Verification – Lecture 23
Simulation Quotients (continued)

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

Wintersemester 2007/2008

NEW YORK, February 4, 2008 — *ACM has named Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis the winners of the 2007 A.M. Turing Award for their original and continuing research in a quality assurance process known as Model Checking.*

Simulation order

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

but <u>not</u> necessarily:

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

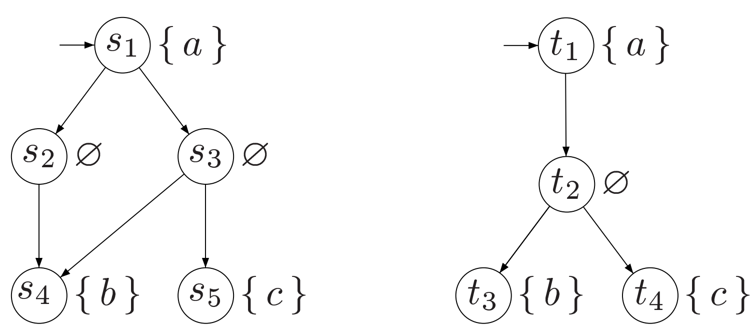
Simulation is a pre-order

$\preceq$ is a preorder, i.e., reflexive and transitive

Simulation equivalence

S_1 and S_2 are *simulation equivalent*, denoted $S_1 \simeq S_2$,
if $S_1 \preceq S_2$ and $S_2 \preceq S_1$

Similar but not bisimilar



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

Simulation order on states

A *simulation* for $S = (Q, Q_0, E, L)$ 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 $q_1' \in$ *Successors*(q_1)
   then there exists an $q_2' \in$ *Successors*(q_2) with $(q_1', q_2') \in \mathcal{R}$

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

$q_1 \preceq_S q_2$ if and only if $S_{q_1} \preceq S_{q_2}$

$q_1 \simeq_S q_2$ if and only if $q_1 \preceq_S q_2$ and $q_2 \preceq_S q_1$

Simulation quotient

For $S = (Q, Q_0, E, L)$ and simulation equivalence $\simeq \subseteq Q \times Q$ let

$$S/\simeq \; = \; (Q', Q_0', E', L'), \qquad \text{the } \textit{quotient} \text{ of } S \text{ under } \simeq$$

where

- $Q' = Q/\simeq = \{ [q]_\simeq \mid q \in Q \}$ and $Q' = \{ [q]_\simeq \mid q \in Q_0 \}$
- $E' = \{([q]_\simeq, [q']_\simeq) \mid (q, q') \in E\}$.
- $L'([s]_\simeq) = L(s)$

lemma: $S \simeq S/\simeq$; proof not straightforward!

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

The release operator

- The *release* operator: $\varphi \, \mathsf{R} \, \psi \ \stackrel{\text{def}}{=} \ \neg(\neg\varphi \, \mathsf{U} \, \neg\psi)$
  - ψ always holds, a requirement that is released as soon as φ holds
- Until U and release R are *dual*:

$$\begin{aligned} \varphi \, \mathsf{U} \, \psi &\equiv \neg(\neg\varphi \, \mathsf{R} \, \neg\psi) \\ \varphi \, \mathsf{R} \, \psi &\equiv \neg(\neg\varphi \, \mathsf{U} \, \neg\psi) \end{aligned}$$

- Release satisfies the *expansion law*: $\varphi \, \mathsf{R} \, \psi \equiv \psi \ \wedge \ (\varphi \ \vee \ \bigcirc(\varphi \, \mathsf{R} \, \psi))$

Universal CTL* contains LTL

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

Proof: Bring LTL formula into positive normal form (PNF).

