General Models for Timing Analysis

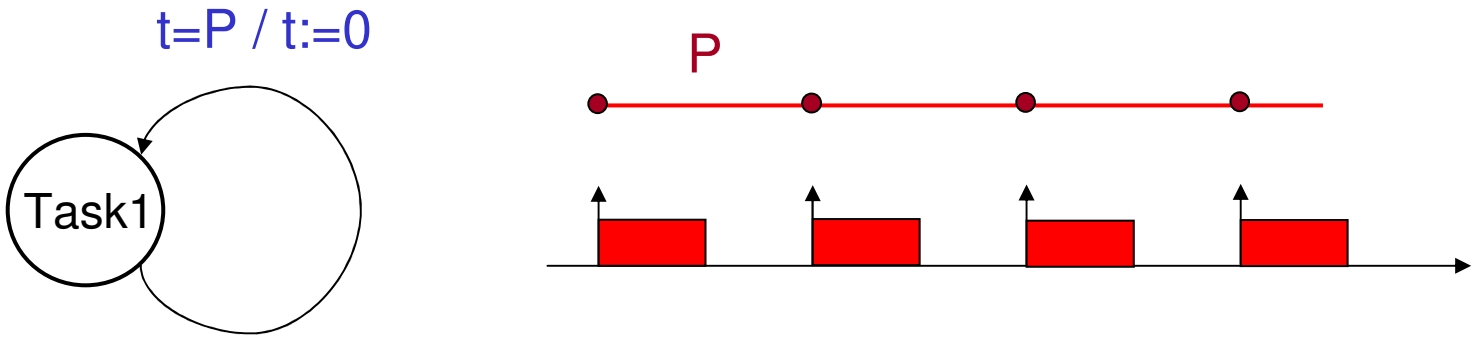
- Typical schedulability analysis setup
- Beyond the task model
- Timing Automata
- Timing Automata with Tasks

General Models for Timing Analysis

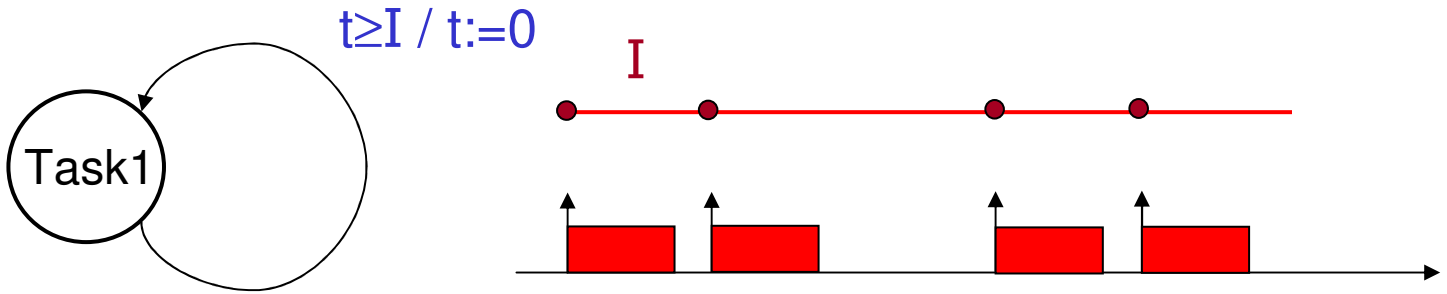
- Where can formal methods be useful
- Models: provide more sophisticated models for logical or time dependency
  - conditional activations
  - precedence constraints
- Answers: if the system is not correct formal methods provide counterexamples (which help guiding the test cases).
  - Please note: this is useful even when confidence in model data (such as the worst case computation times) is not accurate.

Models for Timing Analysis

- The very basic: independent periodic tasks …

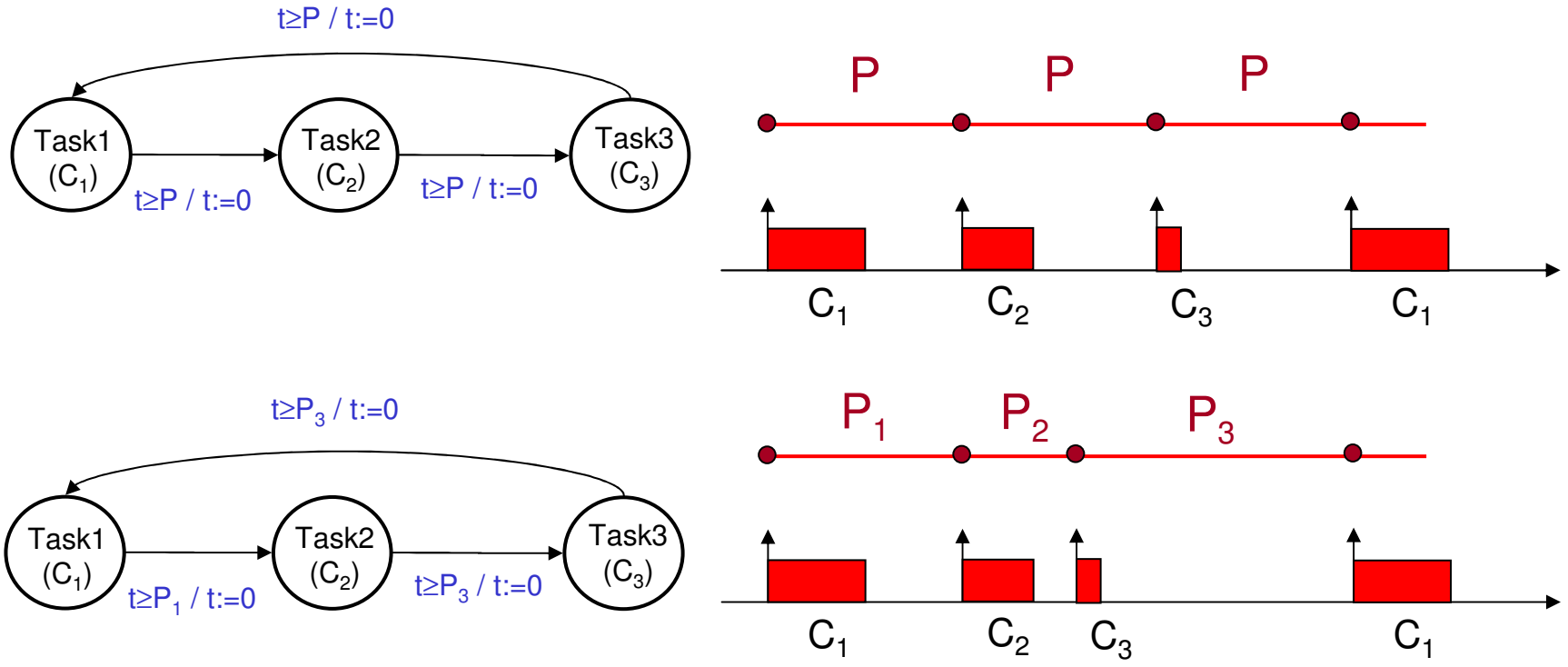


- sporadic tasks minimum interarrival time, worst case based analysis …



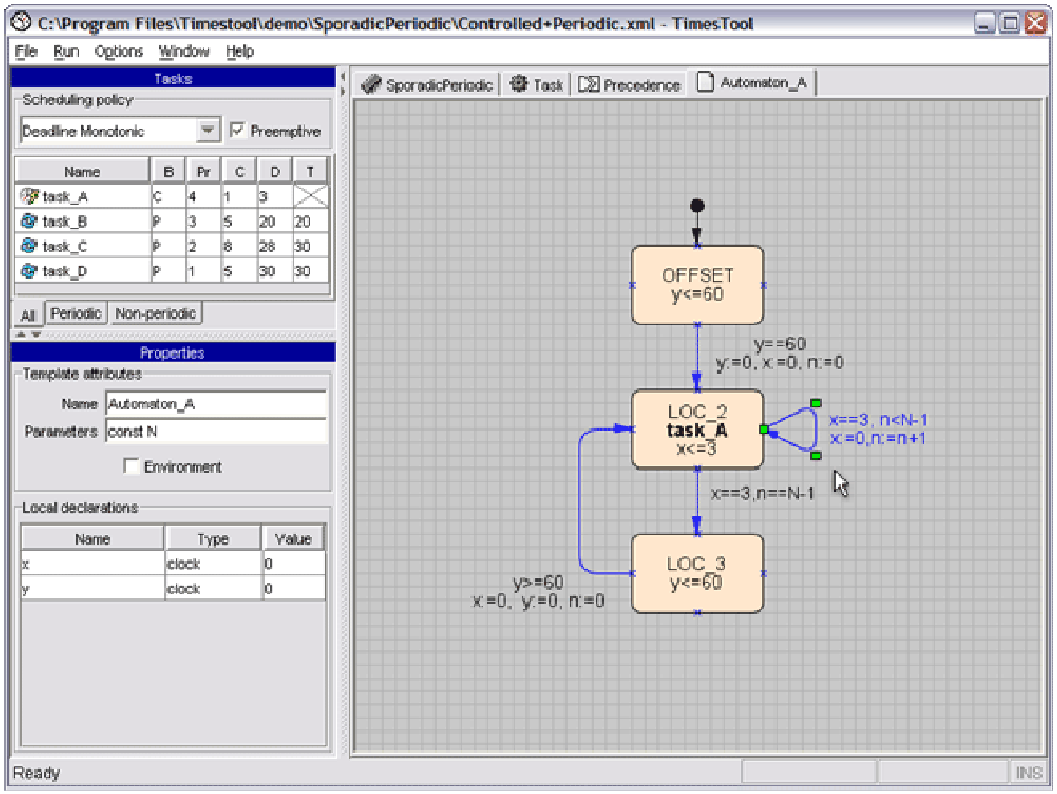
Models for Timing Analysis

- Multiframe (and Generalized Multiframe) models



Models for Timing Analysis

- but even more (almost impossible to deal with with current schedulability theory)…
- Bursty arrival patterns



Introduction to Timed Automata (and Uppaal)

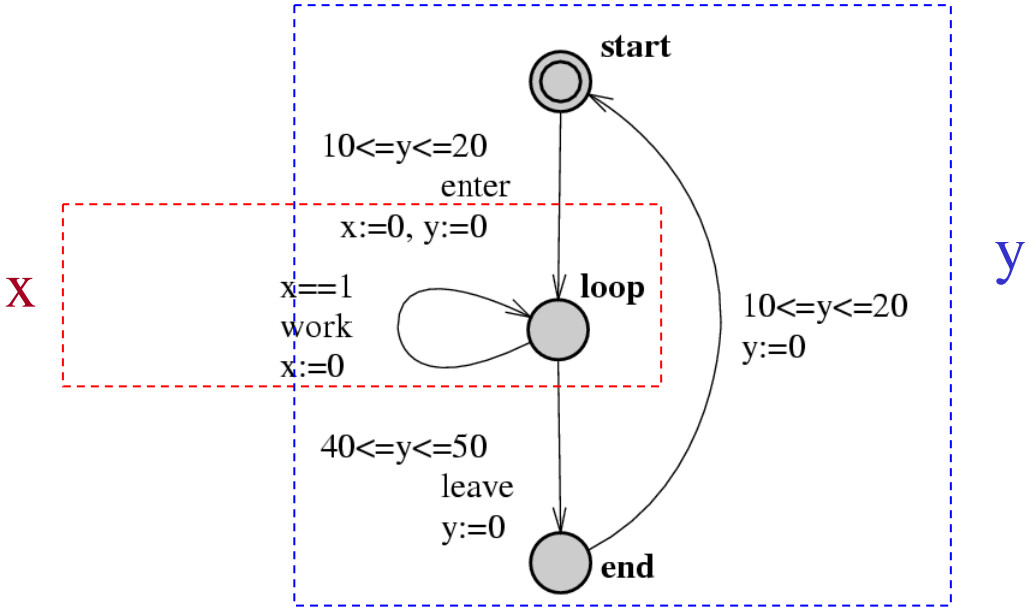
- Timed Automata
- Verification of TA

Introduction to Timed Automata

- Theory originally by Alur & Dill
- A timed automaton is a finite-state Büchi automaton extended with a set of real-valued variables modeling clocks.
  - Büchi accepting conditions are used to enforce progress properties.
  - A simplified version, namely Timed Safety Automata is introduced in [HNSY94] to specify progress properties using local invariant conditions.

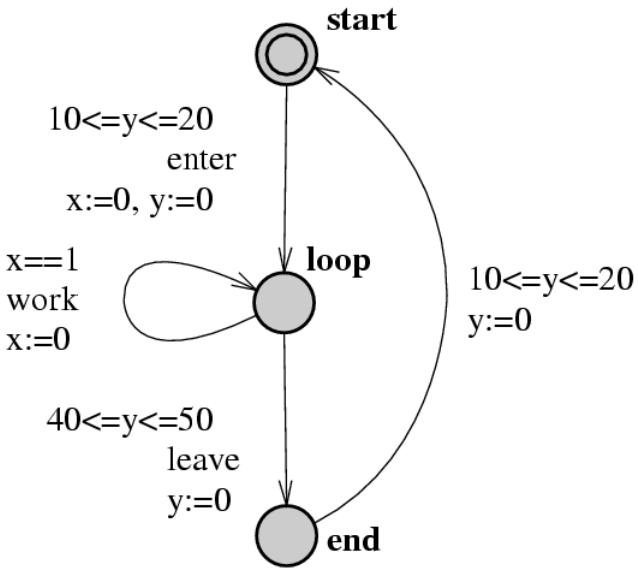
Introduction to Timed Automata

- A timed automaton is essentially a finite automaton (that is a graph containing a finite set of nodes or locations and a finite set of labeled edges) extended with real-valued variables.
- The variables model the logical clocks in the system, that are initialized with zero when the system is started, and then increase synchronously with the same rate.



Introduction to Timed Automata

- Clock constraints i.e. guards on edges are used to restrict the behavior of the automaton.
- A transition represented by an edge can be taken when the clocks values satisfy the guard labeled on the edge.
- Clocks may be reset to zero when a transition is taken.

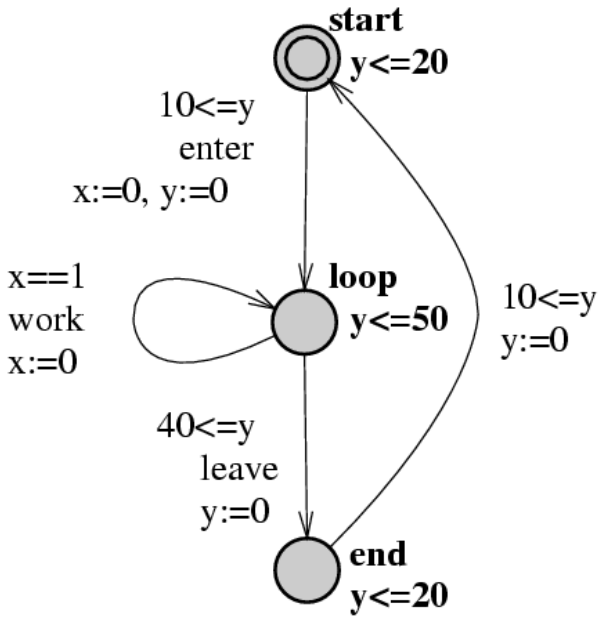


Timed Büchi Automata or Timed Safety Automata ?

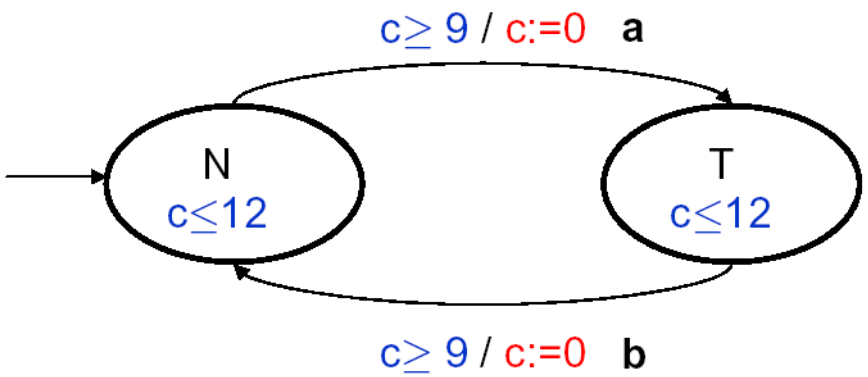
- Timed Büchi Automata A guard on an edge of an automaton is only an enabling condition of the transition represented by the edge; but it can not force the transition to be taken (the example automaton may stay forever in any location). In the initial work by Alur and Dill [AD90], the problem is solved by introducing Büchi-acceptance conditions; a subset of the locations in the automaton are marked as accepting, and only those executions passing through an accepting location infinitely often are considered valid behaviors of the automaton.
- As an example, consider the automaton in Fig. 1(a) and assume that end is marked as accepting.

Timed Büchi Automata or Timed Safety Automata ?

- Timed Safety Automata A more intuitive notion of progress is introduced in timed safety automata [HNSY94]. Instead of accepting conditions, in timed safety automata, locations may be put local timing constraints called location invariants. An automaton may remain in a location as long as the clocks values satisfy the invariant condition of the location.



Informal Syntax



Locations N, T
Initial location N
Actions a,b
Clock c

Guard c≥9
Invariant c≤12
Clock reset c:=0

Formal Syntax

- Finite set of real-valued variables C ranged over by x, y etc.standing for clocks
- Finite alphabet Σ ranged over by *a; b etc*.standing for actions.
- Clock Constraints: conjunctive formulas of atomic constraints of the form x ~ n or x - y ~ n for x, y $\in$ C; ~ $\in$ {$\leq$; <; =; >; $\geq$} and n $\in$ N. Clock constraints will be used as guards for timed automata. We use B(C) to denote the set of clock constraints.
- Definition 1 (Timed Automaton) A timed automaton A is a tuple $\langle N; l_0 ; E; I \rangle$ where
  - N is a finite set of locations (or nodes),
  - $l_0 \in$ N is the initial location,
  - $E \subseteq N \times B(C) \times \Sigma \times 2^C \times N$ is the set of edges and
  - $I : N \rightarrow B(C)$ assigns invariants to locations

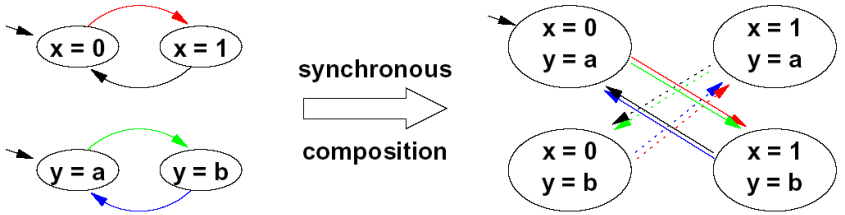
$$We\ shall\ write\ l \xrightarrow{g,a,r} l'\ when\ \langle l, g, a, r, l' \rangle \in E.$$

Formal Syntax: Concurrency and communication

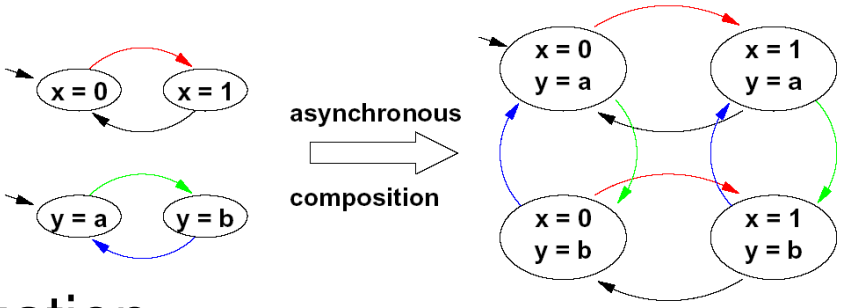
- To model concurrent systems, timed automata can be extended with parallel composition. In process algebras, various parallel composition operators have been proposed to model different aspects of concurrency (see e.g. CCS and CSP [Mil89,Hoa78]).
- In the UPPAAL modeling language [LPY97], the CCS parallel composition operator [Mil89] is used, which allows interleaving of actions as well as hand-shake synchronization.
- Essentially the parallel composition of a set of automata is the product of the automata.
- Building the product automaton is an entirely syntactical but computationally expensive operation. In UPPAAL, the product automaton is computed on-the-fly during verification.

Product Automaton

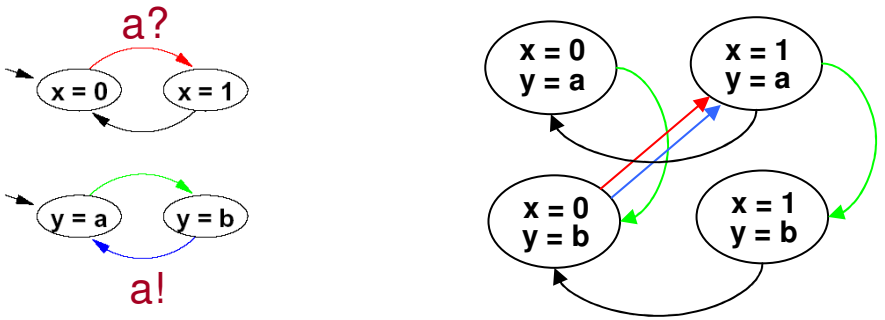
- Depends on the semantics
- Synchronous



- Asynchronous



- Explicit Synchronization



Operational Semantics

- The semantics of a timed automaton is defined as a transition system where a *state or configuration consists of the current location and the current values of clocks*. There are two types of transitions between states. The automaton may either delay for some time (a delay transition), or follow an enabled edge (an action transition).

timed passing transitions:

$(q,\nu) \rightarrow_{\delta} (q,\nu+\delta)$ if $\nu \models I(q)$ and $\nu+\delta \models I(q`)$

action transitions:

$(q,\nu) \rightarrow_{a} (q',\nu[Y:=0])$ if $(q,a,\phi,Y,q') \in E$ and $\nu \models \phi$

run: infinite sequence of transitions starting in (q_0,ν_0), where $q_0 \in Q_0$

Operational Semantics

- Clock assignment function: mapping the set of clocks C to real values $\in \Re^+$
- Timed action: a pair (t, a), where $a \in \Sigma$ is an action taken by an automaton A after $t \in R^+$ time units since A has been started.
- t is the Time stamp of the action a
- Timed trace possibly infinite sequence of timed actions $\xi=(t_1, a_1)(t_2, a_2)\ldots$ where $t_i \leq t_{i+1}$ for all $i \geq 1$.

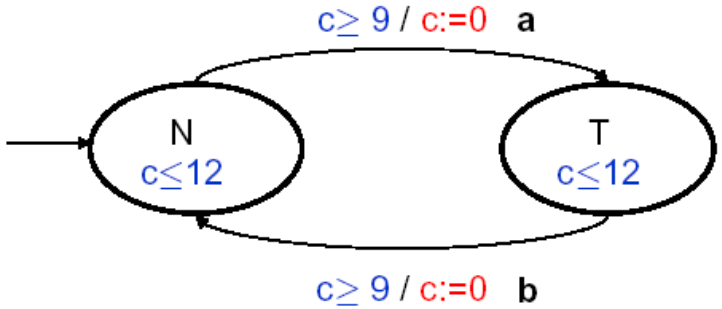
Operational Semantics

- Run: of a time automaton with initial state $<l_0, u_0>$ over a timed trace $\xi=(t_1, a_1)(t_2, a_2)\ldots$ is a sequence of transitions

$$\langle l_0, u_0\rangle \xrightarrow{d_1}\xrightarrow{a_1} \langle l_1, u_1\rangle \xrightarrow{d_2}\xrightarrow{a_2} \langle l_2, u_2\rangle \xrightarrow{d_3}\xrightarrow{a_3} \langle l_3, u_3\rangle \ldots$$

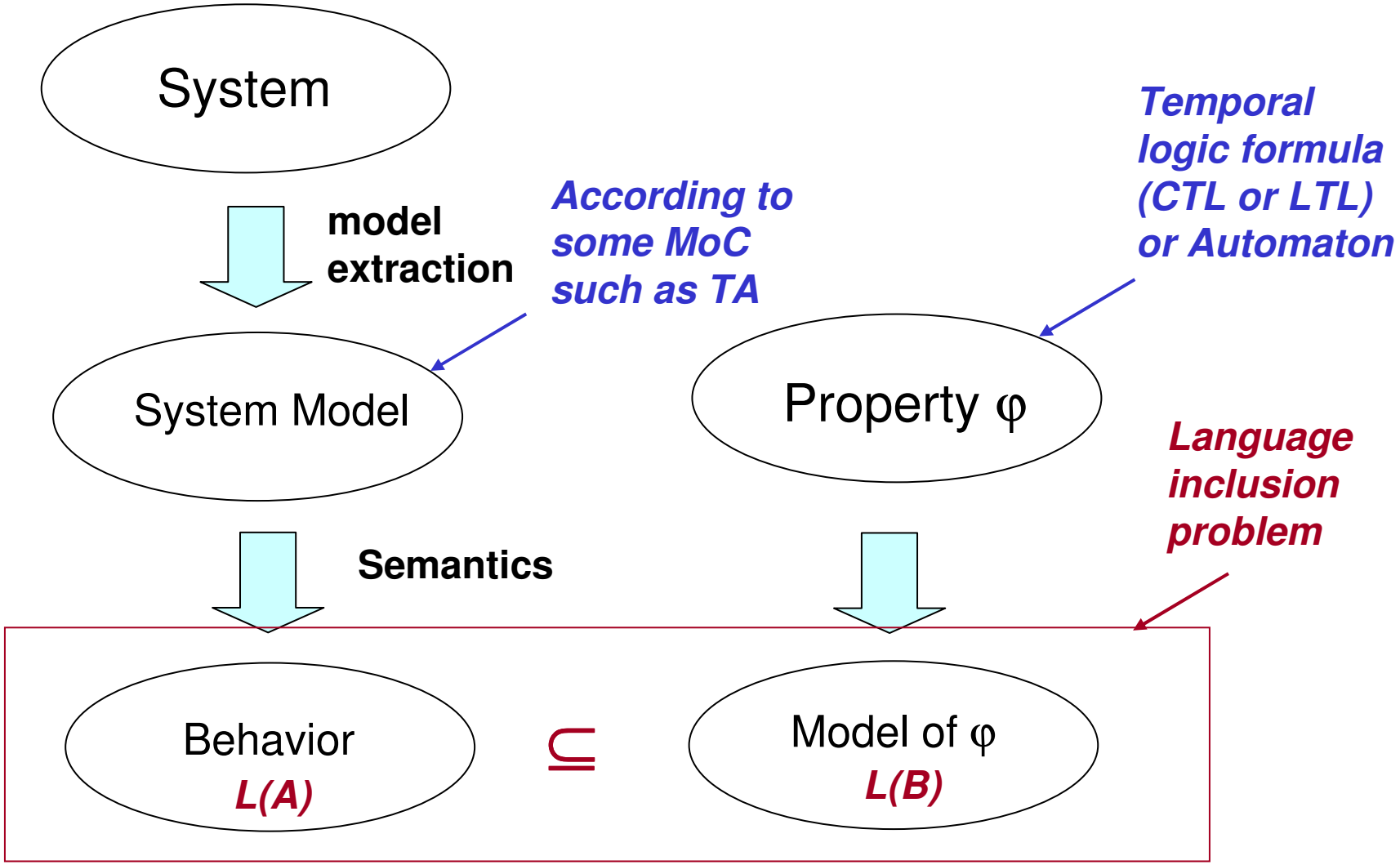
satisfying the condition $t_i = t_{i-1} + d_i$ *for all* $i \geq 1$.

The timed language $L(\mathcal{A})$ *is the set of all timed traces* ξ *for which there exists a run of* $\mathcal{A}$ *over* ξ.



Run: (N,0) $\rightarrow_{4.4}$ (N,4.4) $\rightarrow_{5.3}$ (N, 9.7) $\rightarrow_a$ (T,0)
$\rightarrow_{9.0}$ (T,9.0) $\rightarrow_b$ (N,0) $\rightarrow_{11.8}$ (N,11.8) $\rightarrow$ ....

Verification Setting

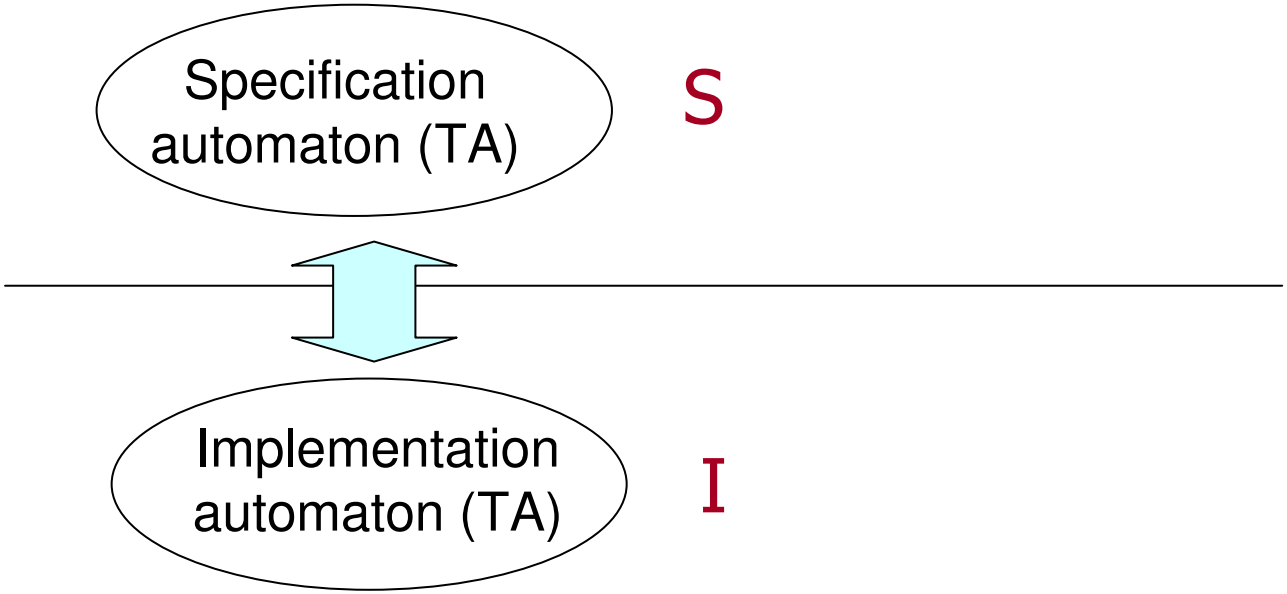


Verification Setting

- Checking a system implementation **I** against a specification of a property **P** in case both are expressed in terms of automata (*homogeneous verification*).
- The implementation automaton **AI** is composed with the complementary automaton ¬**AP** expressing the negation of the desired property.
- The implementation **I** violates the specification property if the product automata **AI** || ¬**AP** has some possible run and it is verified is the composition has no runs.

Bisimulation

- Typical setting: verification of implementation



- Problem: is the automaton I a “correct” implementation of the specification?

Verification and Bisimulation

- Decidability
- Bisimulation
- Reachability
- Decidability of problems

Operational Semantics

- Decidability
- (*for the general class of nondeterministic timed automata*)
- The language inclusion checking problem i.e. to check $L(A) \subseteq L(B)$ is undecidable
- Timed automata are not determinizable in general
- Timed automata cannot be complemented, that is, the complement of the timed language of a timed automaton may not be described as a timed automaton.
- *But …*
- The language inclusion problem is decidable if B in $L(A) \subseteq L(B)$ is restricted to the class of deterministic TA

Operational Semantics

- Bisimulation (Decidability of timed bisimulation)

Definition 4 *A bisimulation R over the states of timed transition systems and the alphabet $\Sigma \cup \mathbb{R}_+$, is a symmetrical binary relation satisfying the following condition:*

for all $(s_1, s_2) \in R$, if $s_1 \xrightarrow{\sigma} s_1'$ for some $\sigma \in \Sigma \cup \mathbb{R}_+$ and s_1', then $s_2 \xrightarrow{\sigma} s_2'$ and $(s_1', s_2') \in R$ for some s_2'.

- Two automata are bisimilar iff there is a bisimulation containing the initial states of the automata.
- Intuitively, two automata are timed bisimilar iff they perform the same action transition at the same time and reach bisimilar states.
- Timed bisimulation is decidable.

Operational Semantics

- Untimed Bisimulation
- Abstracting from timing information, we can establish bisimulation between actions only.
- Define s $\rightarrow\in$ s' if s $\rightarrow d$ s' for some real number d.
- Untimed bisimulation is defined by replacing the alphabet with $\Sigma \cup \{\in\}$ in the previous definition.

Operational Semantics

- Reachability
- Question is reachability of a given final state or a set of final states.
- It may be used to characterize safety properties of a system.
- Invariant properties may be specified and checked using the negation of reachability properties
- Bounded liveness properties may be specified using clock constraints in combination with local properties on locations
- The reachability problem is decidable

Operational Semantics

- The reachability problem is decidable
- *Apparently the transition system of a timed automaton is infinite, given the real-valued clocks.*
- Necessary step: the concept of region equivalence (exploiting bisimulation properties)

Symbolic Semantics & Verification

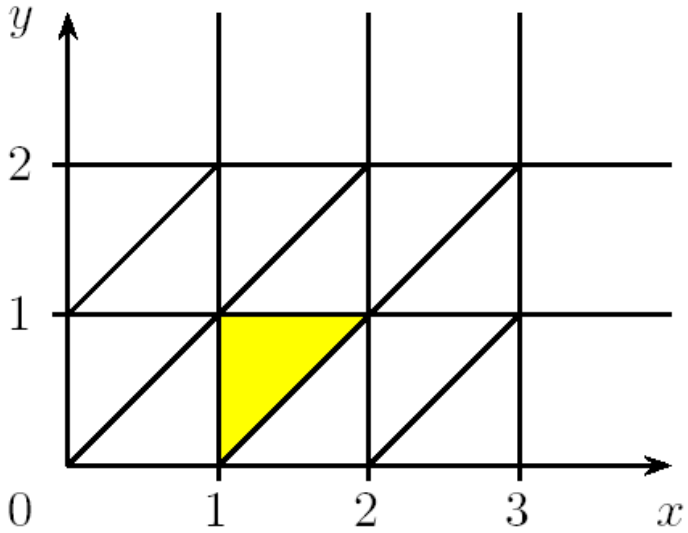
- Regions and Zones
- The foundation for the decidability results in timed automata is based on the notion of region equivalence over clock assignments

Definition 6 (Region Equivalence) *Let k be a function, called a* **clock ceiling**, *mapping each clock $x \in \mathcal{C}$ to a natural number $k(x)$ (*i.e. *the ceiling of x). For a real number d, let $\{d\}$ denote the fractional part of d, and $\lfloor d \rfloor$ denote its integer part. Two clock assignments u, v are* **region-equivalent**, *denoted $u \stackrel{\cdot}{\sim}_k v$, iff*

1. *for all x, either $\lfloor u(x) \rfloor = \lfloor v(x) \rfloor$ or both $u(x) > k(x)$ and $v(x) > k(x)$,*
2. *for all x, if $u(x) \leq k(x)$ then $\{u(x)\} = 0$ iff $\{v(x)\} = 0$ and*
3. *for all x, y if $u(x) \leq k(x)$ and $u(y) \leq k(y)$ then $\{u(x)\} \leq \{u(y)\}$ iff $\{v(x)\} \leq \{v(y)\}$*

- Regions
- Basis for a finite partitioning of the state space (useful for proving theorems rather than in practice)

Symbolic Semantics & Verification



Equivalence of finite index

region defined by

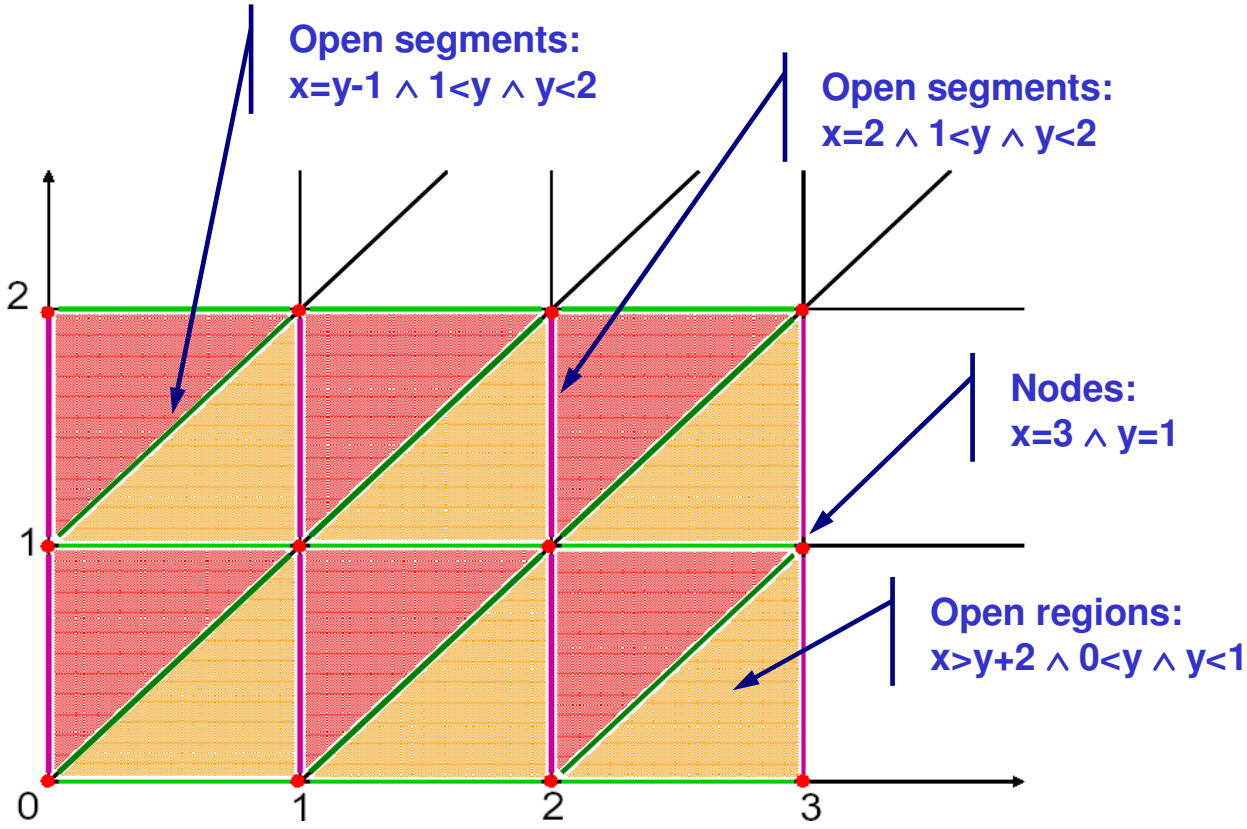
$I_x =]1;2[, I_y =]0;1[$

$\{x\} < \{y\}$

Symbolic Semantics & Verification

- Regions
- u~v implies (l,u) and (l,v) are bisimilar wrt untimed bisimulation

the number of regions is exponential in the number of clocks and the maximal constants in the guards (the figure has 60 regions!)



Symbolic Semantics & Verification

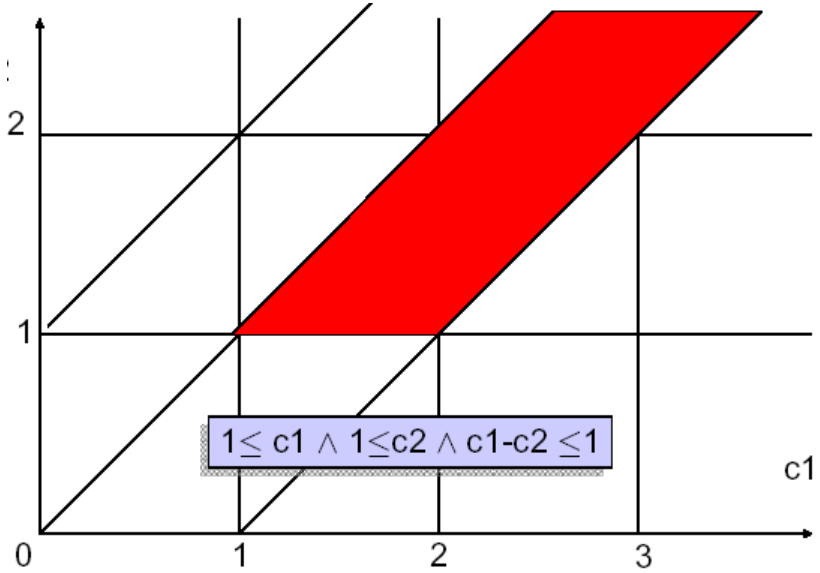
- Region graph or Region automaton (of the original timed automaton)
- With symbolic states (regions) and transitions among them.

  - $\langle l, [u] \rangle \Rightarrow \langle l, [v] \rangle$ if $\langle l, u \rangle \xrightarrow{d} \langle l, v \rangle$ for a positive real number d and
  - $\langle l, [u] \rangle \Rightarrow \langle l', [v] \rangle$ if $\langle l, u \rangle \xrightarrow{a} \langle l', v \rangle$ for an action a.

Note that the transition relation $\Rightarrow$ is finite. Thus the region graph for a timed automaton is finite. Several verification problems such as reachability analysis, untimed language inclusion, language emptiness [AD94] as well as timed bisimulation [Cer92] can be solved by techniques based on the region construction.

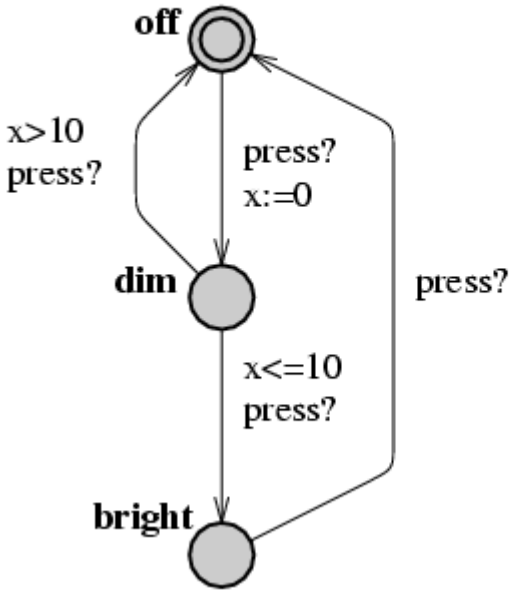
Symbolic Semantics & Verification

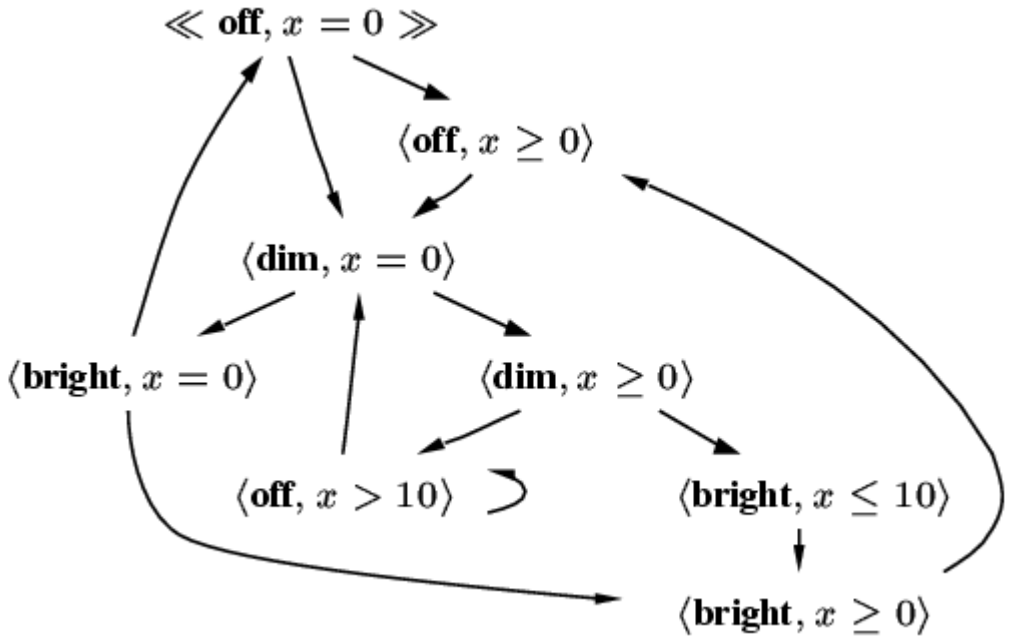
- More efficient encoding: Zones and Zone graph
- consider only “relevant” clock constraints
- zones as unions of regions (solutions of clock constraints)
- Zones are convex polyhedra



Symbolic Semantics & Verification

- An example
- Only 8 zones are necessary (instead of more than 50 regions)





Symbolic Semantics & Verification

- Representing zones
- a zone is the solution set of a clock constraint, that is the maximal set of clock assignments satisfying the constraint. It is well-known that such sets can be efficiently represented and stored in memory as DBMs (Difference Bound Matrices) [Bel57]. For a clock constraint D, let [D] denote the maximal set of clock assignments satisfying D.

Symbolic Semantics & Verification

- Symbolic state representation based on zones
- a symbolic state is a pair <l, D> (l is a location, D is a zone) representing a set of states of the automaton. A symbolic transition describes all the possible concrete transitions from the set of states.
- Definition
- Let D be a zone and r a set of clocks.
- $D\uparrow = \{u+d \mid u \in D, d \in \Re^+\}$
- $r(D) = \{[r \rightarrow 0]u \mid u \in D\}$
- Let $\rightsquigarrow$ denote the symbolic transition relation over symbolic states defined by the rules
- $<l,D> \rightsquigarrow <l,D\uparrow \wedge I(l)>$
- $<l,D> \rightsquigarrow <l',r(D \wedge g) \wedge I(l')>$ if $l \xrightarrow{g,a,r} l'$

Symbolic Semantics & Verification

- The set of zones is closed under the operations $D^{\uparrow}$ and r(D)
- Correspondence between symbolic semantics and operational semantics:
- $\langle l,D\rangle \rightsquigarrow \langle l',D'\rangle$ implies that for all $u' \in D'$, $\langle l,u\rangle \rightarrow \langle l',u'\rangle$ for some $u \in D$.

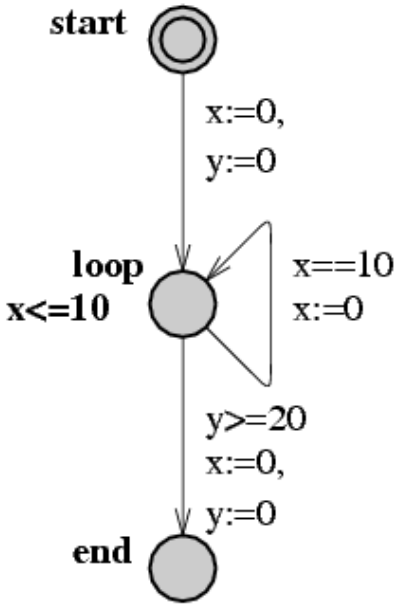
Theorem 1 *Assume a timed automaton with initial state* $\langle l_0, u_0\rangle$.

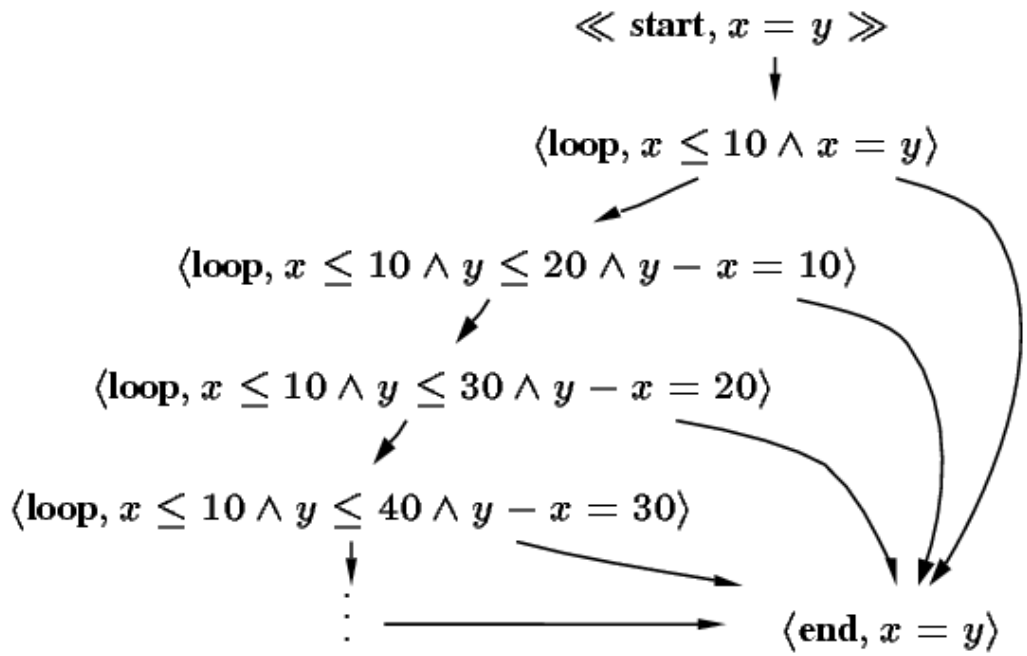
1. *(soundness)* $\langle l_0, \{u_0\}\rangle \rightsquigarrow^* \langle l_f, D_f\rangle$ *implies* $\langle l_0, u_0\rangle \rightarrow^* \langle l_f, u_f\rangle$ *for all* $u_f \in D_f$.
2. *(Completeness)* $\langle l_0, u_0\rangle \rightarrow^* \langle l_f, u_f\rangle$ *implies* $\langle l_0, \{u_0\}\rangle \rightsquigarrow^* \langle l_f, D_f\rangle$ *for some* D_f *such that* $u_f \in D_f$

- Unfortunately, the relation $\rightsquigarrow$ is infinite and the zone graph of a timed automaton may be infinite.

Symbolic Semantics & Verification

- An infinite zone graph





Symbolic Semantics & Verification

- Solution: k-normalization
- Intuitively: if the value of a clock is larger than the maximal clock constant in the automaton, it is no longer important to know the exact value, but only the fact that it is above the constant ...

Definition 8 (**k-Normalization**) *Let D be a zone and k a clock ceiling. The semantics of the k-normalization operation on zones is defined as follows:*

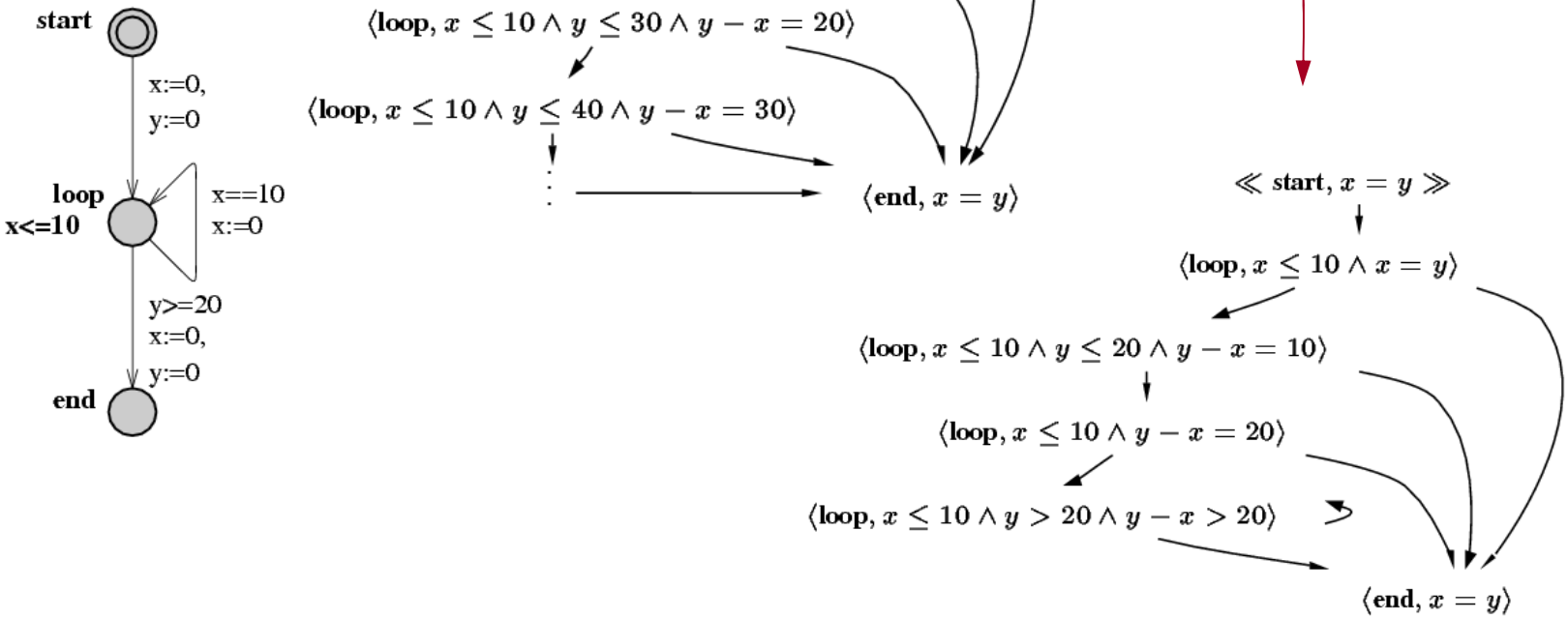
$$norm_k(D) = \{u | u \dot{\sim}_k v, v \in D\}$$

Note that the normalization operation is indexed by a clock ceiling k. According to [Rok93,Pet99], $\mathsf{norm}_k(D)$ can be computed from the canonical representation of D by

1. removing all constraints of the form $x < m$, $x \leq m$, $x - y < m$ and $x - y \leq m$ where $m > k(x)$,
2. replacing all constraints of the form $x > m$, $x \geq m$, $x - y > m$ and $x - y \geq m$ where $m > k(x)$ with $x > k(x)$ and $x - y > k(x)$ respectively.

Symbolic Semantics & Verification

- Normalizing the previous example



Symbolic Semantics & Verification

- For automata without difference constraints in the form x-y ~ n (diagonal free automata), the symbolic set <l,D> and the transition $\rightsquigarrow_k$ resulting from k-normalization are sound and complete and the transition relation is finite.
- This is not true for non diagonal-free TA

Symbolic Semantics & Verification

- Counterexample ...

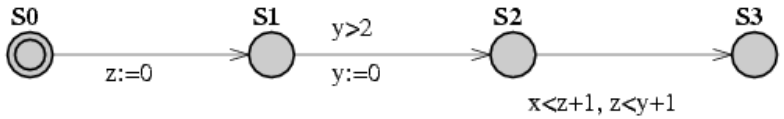


Fig. 6. A counter example

$$S_0 : \begin{cases} x - y = 0 \\ y - z = 0 \\ z - x = 0 \end{cases}$$
$$S_1 : \begin{cases} x - y = 0 \\ z - x \leq 0 \\ z - y \leq 0 \end{cases}$$
$$S_2 : \begin{cases} y - x < -2 \\ y - z \leq 0 \\ z - x \leq 0 \\ \mathbf{0} - x < -2 \end{cases}$$

$$S_0 : \begin{cases} x - y = 0 \\ y - z = 0 \\ z - x = 0 \end{cases}$$
$$S_1 : \begin{cases} x - y = 0 \\ z - x \leq 0 \\ z - y \leq 0 \end{cases}$$
$$S_2 : \begin{cases} y - x < -1 \\ y - z \leq 0 \\ z - x \leq 0 \\ \mathbf{0} - x < -1 \end{cases}$$
$$S_3 : \begin{cases} y - x < -1 \\ y - z < 0 \\ z - x < 0 \\ \mathbf{0} - x < -1 \\ \mathbf{0} - z < 0 \\ x - z < 1 \end{cases}$$

the normalized zone definition for S2 incorrectly allows transition to S3

the non normalized zone definition for S2 does not allow transition to S3

Symbolic Semantics & Verification

- A new definition of k normalization is needed (more complex, may lead to partitioning of symbolic states)

Definition 10 (Normalization Using Difference Constraints) *Let $\mathcal{G}$ stand for a finite set of difference constraints of the form $x-y \sim n$ for $x, y \in C$, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$, and k for a clock ceiling. Two clock assignments u, v are equivalent, denoted $u \dot{\sim}_{k,\mathcal{G}} v$ if the following holds:*

- *$u \dot{\sim}_k v$ and*
- *for all $g \in \mathcal{G}$, $u \in g$ iff $v \in g$.*

The semantics of the refined k-normalization operation on zones is defined as follows:

$$norm_{k,\mathcal{G}}(D) = \{u | u \ \dot{\sim}_{k,\mathcal{G}} \ v, v \in D\}$$

- The new definition is sound, complete and the transition relation is finite

Symbolic Semantics & Verification

- Problems:
- Representing and storing the zones
  - is there a “canonical” zone representation ?
- Operations on zones
- Computing the zones (forward and backward analysis)

DBMs

- A clock constraint over C is a conjunction of atomic constraints of the form $x \sim m \geq m$ and $x - y \sim n$ where $x, y \in C$, $\sim \in \{\leq, <, =, >, \geq\}$ and $m,n \in N$. A zone denoted by a clock constraint D is the maximal set of clock assignments satisfying D.
- introduce a reference clock **0** with the constant value 0. Let $C_0 = C \cup \{\mathbf{0}\}$. Then any zone $D \in B(C)$ can be rewritten as a conjunction of constraints of the form $x - y < n$ or $x - y \leq n$ for $x, y \in C\ 0$, and $n \in Z$.
  - If the rewritten zone has two constraints on the same pair of variables only the intersection of the two is significant.
- A zone can be represented using at most $|C_0|^2$ atomic constraints of the form $x - y \leq n$ such that each pair of clocks x y is mentioned only once.
- The $C_0 x C_0$ matrix of these elements is the DBM matrix

DBMs

- Example, consider the zone
  D = x-**0** < 20 ∧ y-**0**≤20 ∧ y-x≤10 ∧ x-y ≤-10 ∧ **0**-z < 5.
- If the clocks are ordered **0**,x,y,z

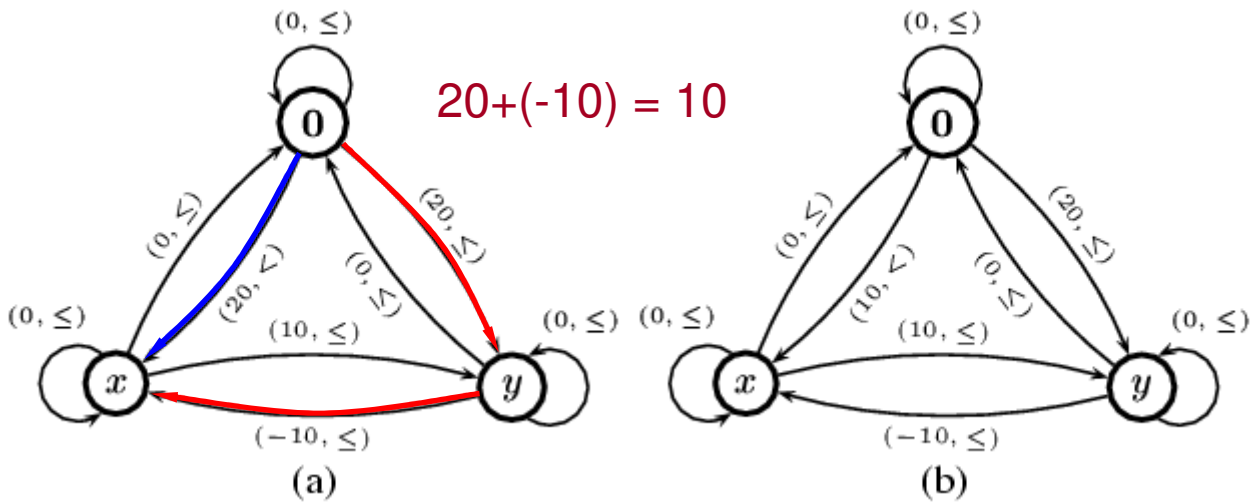
$$M(D) = \begin{pmatrix} (0,\leq) & (0,\leq) & (0,\leq) & (5,<) \\ (20,<) & (0,\leq) & (-10,\leq) & \infty \\ (20,\leq) & (10,\leq) & (0,\leq) & \infty \\ \infty & \infty & \infty & (0,\leq) \end{pmatrix}$$

DBMs

- Canonical DBMs
- Usually there are an infinite number of zones sharing the same solution set, but there is a unique representation where no atomic constraint can be strengthened without losing solutions.
- To compute the canonical form of a given zone we need to derive the tightest constraint on each clock difference.
- To solve this problem, we use a weighted graph interpretation of zones where the clocks are nodes and the atomic constraints are edges labeled with bounds.
- A constraint in the form of $x-y \leq n$ will be converted to an edge from node y to node x labeled with $(n; \leq)$, namely the distance from y to x is bounded by n.

DBMs

- Canonical DBMs: an example
- consider the zone x-**0**<20∧y-**0**≤20∧y-x≤10∧x-y≤-10
- by combining y-**0**≤20 and x-y≤-10 → x-**0**≤10



DBMs

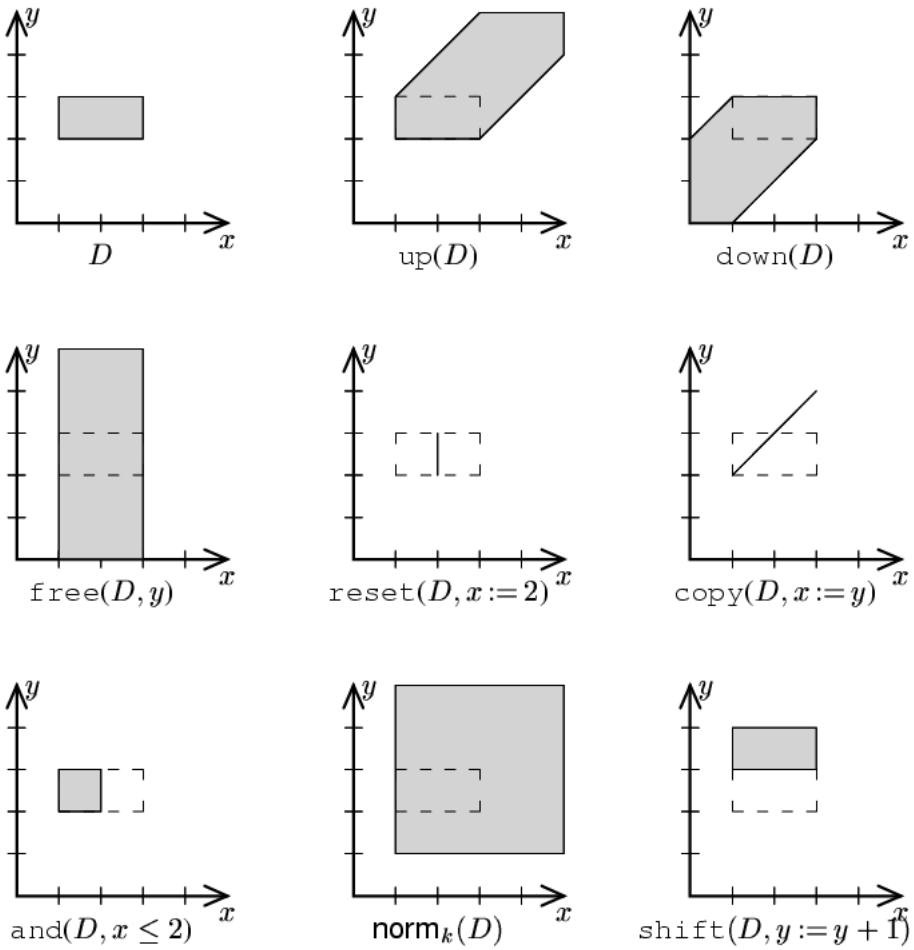
- Canonical DBMs: an example
- Thus, deriving the tightest constraint on a pair of clocks in a zone is equivalent to finding the shortest path between their nodes in the graph interpretation.
- A canonical version of a zone can be computed using a shortest path Floyd-Warshall algorithm.
- This algorithm is quite expensive (cubic in the number of clocks), hence it is desirable to produce DBM that are already in canonical form (as the result of performing an operation on a canonical zone).

DBMs: Practical issues

- Canonical DBMs: removing redundant constraints
- For example, in a zone containing constraints $x-y<2$, $y-z<5$ and $x-z<7$ the last constraint is redundant.
- From [LLPY97] it is known that for each zone there is a minimal constraint system with the same solution set.
- By computing this minimal form for all zones it is possible to reduce memory consumption

DBMs: Practical issues

- Operations on Zones



DBMs: Practical issues

- Classes: property checking, transformation and zone-normalization
- Property checking
  - `consistent(D)`
  - `relation(D,D')`
  - `satisfied(D, `$x_i - x_j \leq m$`)`
- Transformation
  - `up(D)`
  - `down(D)`
  - `and(D, `$x_i - y_j \leq b$`)`
  - `free(D,x)`
  - `reset(D,x:=m)`
  - `copy(D,x:=y)`
  - `shift(D,x:=x+m)`
- Transformation
  - $\mathtt{norm}_k(D)$
  - $\mathtt{norm}_{k,G}(D)$

DBMs: Practical issues

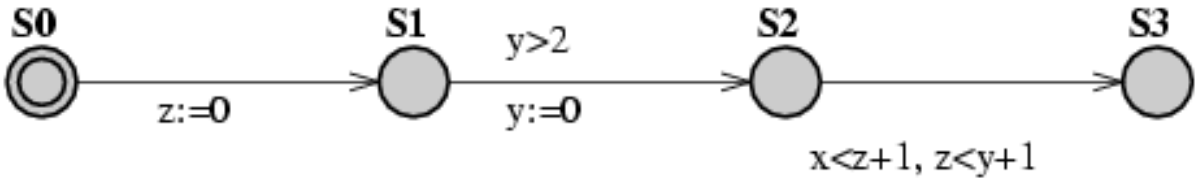
- Example of pseudocode implementing `and(D, x_i − y_j ≤ b)`

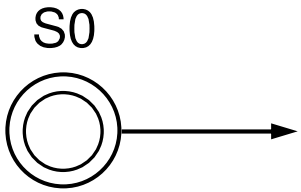
Algorithm 8 `and`(D, g)

```
if D_yx + (m, ≺) < 0 then
    D_00 = (−1, ≤)
else if (m, ≺) < D_xy then
    D_xy = (m, ≺)
    for i := 0 to n do
        for j := 0 to n do
            if D_ix + D_xj < D_ij then
                D_ij = D_ix + D_xj
            end if
            if D_iy + D_yj < D_ij then
                D_ij = D_iy + D_yj
            end if
        end for
    end for
end if
```

DBMs: Practical issues

- starting state
  - all clock variables begin with value=0 and all advance at the same rate

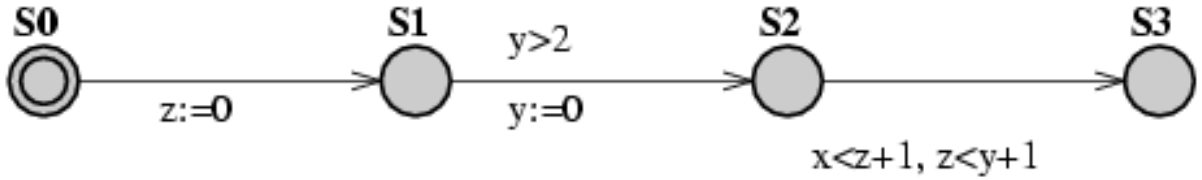


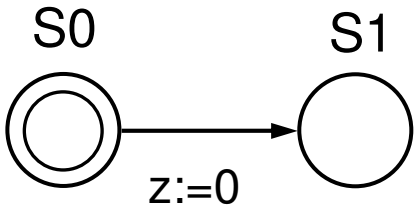


$$\begin{matrix}\text{reset(x, 0)}\\ \text{reset(y, 0)}\\ \text{reset(z, 0)}\end{matrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad \begin{matrix}\text{free(D,x)}\\ \text{free(D,y)}\\ \text{free(D,z)}\end{matrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ \infty & 0 & 0 & 0 \\ \infty & 0 & 0 & 0 \\ \infty & 0 & 0 & 0 \end{pmatrix}$$

DBMs: Practical issues

- starting state
  - forward analysis





$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ \infty & 0 & 0 & 0 \\ \infty & 0 & 0 & 0 \\ \infty & 0 & 0 & 0 \end{pmatrix} \text{reset(D, z:=0)} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ \infty & 0 & 0 & \infty \\ \infty & 0 & 0 & \infty \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} \text{free(D,x)} \\ \text{free(D,y)} \\ \text{free(D,z)} \end{matrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ \infty & 0 & 0 & \infty \\ \infty & 0 & 0 & \infty \\ \infty & 0 & 0 & 0 \end{pmatrix}$$

Symbolic Reachability Analysis

- Model checking of two fundamental properties:
  - liveness
  - safety
- Safety properties can be checked using reachability analysis

- Reachability analysis consists of two basic steps:
  - computing the state space of the automaton
  - searching for states that satisfy or contradict given properties
- The first step can be performed prior to the search or on-the-fly during the search process.
- On-the-fly methods have the advantage that only the part of the state space that is required to prove the given property is generated
- but on-the-fly methods will generate the entire state space for proving invariants (and other properties)

Symbolic Reachability Analysis

- Here is the core of the verification engine of Uppaal

Algorithm 1 Reachability analysis

PASSED $= \emptyset$, WAIT $= \{\langle l_0, D_0 \rangle\}$
while WAIT $\neq \emptyset$ **do**
  take $\langle l, D \rangle$ from WAIT
  if $l = l_f \wedge D \cap \phi_f \neq \emptyset$ **then return** "YES"
  if $D \not\subseteq D'$ for all $\langle l, D' \rangle \in$ PASSED **then**
    add $\langle l, D \rangle$ to PASSED
    for all $\langle l', D' \rangle$ such that $\langle l, D \rangle \rightsquigarrow k, \mathcal{G}\langle l', D' \rangle$ **do**
      add $\langle l', D' \rangle$ to WAIT
    end for
  end if
end while
return "NO"

Symbolic Reachability Analysis

- Assume a timed automaton A with a set of initial states and a set of final states (e.g. the bad states) characterized as $<l_0,D_0>$ and $<l_f, \phi_f>$ respectively.
- Assume that k is the clock ceiling defined by the maximal constants appearing in A and ϕ_f, and G denotes the set of difference constraints appearing in A and ϕ_f. Algorithm 1 can be used to
- check if the initial states may evolve to any state whose location is l_f and whose clock assignment satisfies ϕ_f. It computes the normalized zone-graph of the automaton on-the-fly, in search for symbolic states containing location l_f and zones intersecting with ϕ_f.
- The algorithm computes the transitive closure of $\rightarrow_{k,G}$ step by step, and at each step, checks if the reached zones intersect with ϕ_f. From Theorem 2, it follows that the algorithm will return with a correct answer. It is also guaranteed to terminate because $\rightarrow_{k,G}$ is finite.

Symbolic Reachability Analysis

- As mentioned earlier, for a given timed automaton with a fixed set of clocks whose maximal constants are bounded by a clock ceiling k, and a fixed set of diagonal constraints contained in the guards, the number of all possible normalized zones is bounded because a normalized zone can not contain arbitrarily large or arbitrarily small constants. In fact the smallest possible zones are the refined regions. Thus the whole state-space of a timed automaton can only be partitioned into finitely many symbolic states and the worst case is the size of the region graph of the automaton, induced by the refined region equivalence. Therefore, the algorithm is working on a finite structure and it will terminate.

Symbolic Reachability Analysis

- Algorithm 1 also highlights some of the issues in developing a model-checker for timed automata. Firstly, the representation and manipulation of states, primarily zones, is crucial to the performance of a model-checker. Note that in addition to the operations to compute the successors of a zone according to $\rightarrow_{k,G}$, the algorithm uses two more operations to check the emptiness of a zone as well as the inclusion between two zones. Thus, designing efficient algorithms and data-structures for zones is a major issue in developing a verification tool for timed automata. Secondly, PASSED holds all encountered states and its size puts a limit on the size of systems we can verify. This raises the research challenges e.g. state compression [Ben02], state-space reduction [BJLY98] and approximate techniques [Bal96].