Formal Verification of Real Time Systems
Timed Automata

Radek Pelánek

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.

EVROPSKÁ UNIE · esf · MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY · OP Vzdělávání pro konkurenceschopnost

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Aim of the Lecture

- knowledge of a basic formalism for modeling timed systems
- basic understanding of verification algorithms for timed systems

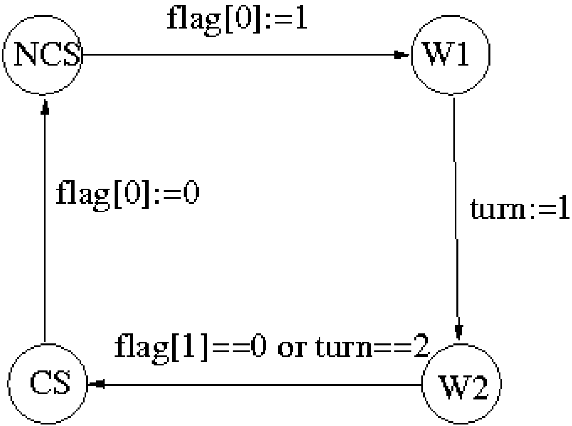
Example: Peterson's Algorithm

- `flag[0]`, `flag[1]` (initialed to `false`) – meaning *I want to access CS*
- `turn` (initialized to `0`) – used to resolve conflicts

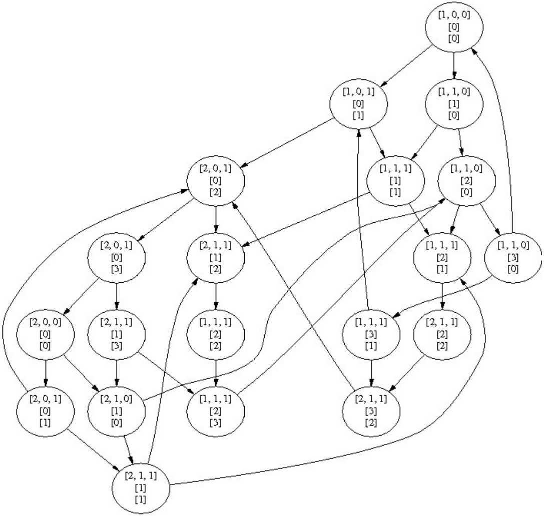
```
Process 0:
while (true) {
   <noncritical section>;
   flag[0] := true;
   turn := 1;
   while flag[1] and
         turn = 1 do { };
   <critical section>;
   flag[0] := false;
}
```

```
Process 1:
while (true) {
   <noncritical section>;
   flag[1] := true;
   turn := 0;
   while flag[0] and
         turn = 0 do { };
   <critical section>;
   flag[1] := false;
}
```

Example: Peterson's Algorithm



Example: Peterson's Algorithm



Fischer's Protocol

- real-time protocol – correctness depends on timing assumptions
- simple, just 1 shared variable, arbitrary number of processes
- assumption: known upper bound D on reading/writing variable in shared memory
- each process has it's own timer (for delaying)

Fischer's Protocol

- id – shared variable, initialized -1
- each process has it's own timer (for delaying)
- for correctness it is necessary that $K > D$

```
Process i:
while (true) {
    <noncritical section>;
    while id != -1 do {}
    id := i;
    delay K;
    if (id = i) {
        <critical section>;
        id := -1;
    }
}
```

Modeling Fischer's Protocol

- how do we model clocks?
- how do we model waiting (delay)?

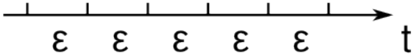
Modeling Real Time Systems

Two models of time:

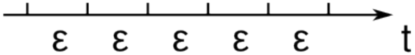
- discrete time domain
- continuous time domain

Discrete Time Domain

- clocks tick at regular interval
- at each tick something may happen
- between ticks – the system only waits



Discrete Time Domain



- choose a fixed sample period ϵ
- all events happen at multiples of ϵ
- simple extension of classical model (time = new integer variable)
- main disadvantage – how to choose ϵ?
  - big $\epsilon \Rightarrow$ too coarse model
  - low $\epsilon \Rightarrow$ time fragmentation, too big state space
- usage: particularly synchronous systems (hardware circuits)

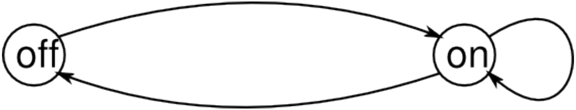
Continuous Time Domain

- time $\sim$ real number
- delays may be arbitrarily small
- more faithful model, suited for asynchronous systems
- model checking (automatic verification) $\sim$ traversal of state space
- uncountable state space $\Rightarrow$ cannot be directly handled automatically by "brute force"

Timed Automata

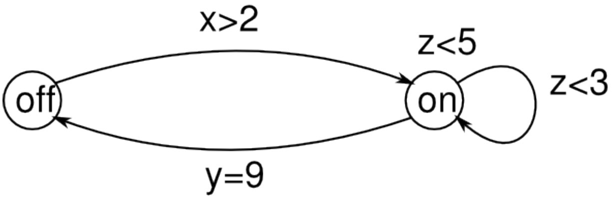
- extension of finite state machines with clocks
- continuous real semantics
- limited list of operations over clocks $\Rightarrow$ automatic verification is feasible
- allowed operations:
  - comparison of a clock with a constant
  - reset of a clock
  - uniform flow of time (all clocks have the same rate)
- note: even simple extensions lead to undecidability

What is a Timed Automaton?



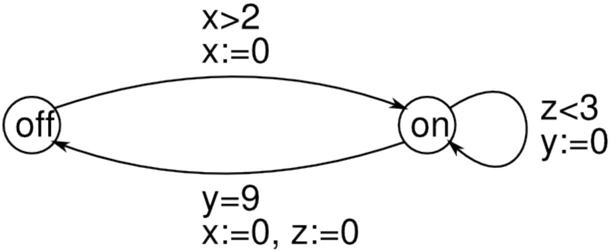
- an automaton with locations (states) and edges
- the automaton spends time only in locations, not in edges

What is a Timed Automaton? (2)



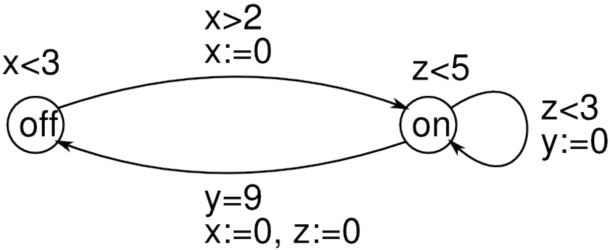
- real valued clocks
- all clocks run at the same speed
- clock constraints can be guards on edges

What is a Timed Automaton? (3)



- clocks can be reseted when taking an edge
- only a reset to value 0 is allowed

What is a Timed Automaton? (4)



- location invariants forbid to stay in a state too long
- invariants force taking an edge

Clock Constraints

Definition (Clock constraints)

Let X be a set of clock variables. Then set $C(X)$ of clock constraints is given by the following grammar:

$$\phi \equiv x \leq k \mid k \leq x \mid x < k \mid k < x \mid \phi \wedge \phi$$

where $x \in X, k \in N$.

Timed Automata Syntax

Definition (Timed Automaton)

A timed automaton is a 4-tuple: $A = (L, X, l_0, E)$

- L is a finite set of locations
- X is a finite set of clocks
- $l_0 \in L$ is an initial location
- $E \subseteq L \times C(X) \times 2^X \times L$ is a set of edges

edge $=$ (source location, clock constraint, set of clocks to be resetted, target location)

Semantics: Main Idea

- semantics is a state space
  (reminder: guarded command language, extended finite state machines)
- states given by:
  - location (local state of the automaton)
  - clock valuation
- transitions:
  - waiting – only clock valuation changes
  - action – change of location

Clock Valuations

- a clock valuation is a function $\nu : X \to \mathbb{R}^+$
- $\nu[Y := 0]$ is the valuation obtained from ν by resetting clocks from Y:

$$\nu[Y := 0](x) = \begin{cases} 0 & x \in Y \\ x & \text{otherwise} \end{cases}$$

- $\nu + d$ = flow of time (d units):

$$(\nu + d)(x) = \nu(x) + d$$

- $\nu \models c$ means that valuation ν satisfies the constraint c

Evaluation of Clock Constraints

Evaluation of a clock constraint ($\nu \models g$):

- $\nu \models x < k$ iff $\nu(x) < k$
- $\nu \models x \leq k$ iff $\nu(x) \leq k$
- $\nu \models g_1 \wedge g_2$ iff $\nu \models g_1$ and $\nu \models g_2$

Examples

let $\nu = (x \to 3, y \to 2.4, z \to 0.5)$

- what is $\nu[y := 0]$?
- what is $\nu + 1.2$?
- does $\nu \models y < 3$?
- does $\nu \models x < 4 \wedge z \geq 1$?

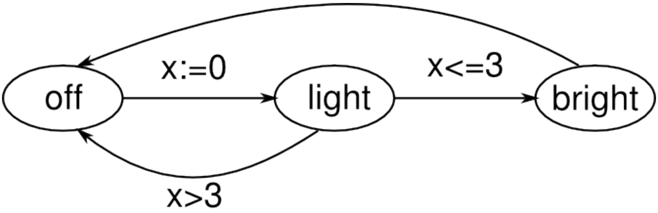
Timed Automata Semantics

Definition (Timed automata semantics)

The semantics of a timed automaton A is a transition system $S_A = (S, s_0, \longrightarrow)$:

- $S = L \times (X \to \mathbb{R}^+)$
- $s_0 = (l_0, \nu_0)$, $\nu_0(x) = 0$ for all $x \in X$
- transition relation $\longrightarrow \subseteq S \times S$ is defined as:
  - (delay action) $(l, \nu) \xrightarrow{\delta} (l, \nu + \delta)$
  - (discrete action) $(l, \nu) \longrightarrow (l', \nu')$ iff there exists $(l, c, Y, l') \in E$ such that $\nu \models c, \nu' = \nu[Y := 0]$

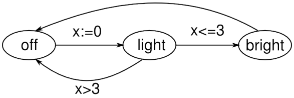
Example



- What is a clock valuation?
- What is a state?
- Find a run = sequence of states

Example

- clock valuation: assignment of a real value to x
- initial state $(off, 0)$
- example of a run:
  $(off, 0) \xrightarrow{2.4} (off, 2.4) \longrightarrow$
  $(light, 0) \xrightarrow{1.5}$
  $(light, 1.5) \longrightarrow$
  $(bright, 1.5) \longrightarrow \ldots$



Example

Construct a timed automaton, which models the following schedule of a student:

- the student wakes up between 7 and 9
- if the student wakes up before 8, he has a breakfast, which takes exactly 15 minutes
- the students travels to school, it takes between 30 and 45 minutes
- if the student arrives to school before 10, he goes to the lecture, otherwise he goes to the library

Semantics: Notes

- the semantics is infinite state (even uncountable)
- the semantics is even infinitely branching

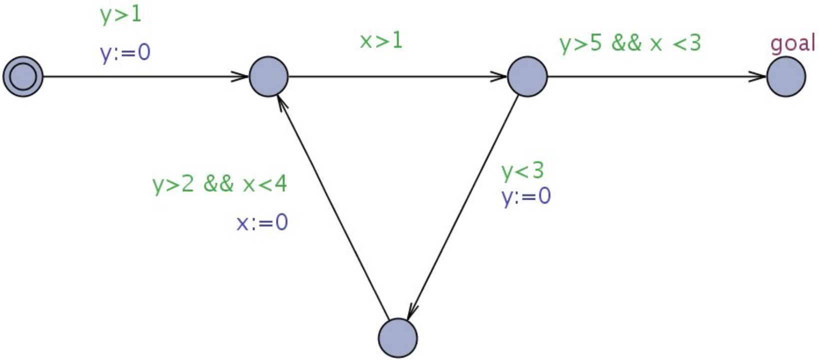
Reachability Problem

Reachability Problem

Input: a timed automaton A, a location l of the automaton
Question: does there exists a run of A which ends in l

This problem formalises the verification of *safety* problems – is an erroneous state reachable?

Example



How to do it algorithmically?

Other Verification Problems

- verification of temporal (timed) logic
- equivalence checking – (timed) bisimulation of timed automata
- universality, language inclusion (undecidable)

Reachability: Attempt 1

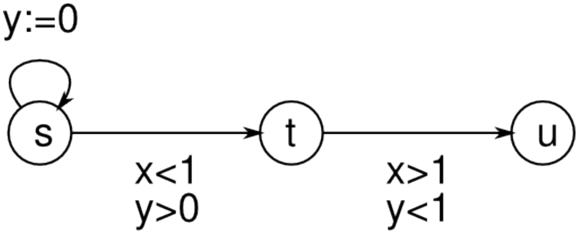
- discretization (sampled semantics)
- allow time step (delay) 1
- clock above maximal constant $\Rightarrow$ value does not increase
- finite state space
- but not equivalent $\Rightarrow$ find counterexample

Reachability: Attempt 2

- what about time step 0.5

Reachability: Attempt 2

- what about time step 0.5



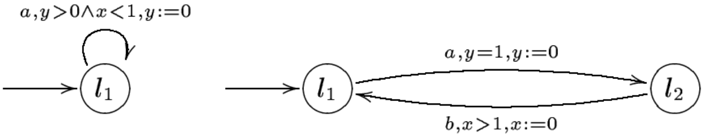
Reachability: Attempt X

- what about time step 0.25?
- what about time step 2^{-n}?

Reachability and Discretization

- for each automaton there exists ϵ such that sampled and dense semantics are reachability equivalent
  - why?
  - how to determine ϵ?
- no fixed ϵ is sufficient for all timed automata
- more complex equivalences (trace equivalence, bisimulation) and verification problems – sampled and dense semantics are not equivalent

Sampled vs Dense Semantics



- dense semantics: arbitrary long words
- sampled semantics: bounded length of words

Another Approach?

- discretization (sampling) is not sufficient
- any other idea?

Another Approach?

- discretization (sampling) is not sufficient
- any other idea?
- is it necessary to distinguish the following valuations? $(0.589, 1.234)$ and $(0.587, 1.236)$

Another Approach?

- discretization (sampling) is not sufficient
- any other idea?
- is it necessary to distinguish the following valuations? $(0.589, 1.234)$ and $(0.587, 1.236)$
- some clock valuations are equivalent $\sim$ the automaton cannot distinguish between them $\sim$ any run possible from one valuation is also possible from the second
- let us find these equivalence classes (regions)

Reachability Problem

Theorem

The reachability problem is PSPACE-complete.

- note that even decidability of the problem is not straightforward – the semantics is infinite state
- decidability proved by region construction (to be discussed)
- completeness proved by general reduction from linearly bounded Turing machine (not discussed)

Region Construction

Main idea:

- some clock valuations are equivalent
- work with regions of valuations instead of valuations
- finite number of regions

Preliminaries

Let $d \in \mathbb{R}^{\geq 0}$. Then:

- let $\lfloor d \rfloor$ be the integer part of d
- let $fr(d)$ be the fractional part of d

Thus $d = \lfloor d \rfloor + fr(d)$.

Example: $\lfloor 42.37 \rfloor = 42$, $fr(42.37) = 0.37$

Equivalence on Clock Valuation

- we want an equivalence $\cong$ such that if $\nu \cong \nu'$ then the automaton "cannot distinguish between ν and ν'"
- formally: bisimulation
- informally: whatever action an automaton can do in ν, it can also do it in ν' (and vice verse, repeatedly)
- what conditions on $\cong$ do we need?

Equivalence on Clock Valuation: Condition 1

Let c_x by the largest constant compared to a clock x ("max bound").

Condition 1:

Clock x is in both valuations ν and ν' are above its max bound, or it has the same integer part in both of them.

$\nu(x) \geq c_x \wedge \nu'(x) \geq c_x$ or $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$

Equivalence on Clock Valuation: Condition 2

Condition 2:

If the value of clock is below its max bound, then either it has zero fractional part in both ν and ν' or in neither of them.

$$\nu(x) \leq c_x \Rightarrow (fr(\nu(x)) = 0 \Leftrightarrow fr(\nu'(x) = 0))$$

Equivalence on Clock Valuation: Condition 3

Condition 3:

For two clocks that are below their max bound, the ordering of fractional parts must be the same in both ν and ν'.

$$\nu(x) \leq c_x \wedge \nu(y) \leq c_y \Rightarrow$$
$$fr(\nu(x)) \leq fr(\nu(y)) \Leftrightarrow fr(\nu'(x)) \leq fr(\nu'(y))$$

Equivalence on Clock Valuation

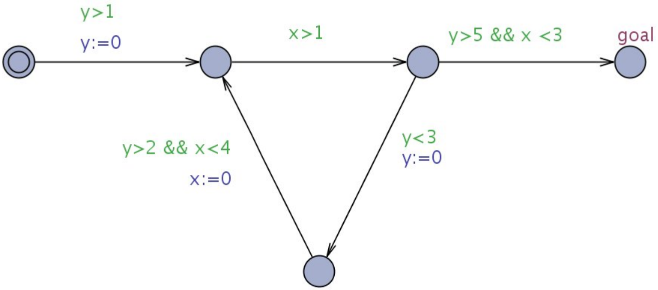
Let c_x by the largest constant compared to a clock x ("max bound").
$\cong$ is equivalence on clock valuations such that $\nu \cong \nu'$ iff for all clocks x, y holds:

1. $\nu(x) \geq c_x \wedge \nu'(x) \geq c_x$ or $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$
2. $\nu(x) \leq c_x \Rightarrow (fr(\nu(x)) = 0 \Leftrightarrow fr(\nu'(x) = 0))$
3. $\nu(x) \leq c_x \wedge \nu(y) \leq c_y \Rightarrow$
   $fr(\nu(x)) \leq fr(\nu(y)) \Leftrightarrow fr(\nu'(x)) \leq fr(\nu'(y))$

Why Do We Need Condition 3?

- Why do we need condition 3, when the automaton cannot compare clocks?
- Find an automaton and clock valuations ν_1, ν_2 such that:
  - ν_1, ν_2 satisfy condition 1 and 2, but not condition 3
  - automaton can "distinguish" between ν_1, ν_2, i.e. there exists timed run r such that r is possible from ν_1 but not from ν_2

Equivalence: Example 1



Identify c_x, c_y

Equivalence: Example 2

- suppose $c_x = 4, c_y = 5, c_z = 1$
- let (x, y, z) denote valuations, decide:
  1. $(0, 0.14, 0.3) \cong (0.05, 0.1, 0.32)$?
  2. $(1.9, 4.2, 0.4) \cong (2.8, 4.3, 0.7)$?
  3. $(0.05, 0.1, 0.3) \cong (0.2, 0.1, 0.4)$?
  4. $(0.03, 1.1, 0.3) \cong (0.05, 1.2, 0.3)$?
  5. $(3.9, 5.3, 0.4) \cong (3.8, 6.9, 0.8)$?

Regions

Definition (Region)

Classes of equivalence $\cong$ are called regions, denoted $[\nu]$.

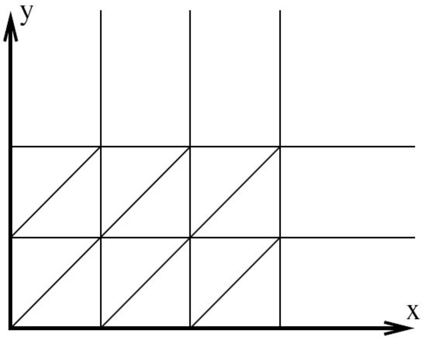
Lemma

The number of regions is at most $|X|! \cdot 2^{|X|} \cdot \prod_{x \in X}(2c_x + 2)$.

Regions: Example

- suppose TA with two clocks, $c_x = 3, c_y = 2$
- draw all regions (since we have just 2 clocks, we can draw them in plane)
- hints:
  - what is the region $[(x = 0.3, y = 0.2)]$?
  - what is the region $[(x = 1.3, y = 0.3)]$?
  - what is the region $[(x = 2.0, y = 1.0)]$?

Regions: Example



Regions for TA with two clocks $c_x = 3, c_y = 2$.

Region Graph

- states are 2-tuples location + clock region: $(l, [\nu])$
- there is a transition from $(l, [\nu])$ to $(l', [\nu'])$ if there exists $\omega \cong \nu, \omega' \cong \nu'$ such that $(l, \omega) \rightarrow (l', \omega')$
- region graph is equivalent to the semantics of A with respect to reachability
  (note: in fact it is equivalent wrt bisimulation equivalence)
- moreover region graph is finite and can be effectively constructed $\Rightarrow$ region graph can be used to answer the reachability problem

Operations on Regions

To construct the region graph, we need the following operations:

- let time pass – go to adjacent region at top right
- intersect with a clock constraint (note that clock constraints define supersets of regions)
  - if region is in the constraint: no change
  - otherwise: empty
- reset a clock – go to a corresponding region

Example: Automaton

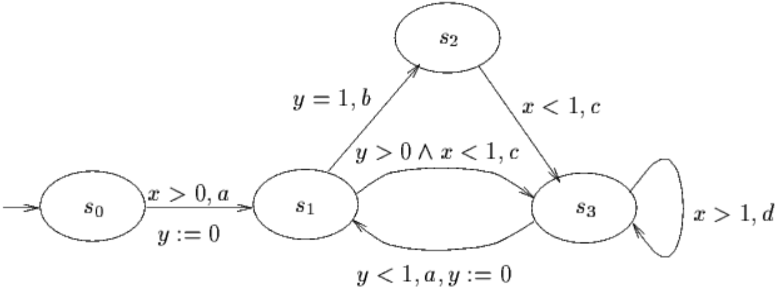


Figure 6: The automaton A_0

(source: R. Alur)

Example: Region Graph

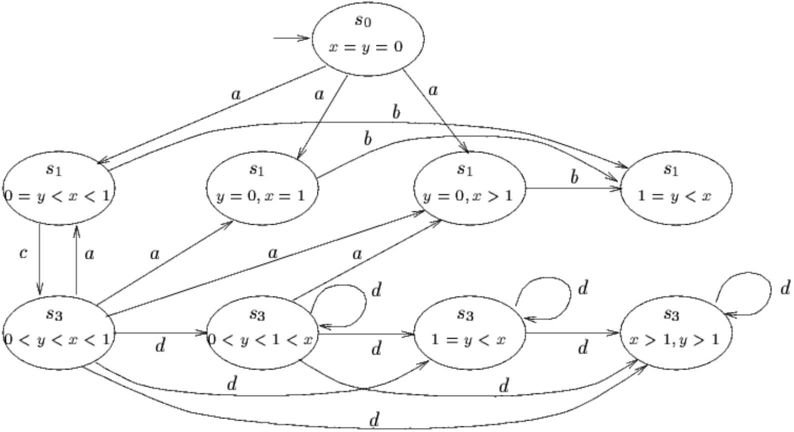


Figure 7: The region automaton $R(A_0)$

Zones

- regions ... nice theory, but inefficient and hard to implement
- zones:
  - convex sets of clock valuations
  - defined by conjunction of constraints $x - y < k$
  - allows efficient representation and manipulation (Difference Bound Matrix)

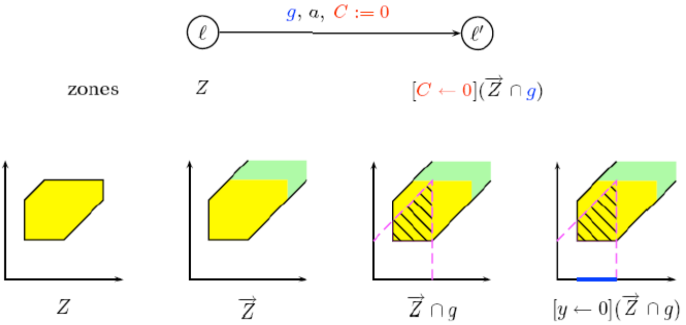
Difference Bound Matrix

$$x < 20 \wedge y \leq 20 \wedge y - x \leq 10 \wedge x - y \leq -10 \wedge z > 5$$

$$M(D) = \begin{pmatrix} (0, \leq) & (0, \leq) & (0, \leq) & (5, <) \\ (20, <) & (0, \leq) & (-10, \leq) & \infty \\ (20, \leq) & (10, \leq) & (0, \leq) & \infty \\ \infty & \infty & \infty & (0, \leq) \end{pmatrix}$$

matrix representation can be used to perform necessary operation: passing of time, resetting clock, intersection with constraint, ...

Zones: Operations



(source: J.P. Katoen)

Zone Graph: Example

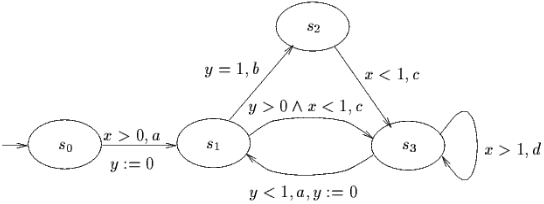


Figure 6: The automaton A_0

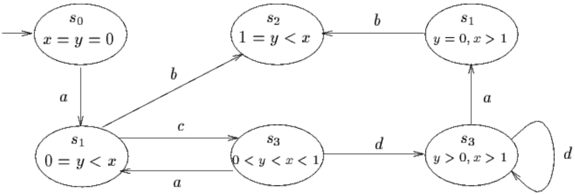


Figure 8: Reachable zone automaton

Extensions

For practical modeling we use several extensions:

- location invariants
- parallel composition of automata
- channel communication, synchronization
- integer variables

These issues are solved in the 'usual way'. Here we focused on the basic model, basic aspects dealing with time.

Example: Parallel Composition

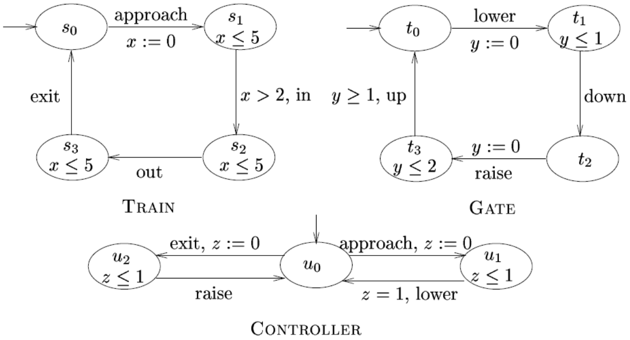


Fig. 2. Train-gate controller

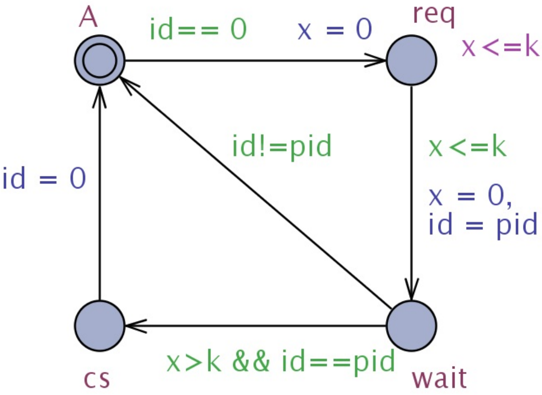
(source: R. Alur)

Fischer's Protocol

- id – shared variable, initialized -1
- assumption: known upper bound D on reading/writing variable in shared memory, for correctness it is necessary that $K > D$

```
Process i:
while (true) {
   <noncritical section>;
   while id != -1 do {}
   id := i;
   delay K;
   if (id = i) {
      <critical section>;
      id := -1;
   }
}
```

Fischer's Protocol: Model



Summary

- timed automata: formal syntax and semantics
- reachability problem, equivalence of valuations, region automaton
- practical verification: zones, extensions