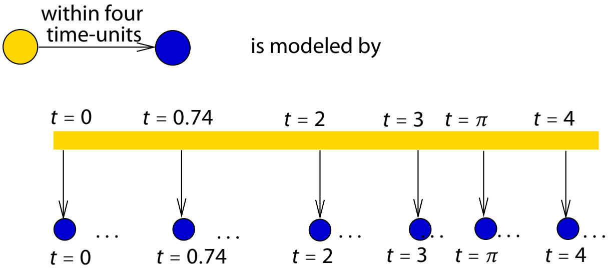


A discrete time domain

- Main advantage: conceptual simplicity
  - state graphs systems equipped with a "tick" transition suffice
  - standard temporal logics can be used
  - ⇒ traditional model-checking algorithms suffice
- Main limitations:
  - (minimal) delay between any pair of actions is a multiple of an a priori fixed minimal delay
  - ⇒ difficult (or impossible) to determine this in practice
  - ⇒ limits modeling accuracy
  - ⇒ inadequate for asynchronous systems. e.g., distributed systems

A continuous time-domain

If time is continuous, state changes can happen at any point in time:



but: infinitely many states and infinite branching

How to check a property like:

once in a yellow state, eventually the system is in a blue state within π *time-units?*

Approach

- Restrict expressivity of the property language
  - e.g., only allow reference to natural time units

$\Longrightarrow$ Timed CTL

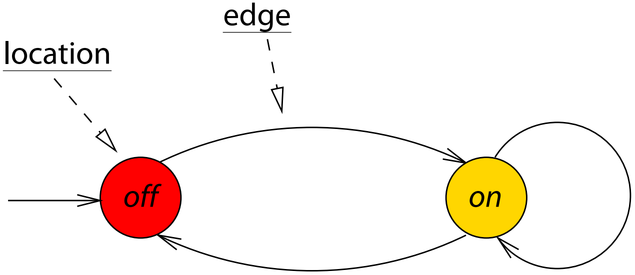
- Model timed systems symbolically rather than explicitly

$\Longrightarrow$ Timed Automata

- Consider a finite quotient of the infinite state space on-demand
  - i.e., using an equivalence that depends on the property and the timed automaton

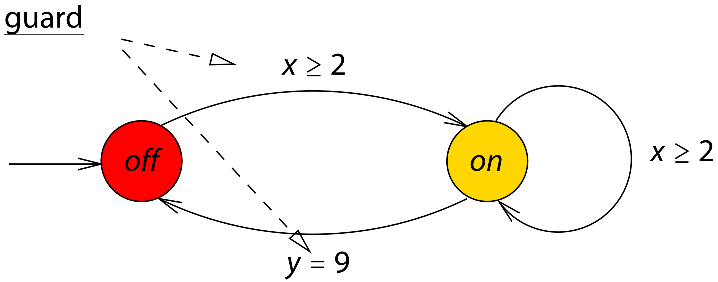
$\Longrightarrow$ Region Automata

What is a timed automaton?



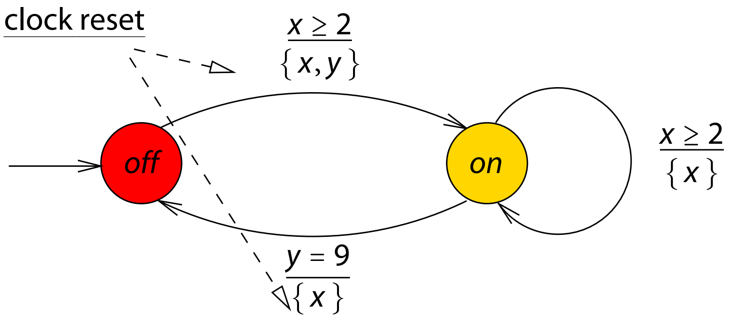
- a program graph with locations and edges
- a location is labeled with the valid atomic propositions
- taking an edge is instantaneous, i.e, consumes no time

What is a timed automaton?



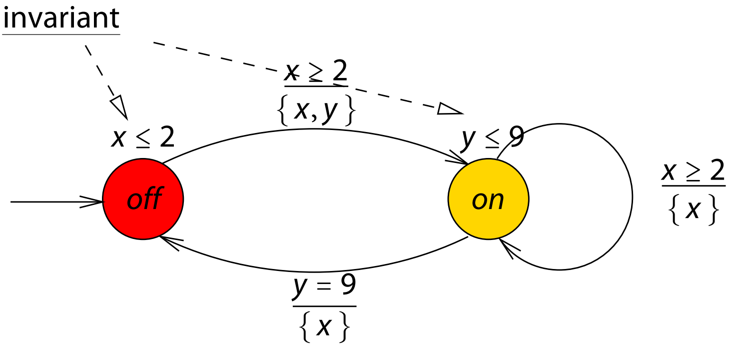
- equipped with real-valued clocks $x, y, z, \ldots$
- clocks advance implicitly, all at the same speed
- logical constraints on clocks can be used as guards of actions

What is a timed automaton?