For $a \in AP$, LTL formulas in PNF are given by:

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

Transformation

For any LTL-formula φ there exists
an equivalent LTL-formula ψ in PNF with $|\psi| = \mathcal{O}(|\varphi|)$

Transformations:

$$\begin{array}{lcl}
\neg\text{true} & \rightsquigarrow & \text{false} \\
\neg\neg\varphi & \rightsquigarrow & \varphi \\
\neg(\varphi \wedge \psi) & \rightsquigarrow & \neg\varphi \vee \neg\psi \\
\neg(\varphi \vee \psi) & \rightsquigarrow & \neg\varphi \wedge \neg\psi \\
\neg\bigcirc\varphi & \rightsquigarrow & \bigcirc\neg\varphi \\
\neg(\varphi \,\mathsf{U}\, \psi) & \rightsquigarrow & \neg\varphi \,\mathsf{R}\, \neg\psi \\
\neg\Diamond\varphi & \rightsquigarrow & \Box\neg\varphi \\
\neg\Box\varphi & \rightsquigarrow & \Diamond\neg\varphi
\end{array}$$

Simulation order and $\forall$CTL*

Let S be a finite state graph (without terminal states) and q, q' states in S.

The following statements are equivalent:

(1) $q \preceq_S q'$

(2) for all $\forall$CTL*-formulas Φ: $q' \models \Phi$ implies $q \models \Phi$

(3) for all $\forall$CTL-formulas Φ: $q' \models \Phi$ implies $q \models \Phi$

proof is carried out in three steps: (1) $\Rightarrow$ (2) $\Rightarrow$ (3) $\Rightarrow$ (1)

Existential fragment of CTL*

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

$$\Phi ::= \text{true} \mid \text{false} \mid a \mid \neg a \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \exists \varphi$$

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

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

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

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

Simulation order and $\exists$CTL*

Let S be a finite state graph (without terminal states) and q, q' states in S.

The following statements are equivalent:

(1) $q \preceq_S q'$

(2) for all $\exists$CTL*-formulas Φ: $q \models \Phi$ implies $q' \models \Phi$

(3) for all $\exists$CTL-formulas Φ: $q \models \Phi$ implies $q' \models \Phi$

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

For finite state graph S without terminal states:

$$\simeq_S \; = \; \equiv_{\forall \text{CTL}^*} \; = \; \equiv_{\forall \text{CTL}} \; = \; \equiv_{\exists \text{CTL}^*} \; = \; \equiv_{\exists \text{CTL}}$$

Skeleton for simulation preorder checking

Input: finite state graph $S = (Q, Q_0, E, L)$ over *AP*
Output: simulation order $\preceq_S$

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

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 $(q_2, q_2') \in E$, $(q_1', q_2') \notin \mathcal{R}$;
  $\mathcal{R} := \mathcal{R} \setminus \{\, (q_1, q_2) \,\}$
od
return $\mathcal{R}$

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

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

Complexity

For $S = (Q, Q_0, E, L)$ with $|E| \geqslant |Q|$:

$\boxed{\text{Time complexity of computing } \prec_S \text{ is } \mathcal{O}\big(|Q|\cdot|AP| + |E|\cdot|Q|\big)}$

Details are non-trivial. See Baier/Katoen Section 7.6.

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 $\mathcal{O}(\lvert E\rvert \cdot \log \lvert Q\rvert)$	PTIME $\mathcal{O}(\lvert E\rvert \cdot \lvert Q\rvert)$	—

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

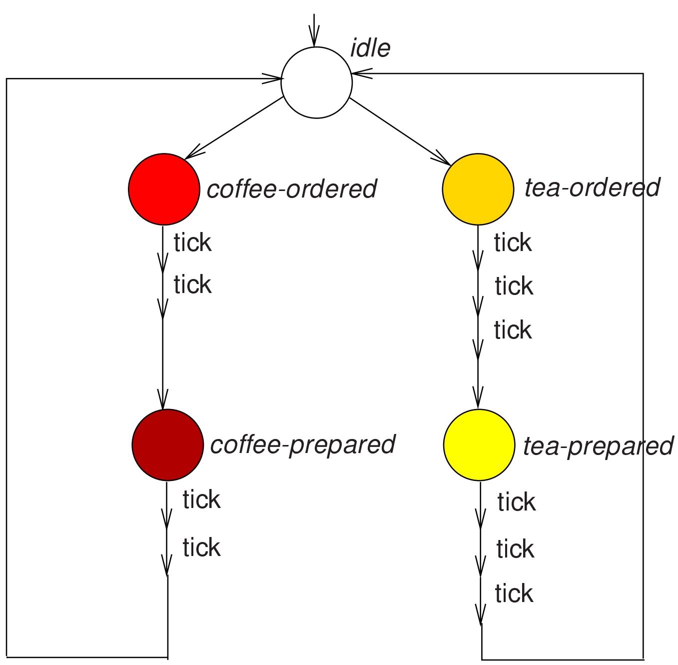
  $\Rightarrow$ delay between any two events is always a multiple of the minimal delay of one time unit

- Properties can be expressed in traditional temporal logic
  - the next-operator "measures" time
  - two time units after being red, the light is green: $\square\,(red \;\Rightarrow\; \bigcirc\bigcirc green)$
  - within two time units after red, the light is green:

$$\square\,(red \;\Rightarrow\; (green \;\vee\; \bigcirc green \;\vee\; \bigcirc\bigcirc green))$$

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

A discrete-time coffee machine



A discrete time domain

- Main advantage: conceptual simplicity
  - labeled transition systems equipped with a tick actions 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

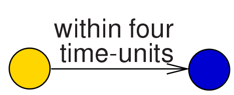
  ⇒ difficult (or impossible) to determine this in practice

  ⇒ limits modeling accuracy

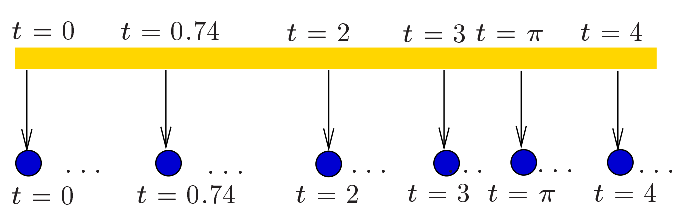
  ⇒ inadequate for *asynchronous* systems. e.g., distributed systems

A continuous time-domain

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



is modeled by



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

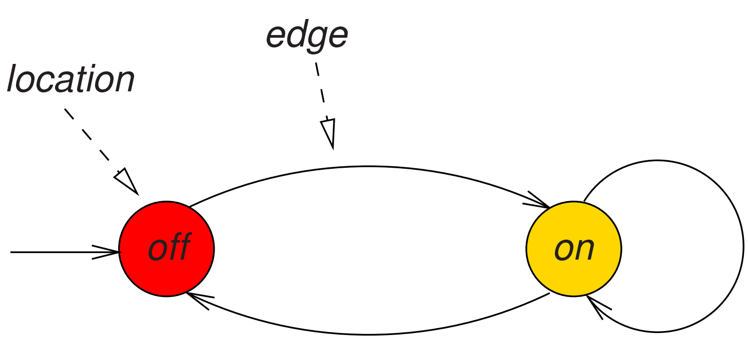
- Model timed systems *symbolically* rather than explicitly

$\Longrightarrow$ 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

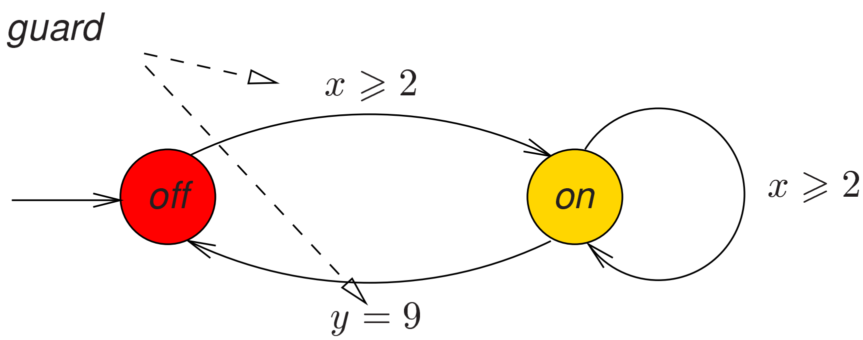
$\Longrightarrow$ Region Automata

What is a timed automaton?



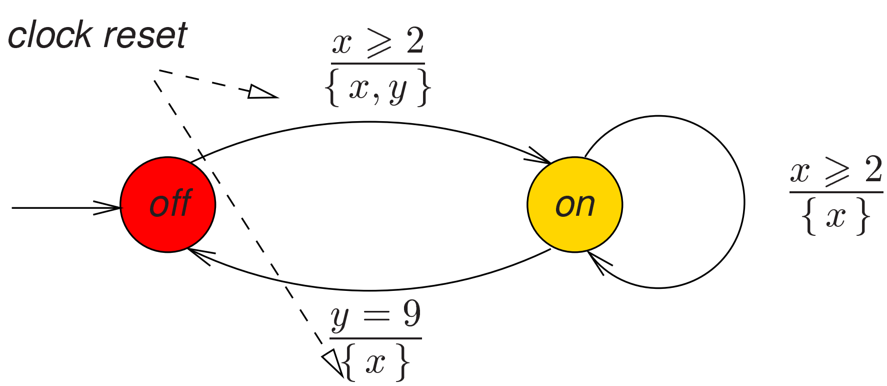
- a program graph with *locations* and *edges*
- a location is labeled with the valid *atomic propositions*
- *taking an edge is instantaneous*, i.e, consumes no time

What is a timed automaton?



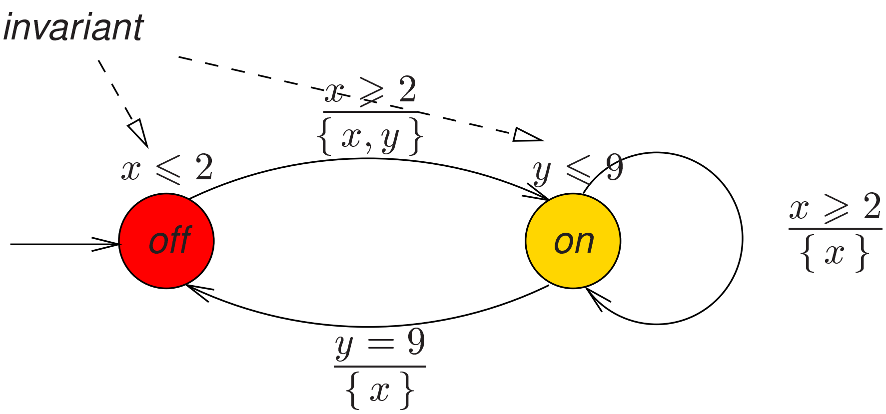
- equipped with real-valued *clocks* $x, y, z, \ldots$
- clocks advance implicitly, all at the *same speed*
- logical constraints on clocks can be used as *guards* of actions

What is a timed automaton?



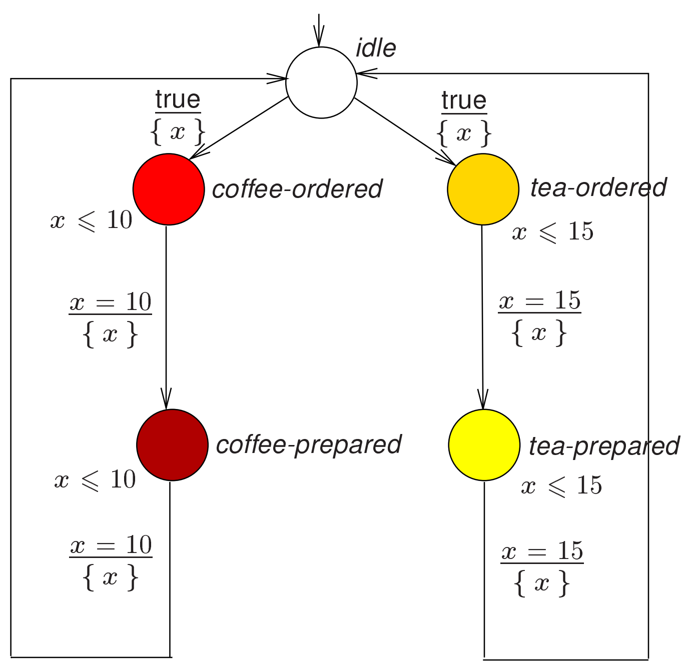
- clocks can be *reset* when taking an edge
- assumption:
  all clocks are zero when entering the initial location initially

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*
  - when 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:

$$g ::= \text{true} \;\Big|\; x < c \;\Big|\; x - y < c \;\Big|\; x \leqslant c \;\Big|\; x - y \leqslant c \;\Big|\; \neg g \;\Big|\; g \wedge g$$

  - where $c \in \mathbb{N}$ and clocks $x, y \in C$
  - rational constants would do; neither reals nor addition of clocks!
  - let $\mathit{CC}(C)$ denote the set of clock constraints over C
  - shorthands: $x \geqslant c$ denotes $\neg(x < c)$ and $x \in [c_1, c_2)$ or $c_1 \leqslant x < c_2$ denotes $\neg(x < c_1) \wedge (x < c_2)$

- *Atomic clock constraints* do not contain true, $\neg$ and $\wedge$
  - let $\mathit{ACC}(C)$ denote the set of atomic clock constraints over C

Timed automaton

A *timed automaton* is a tuple

$$\mathit{TA} = \big(\mathit{Loc}, \mathit{Act}, C, \rightsquigarrow, \mathit{Loc}_0, \mathit{inv}, \mathit{AP}, L\big) \quad \text{where:}$$

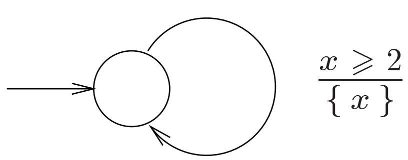
- *Loc* is a finite set of locations.
- $\mathit{Loc}_0 \subseteq \mathit{Loc}$ is a set of initial locations
- C is a finite set of clocks
- $L : \mathit{Loc} \rightarrow 2^{\mathit{AP}}$ is a labeling function for the locations
- $\rightsquigarrow \subseteq \mathit{Loc} \times \mathit{CC}(C) \times \mathit{Act} \times 2^C \times \mathit{Loc}$ is a transition relation, and
- $\mathit{inv} : \mathit{Loc} \rightarrow \mathit{CC}(C)$ is an invariant-assignment function

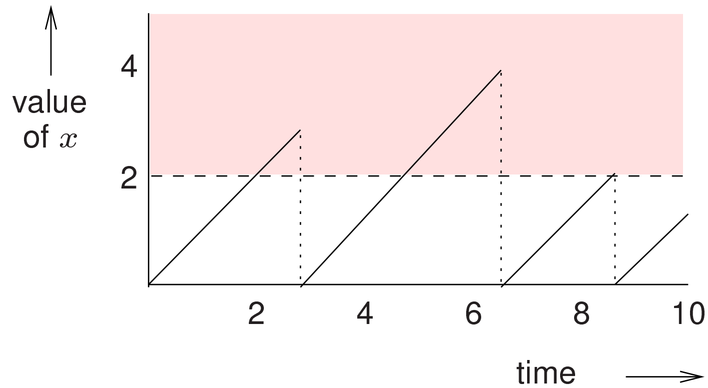
Intuitive interpretation

- 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
- *inv*(ℓ) constrains the amount of time that may be spent in location ℓ
  - the location ℓ must be left before the invariant *inv*(ℓ) becomes invalid

Guards versus location invariants

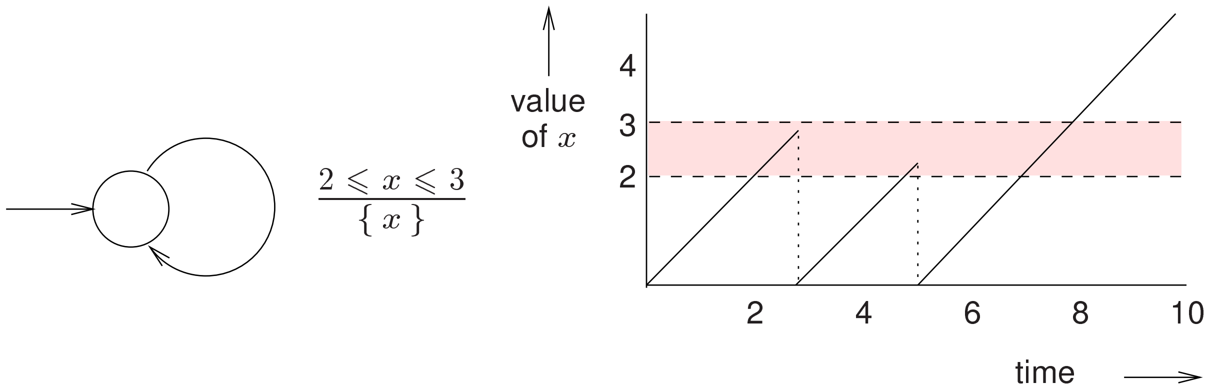
The effect of a lowerbound guard:





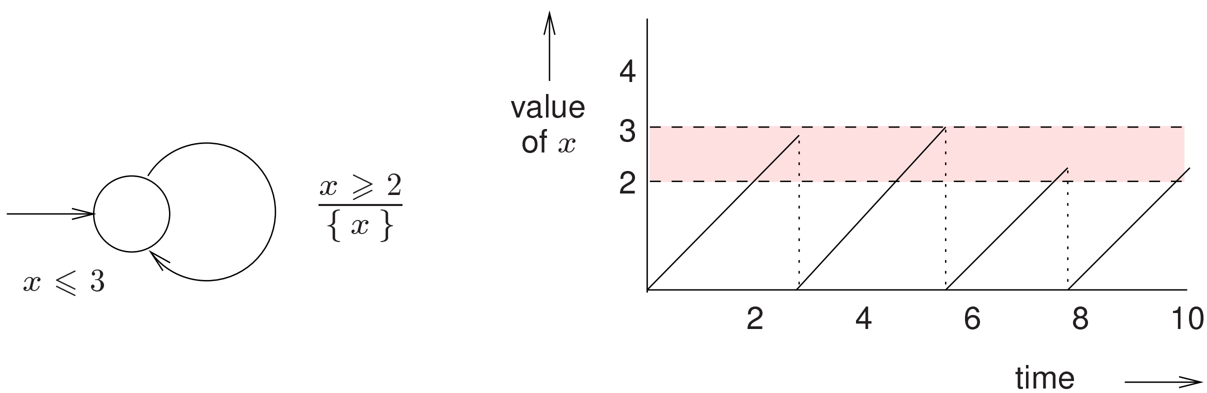
Guards versus location invariants

The effect of a lowerbound and upperbound guard:

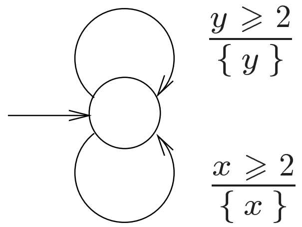


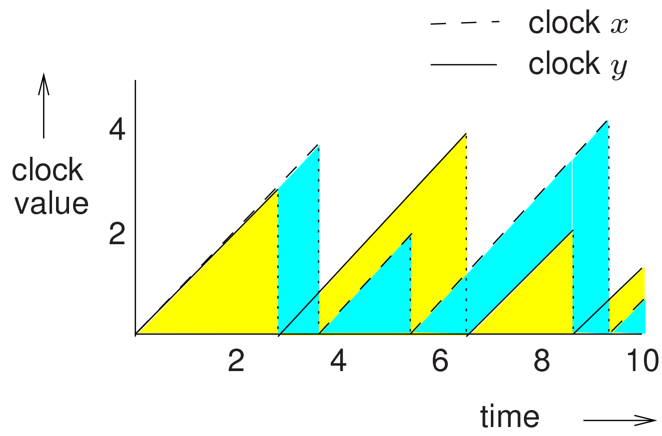
Guards versus location invariants

The effect of a guard and an invariant:



Arbitrary clock differences





This is impossible to model in a discrete-time setting