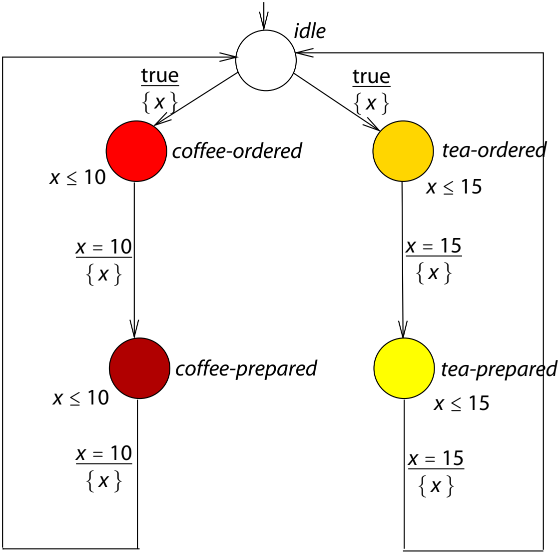
- clocks can be reset when taking an edge
- assumption:
  all clocks are zero when entering the initial location initially

What is a timed automaton?



- guards indicate when an edge may be taken
- a location invariant specifies the amount of time that may be spent in a location
  - before a location invariant becomes invalid, an edge must be taken

A real-time coffee machine



Clock constraints

- Clock constraints over set C of clocks are defined by:

$$g ::= \text{true} \mid x < c \mid x - y < c \mid x \leq c \mid x - y \leq c \mid \neg g \mid g \wedge g$$

  - where $c \in \mathbb{N}$ and clocks $x, y \in C$
  - rational constants would do; neither reals nor addition of clocks!
  - let $CC(C)$ denote the set of clock constraints over C
  - shorthands: $x \geq c$ denotes $\neg(x < c)$ and $x \in [c_1, c_2)$ or $c_1 \leq x < c_2$ denotes $\neg(x < c_1)$ & $(x < c_2)$
- Atomic clock constraints do not contain true, $\neg$ and $\wedge$
  - let $ACC(C)$ denote the set of atomic clock constraints over C
- Simplification: In the following, we assume constraints are diagonal-free, i.e., do neither contain $x - y \leq c$ nor $x - y < c$.

Timed automaton

A timed automaton is a tuple

$$TA = (Loc, Act, C, \rightsquigarrow, Loc_0, inv, AP, L) \quad \text{where:}$$

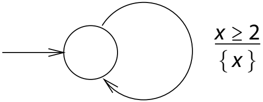
- Loc is a finite set of locations.
- $Loc_0 \subseteq Loc$ is a set of initial locations
- C is a finite set of clocks
- $L : Loc \to 2^{AP}$ is a labeling function for the locations
- $\rightsquigarrow \subseteq Loc \times CC(C) \times Act \times 2^C \times Loc$ is a transition relation, and
- $inv : Loc \to CC(C)$ is an invariant-assignment function

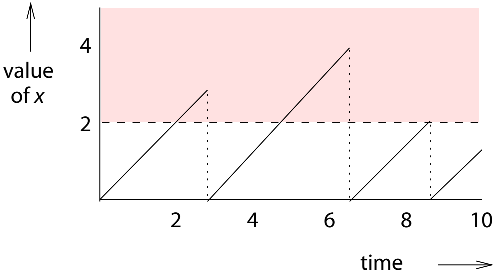
Intuitive interpretation

- Edge $\ell \xrightarrow{g:\alpha,C'} \ell'$ means:
  - action α is enabled once guard g holds
  - when moving from location ℓ to ℓ', any clock in C' will be reset to zero
- $\mathit{inv}(\ell)$ constrains the amount of time that may be spent in location ℓ
  - the location ℓ must be left before the invariant $\mathit{inv}(\ell)$ becomes invalid

Guards versus location invariants

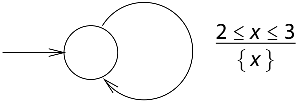
The effect of a lowerbound guard:

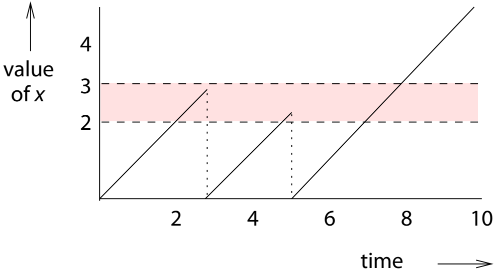




Guards versus location invariants

The effect of a lowerbound and upperbound guard:





Guards versus location invariants

The effect of a guard and an invariant: