Timed Automata

Rajeev Alur

University of Pennsylvania
www.cis.upenn.edu/~alur/

SFM-RT, Bertinoro, Sept 2004

Advantages

Automated formal verification, Effective debugging tool

Moderate industrial success

In-house groups: Intel, Microsoft, Lucent, Motorola…

Commercial model checkers: FormalCheck by Cadence

Obstacles

Scalability is still a problem (about 500 state vars)

Effective use requires great expertise

Still, a great success story for CS theory impacting practice, and a vibrant area of research

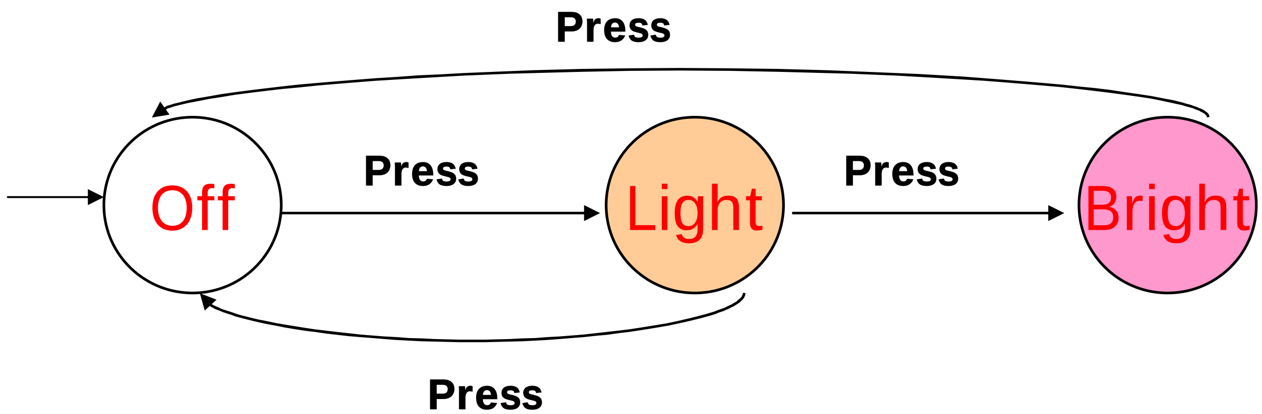
Automata in Model Checking

- Automata Theory provides foundations for model checking
  - Automata / state machines to model components
  - Intersection, projection model operations
  - Verification is inclusion: is System contained in Spec?
- Classical: Finite-state automata (regular languages)
  - Pushdown automata
  - Counter automata
  - Probabilistic automata ....
- Timed automata as a foundation for real-time systems (automata + timing constraints

Course Overview

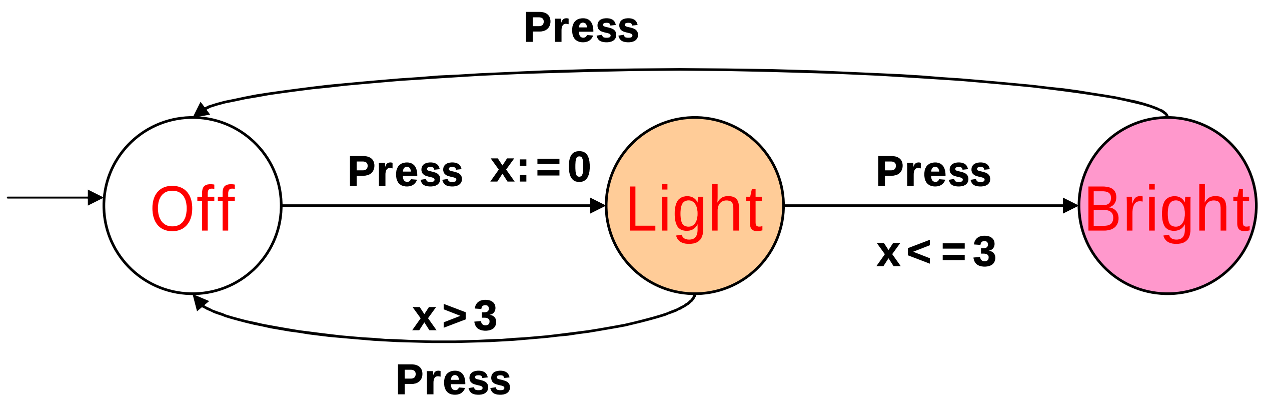
- Timed Automata Model
- Reachability
  - Preliminaries: Transition Systems and Equivalences
  - Region Graph Construction
  - Decidability Boundary
- Timed Regular Languages
  - Closure Properties and Complementation
  - Deterministic and Two-way Automata
  - Robustness
  - Inclusion

Simple Light Control



WANT: if press is issued twice quickly then the light will get brighter; otherwise the light is turned off.

Simple Light Control

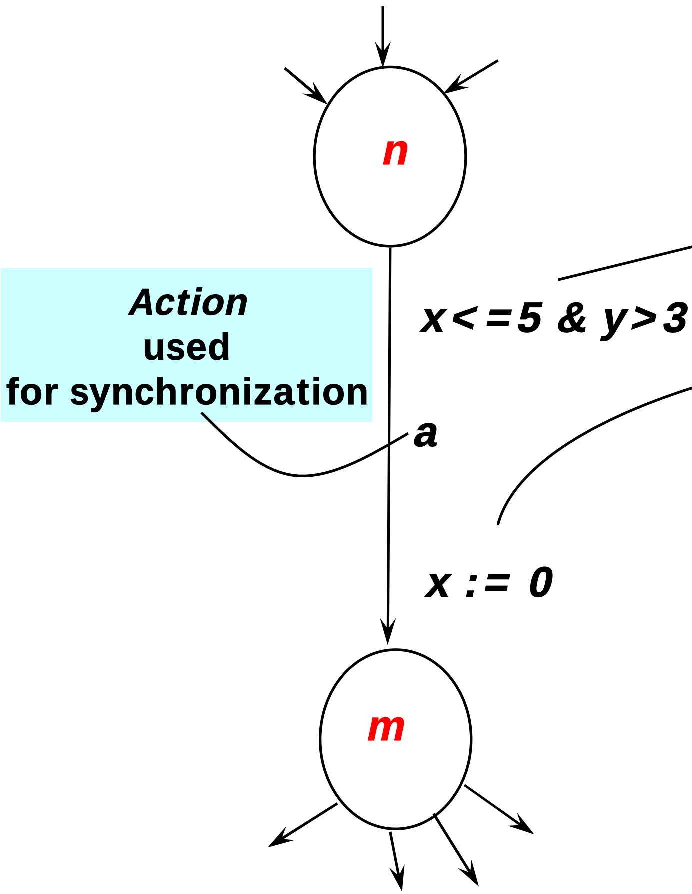


Solution: Add a real-valued clock **x**

Adding continuous variables to state machines

Timed Automata

Clocks: ***x, y***



Guard
Boolean combination of comparisons with Integer/rational bounds

Reset
Action performed on clocks

State

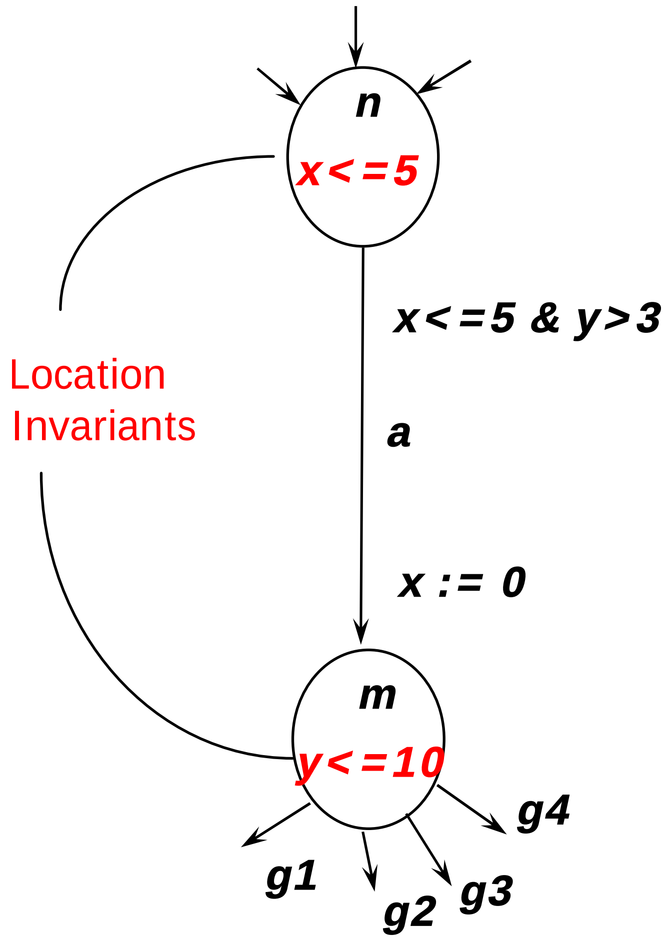
(***location*** , ***x***=v , ***y***=u)   where v,u are in **R**

Transitions

(***n*** , ***x***=2.4 , ***y***=3.1415) $\xrightarrow{a}$ (***m*** , ***x***=0 , ***y***=3.1415)

(***n*** , ***x***=2.4 , ***y***=3.1415) $\xrightarrow{wait(1.1)}$ (***n*** , ***x***=3.5 , ***y***=4.2415)

Adding Invariants



Clocks: ***x, y***

Transitions

(***n*** , ***x***=2.4 , ***y***=3.1415) $\xrightarrow{\cancel{wait(3.2)}}$

(***n*** , ***x***=2.4 , ***y***=3.1415) $\xrightarrow{wait(1.1)}$ (***n*** , ***x***=3.5 , ***y***=4.2415)

Invariants ensure progress!!

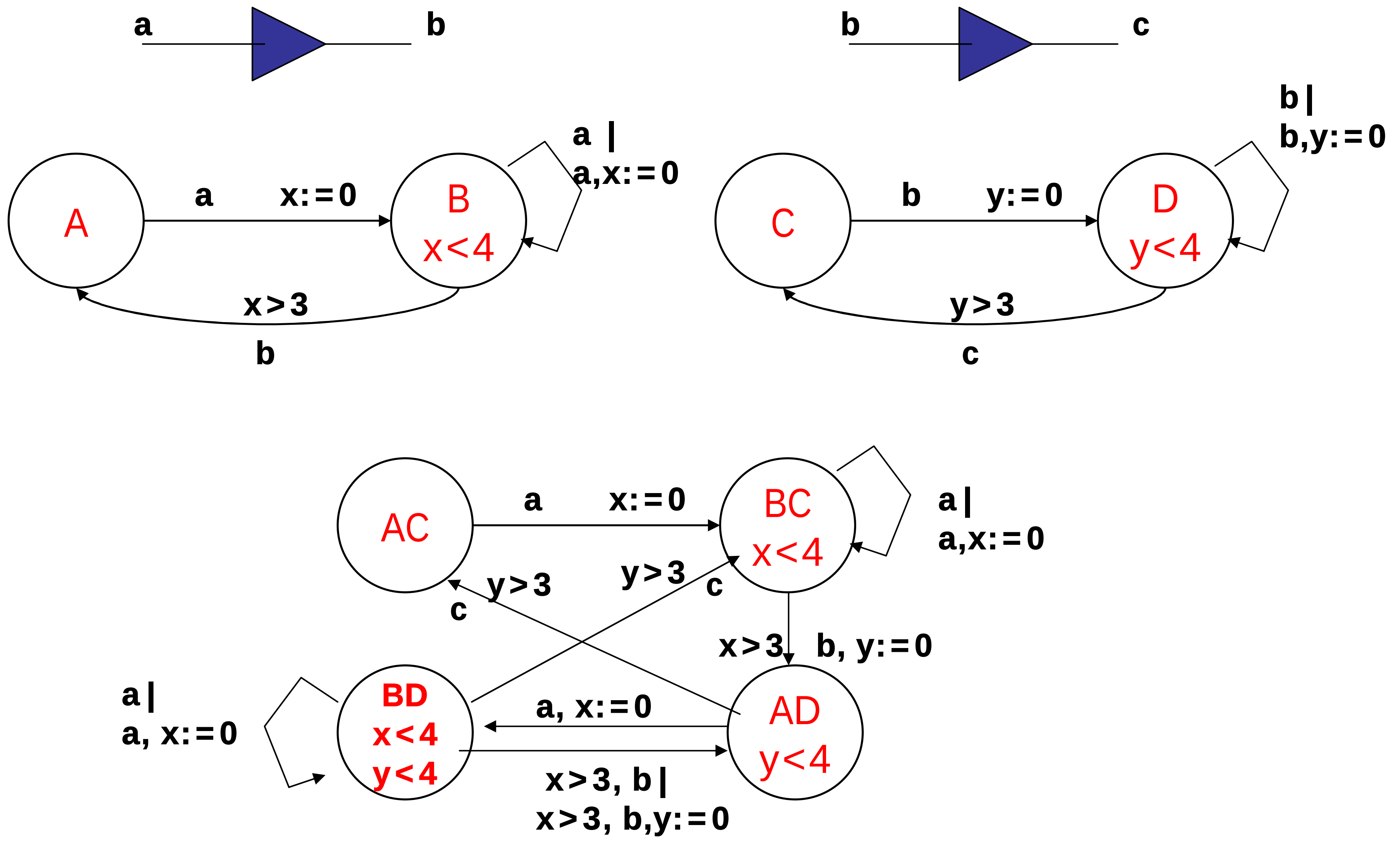
Timed Automata: Syntax

- A finite set *V* of locations
- A subset V^0 of initial locations
- A finite set Σ of labels (alphabet)
- A finite set *X* of clocks
- Invariant *Inv(l)* for each location: (clock constraint over *X*)
- A finite set *E* of edges. Each edge has
  - source location *l*, target location *l'*
  - label *a* in Σ (ε labels also allowed)
  - guard *g* (a clock constraint over *X*)
  - a subset λ of clocks to be reset

Timed Automata: Semantics

- For a timed automaton *A*, define an infinite-state transition system *S(A)*
- **States** *Q*: a state *q* is a pair *(l,v)*, where *l* is a location, and *v* is a clock vector, mapping clocks in *X* to *R*, satisfying *Inv(l)*
- *(l,v)* is **initial state** if *l* is in V^0 and *v(x)=0*
- **Elapse of time transitions**: for each nonnegative real number *d*, *(l,v)-d->(l,v+d)* if both *v* and *v+d* satisfy *Inv(l)*
- **Location switch transitions**: *(l,v)-a->(l',v')* if there is an edge *(l,a,g,λ,l')* such that *v* satisfies *g* and *v'=v[λ:=0]*

Product Construction



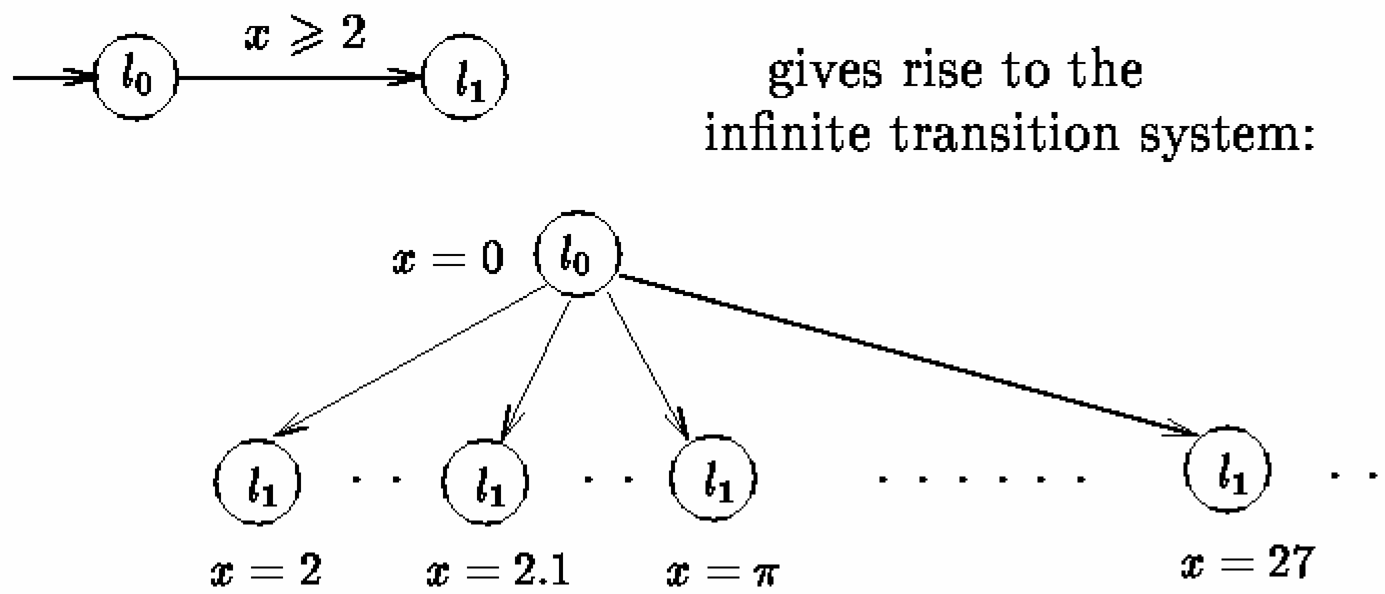
Verification

- System modeled as a product of timed automata
- Verification problem reduced to reachability or to temporal logic model checking
- Applications
  - Real-time controllers
  - Asynchronous timed circuits
  - Scheduling
  - Distributed timing-based algorithms

Course Overview

- ✓ Timed Automata Model
- ❑ Reachability
  - Preliminaries: Transition Systems and Equivalences
  - Region Graph Construction
  - Decidability Boundary
- ❑ Timed Regular Languages
  - Closure Properties and Complementation
  - Deterministic and two-way Automata
  - Robustness
  - Inclusion

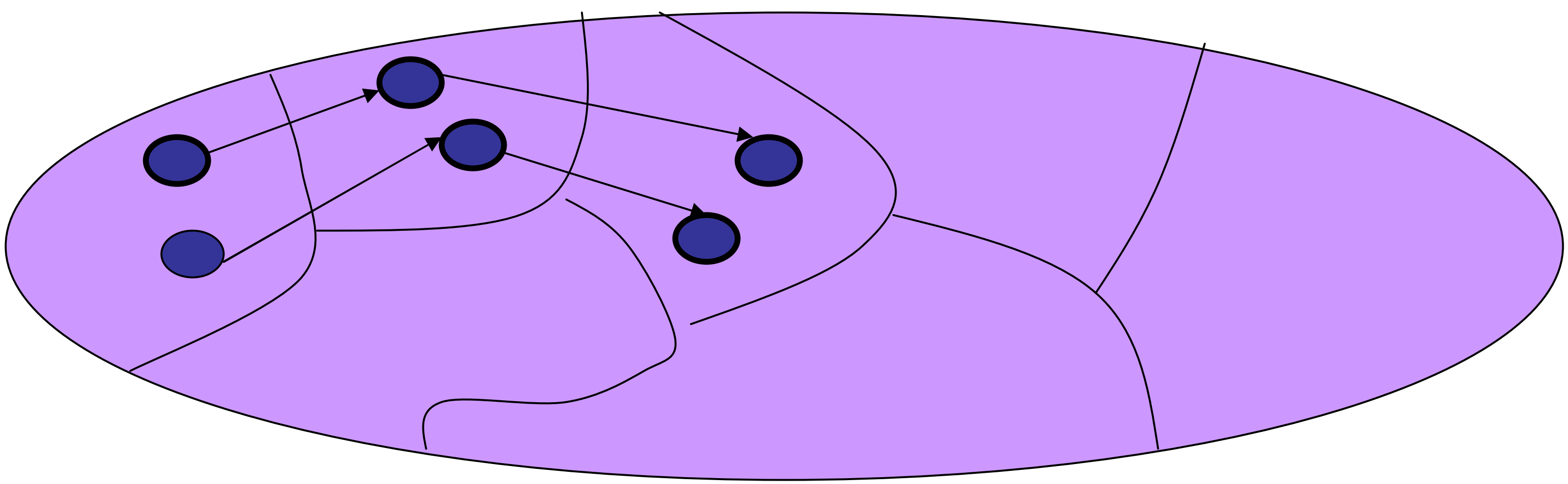
Reachability for Timed Automata



Is finite state analysis possible?
Is reachability problem decidable?

Finite Partitioning

Goal: To partition state-space into finitely many equivalence classes so that equivalent states exhibit similar behaviors



Labeled Transition System T

- Set Q of states
- Set I of initial states
- Set Σ of labels
- Set $\rightarrow$ of labeled transitions of the form $q \xrightarrow{a} q'$

Partitions and Quotients

- Let $T=(Q, I, \Sigma, \rightarrow)$ be a transition system and $\cong$ be a partitioning of Q (i.e. an equivalence relation on Q)
- Quotient $T/\cong$ is transition system:
  1. States are equivalence classes of $\cong$
  2. A state P is initial if it contains a state in I
  3. Set of labels is Σ
  4. Transitions: $P \xrightarrow{a} P'$ if $q \xrightarrow{a} q'$ for some q in P and some q' in P'

Language Equivalence

- **Language of *T*: Set of possible finite strings over Σ that can be generated starting from initial states**
- ***T* and *T'* are language-equivalent iff they generate the same language**
- **Roughly speaking, language equivalent systems satisfy the same set of "safety" properties**

Bisimulation

- Relation $\cong$ on $Q X Q'$ is a bisimulation iff whenever $q \cong q'$ then

  if q-a->u then for some u', $u \cong u'$ and q'-a->u', and

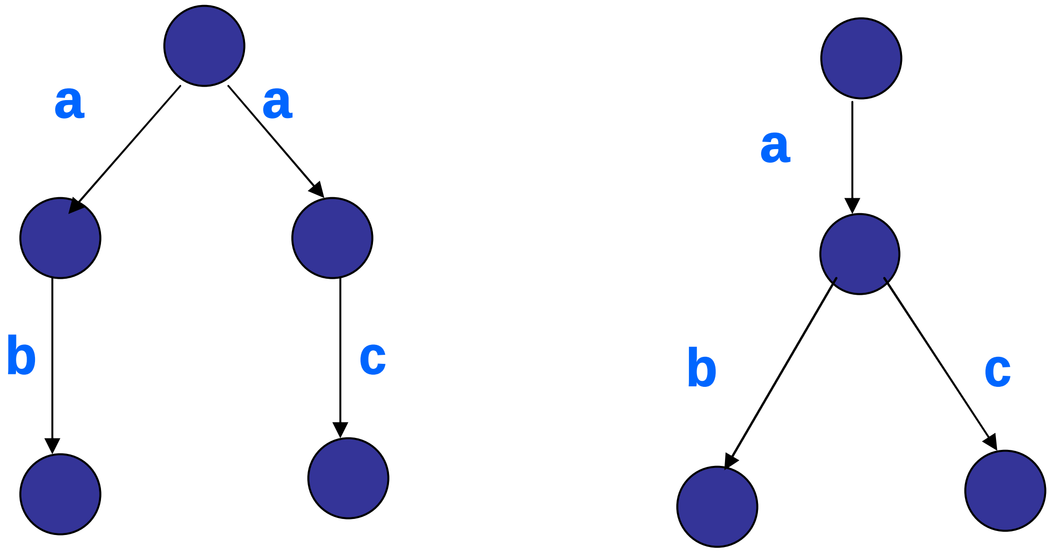
  if q'-a->u' then for some u, $u \cong u'$ and q-a->u.

- Transition systems T and T' are bisimilar if there exists bisimulation $\cong$ on $Q X Q'$ such that

  For every q in I, there is q' in I', $q \cong q'$ and vice versa

- Many equivalent characterizations (e.g. game-theoretic)
- Roughly speaking, bisimilar systems satisfy the same set of branching-time properties (including safety)

Bisimulation Vs Language equivalence



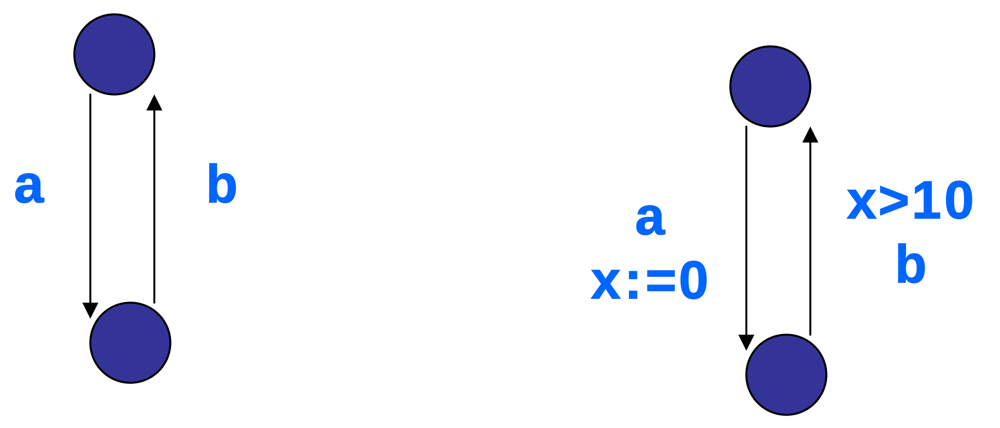
Language equivalent but not bisimilar

Bisimilarity -> Language equivalence

Timed Vs Time-Abstract Relations

- Transition system associated with a timed automaton:
  - Labels on continuous steps are delays in *R*: Timed
  - Actual delays are suppressed (all continuous steps have same label): **Time-abstract**
- Two versions of language equivalence and two versions of bisimulation
- Time-abstract relations enough to capture untimed properties (e.g. reachability, safety)

Time-abstract Vs Timed



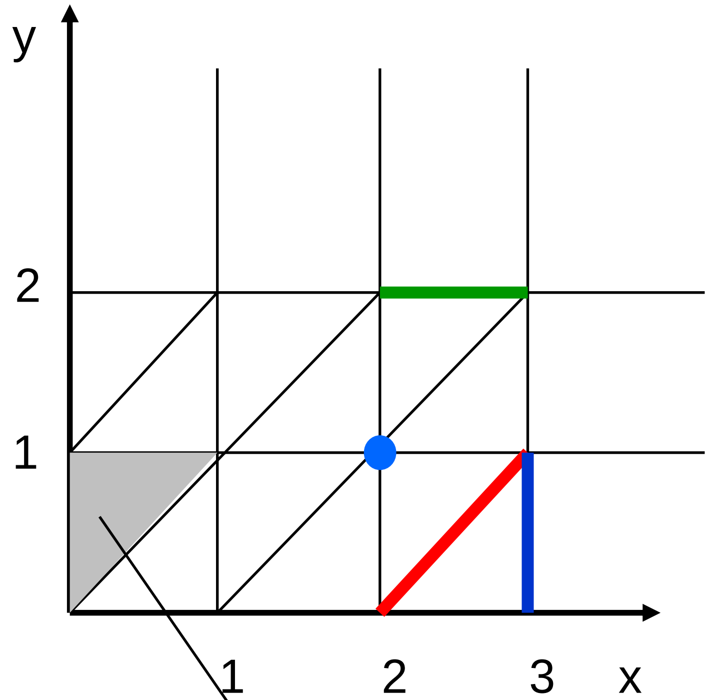
Time-abstract equivalent but not timed equivalent

Timed equivalence -> Time-abstract equivalence

Alur, Dill, 90

Regions

Finite partitioning of state space



Definition

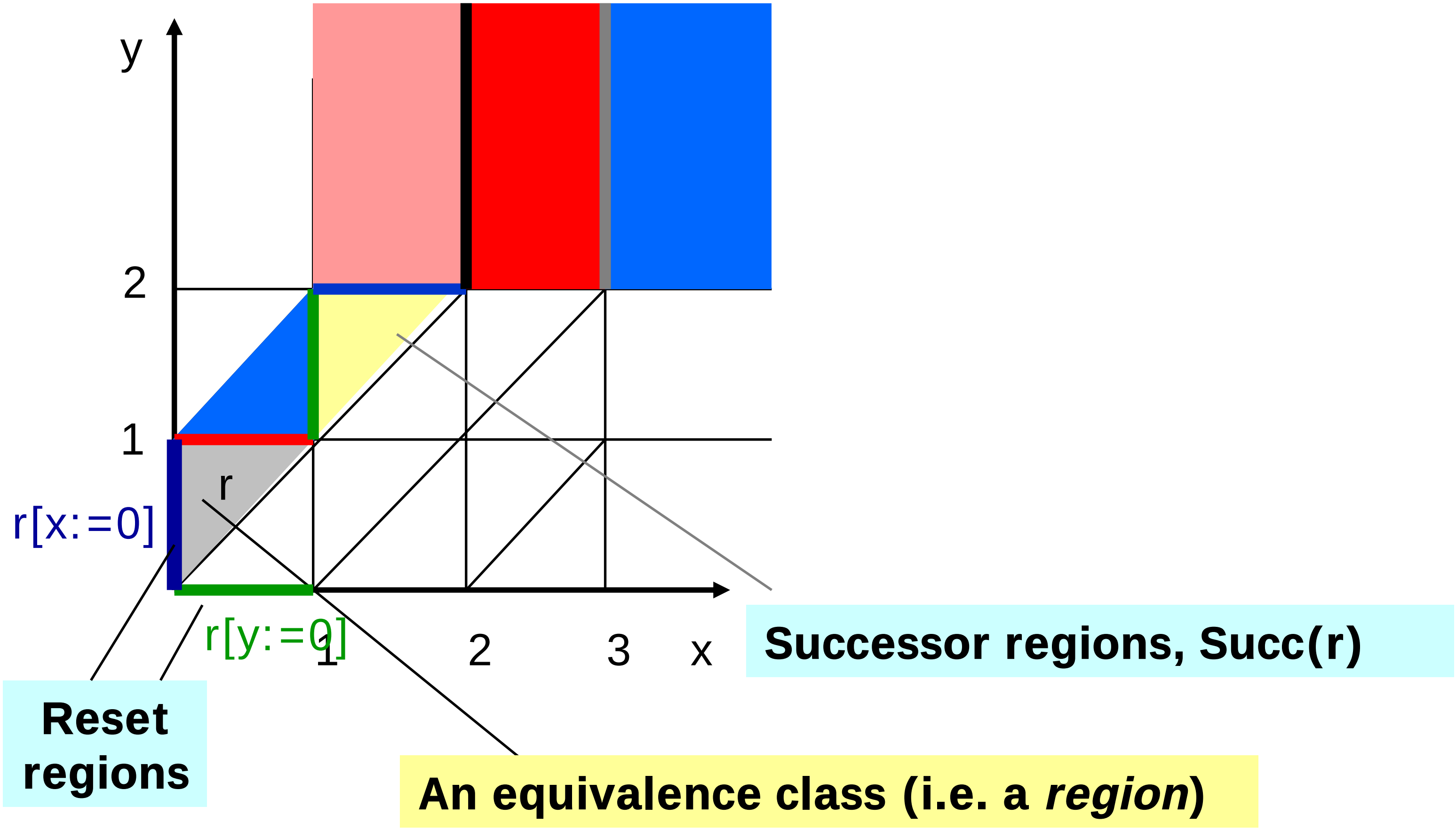
$w \cong w'$ iff they satisfy the same set of constraints of the form

$x_i < c,\ x_i = c,\ x_i - x_j < c,\ x_i - x_j = c$

for $c \le$ largest const relevant to x_i

An equivalence class (i.e. a *region*)
in fact there is only a *finite* number of regions!!

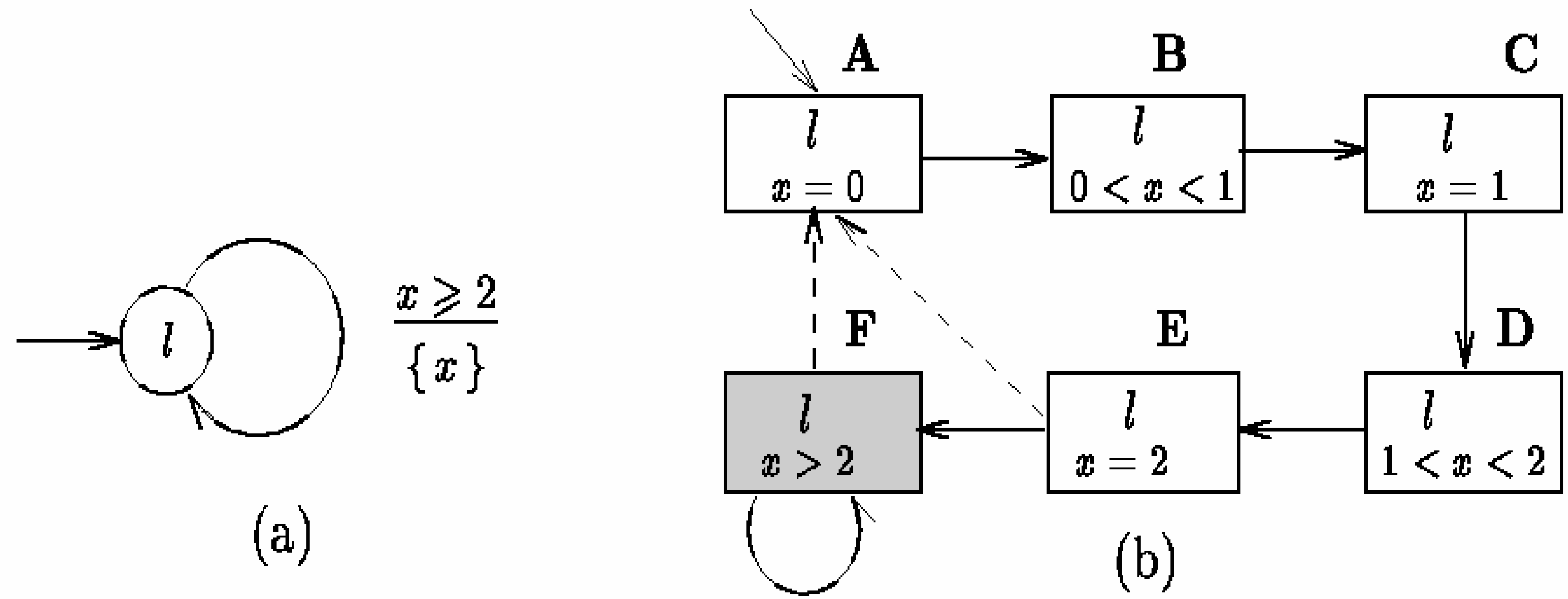
Region Operations



Properties of Regions

- The region equivalence relation $\cong$ is a **time-abstract bisimulation**:
  - Action transitions: If $w \cong v$ and (l,w) -a-> (l',w') for some w', then $\exists\, v' \cong w'$ s.t. (l,v) -a-> (l',v')
  - Delay transitions: If $w \cong v$ then for all real numbers d, there exists d' s.t. $w+d \cong v+d'$
- If $w \cong v$ then (l,w) and (l,v) satisfy the same temporal logic formulas

Region graph of a simple timed automata

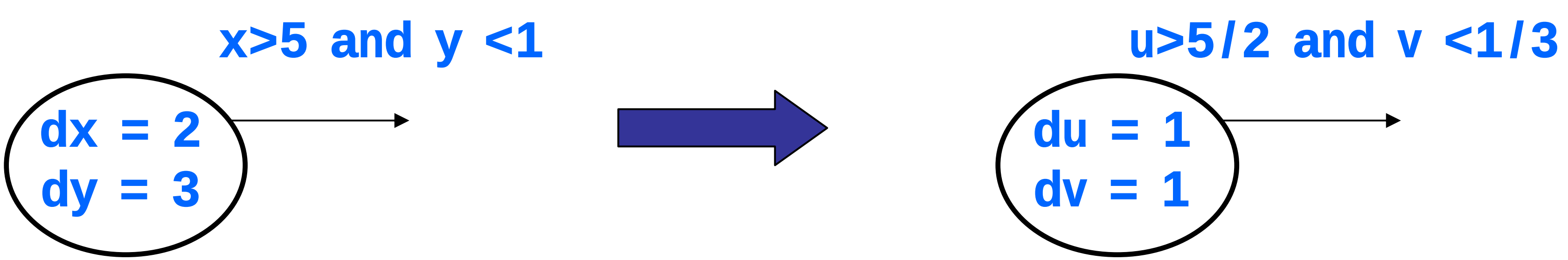


Region Graphs (Summary)

- Finite quotient of timed automaton that is time-abstract bisimilar
- Number of regions: (# of locations) times (product of all constants) times (factorial of number of clocks)
- Precise complexity class of reachability problem: PSPACE (basically, exponential dependence of clocks/constants unavoidable)
  - PSPACE-hard even for bounded constants or for bounded number of clocks

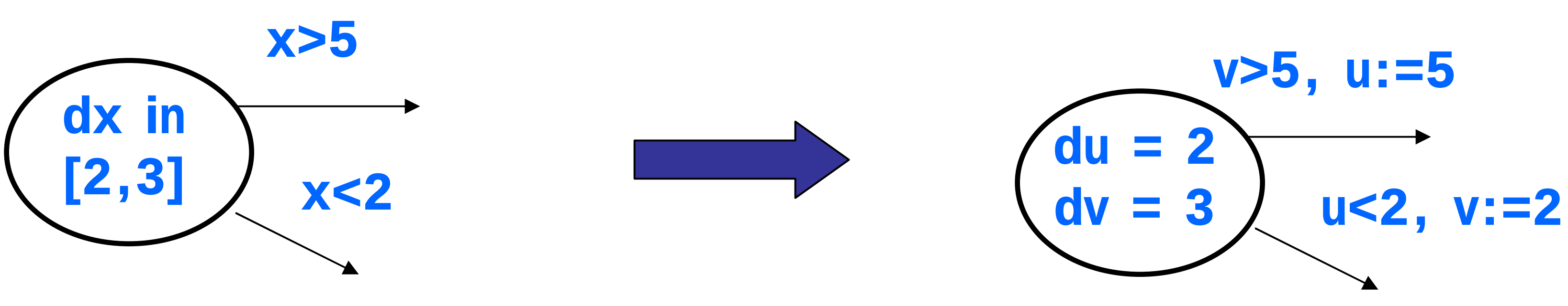
Multi-rate Automata

- Modest extension of timed automata
  - Dynamics of the form *dx = const* (rate of a clock is same in all locations)
  - Guards and invariants: *x < const*, *x > const*
  - Resets: *x := const*
- Simple translation to timed automata that gives time-abstract bisimilar system by scaling

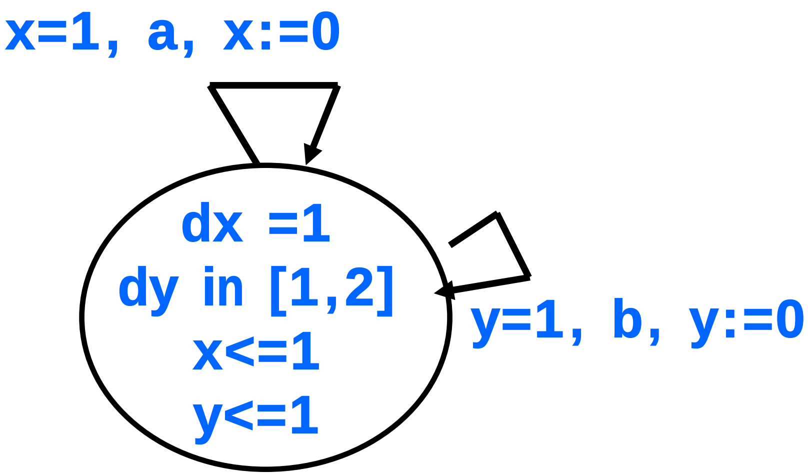


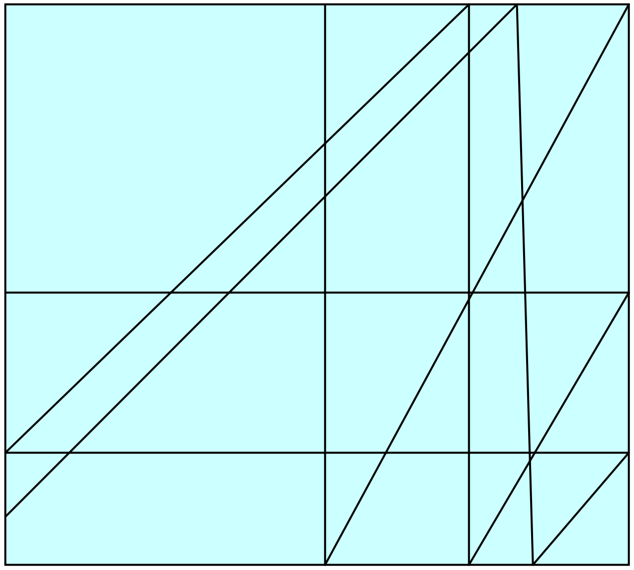
Rectangular Automata

- **Interesting extension of timed automata**
  - **Dynamics of the form *dx* in const interval (rate-bounds of a clock same in all locations)**
  - **Guards/invariants/resets as before**
- **Translation to multi-rate automata that gives time-abstract language-equiv system**



Rectangular Automata may not have finite bismilar quotients!

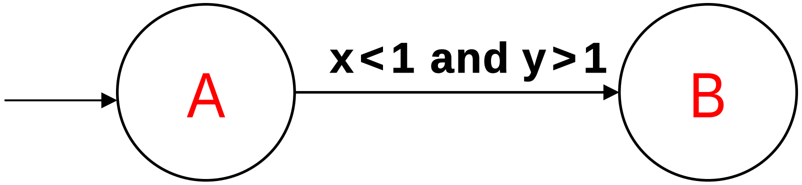




Decidable Problems

- Model checking branching-time properties (TCTL) of timed automata
- Reachability in rectangular automata
- Timed bisimilarity: are given two timed automata bisimilar?
- Optimization: Compute shortest paths (e.g. minimum time reachability) in timed automata with costs on locations and edges
- Controller synthesis: Computing winning strategies in timed automata with controllable and uncontrollable transitions

Limit Reachability



- Given A and error ε, define A^ε to be the rectangular automaton in which every clock *x* has rate in the interval $[1-\varepsilon, 1+\varepsilon]$
- A location *l* is limit reachable if *l* is reachable in A^ε for every $\varepsilon > 0$
- Limit reachability is decidable

Undecidable Reachability Problems

- Linear expressions as guards
- Guards that compare clocks with irrational constants
- Updates of the form x := x-1
- Multi-rate automata with comparisons among clocks as guards
- Timed automata + stop-watches (i.e. clocks that can have rates 0 or 1)

Many such results
Proofs by encoding Turing machines/2-counter machines
Sharp boundary for decidability understood

Course Overview

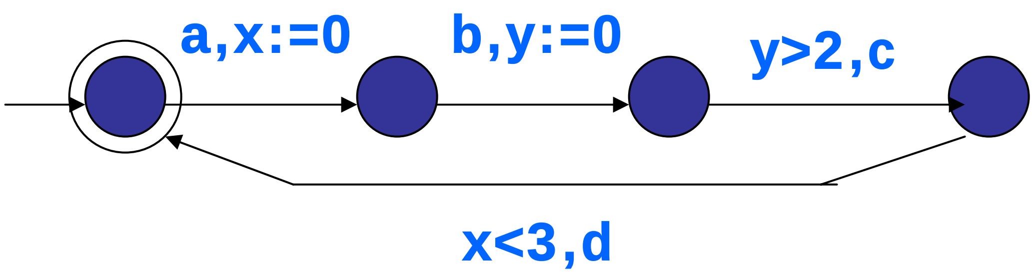
- ✓ Timed Automata Model
- ✓ Reachability
  - Preliminaries: Transition Systems and Equivalences
  - Region Graph Construction
  - Decidability Boundary
- ❑ Timed Regular Languages
  - Closure Properties and Complementation
  - Deterministic and Two-way Automata
  - Robustness
  - Inclusion

Timed Languages

- A timed word over Σ is a sequence $(a_0,t_0),(a_1,t_1)\ldots(a_k,t_k)$ with a_i in Σ, t_i in R, and $t_0<=t_1<=\ldots<=t_k$ (monotonicity of time)
- A timed language is a set of timed words
- Timed automata with final locations can be viewed as generators/acceptors of timed languages: A accepts $(a_0,t_0),(a_1,t_1)\ldots(a_k,t_k)$ if for some initial state q, final state q', there is a run

  $q\text{-}t_0\text{-}>\text{-}a_0\text{-}>\text{-}(t_1\text{-}t_0)\text{-}>\text{-}a_1\text{-}>\ldots\text{-}a_k\text{-}>q'$
- A timed language L is *timed regular* if there is a timed automaton whose timed language is L

Example



Words of the form (*abcd*)* such that *c* occurs after a delay of at least 2 wrt last *b*, and *d* occurs within 3 of last *a*

This timed language cannot be captured by any timed automaton with just 1 clock. In fact, expressiveness strictly increases with the number of clocks.

Untiming

- Given a timed language L over Σ the language Untime(L) consists of words $a_0,a_1,\ldots a_k$ such that there exists a timed word $(a_0,t_0),(a_1,t_1)\ldots(a_k,t_k)$ in L
- Thm: If L is timed regular, then Untime(L) is regular.
  - proof by region construction

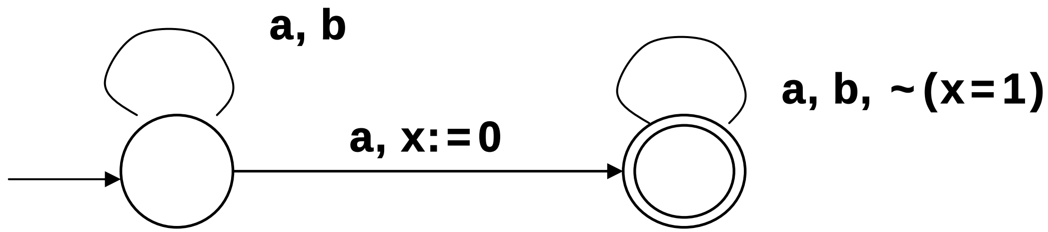
Not timed regular

- Delay between first and second event is the same as the delay between second and third.
  - Can compare delays only with constant bounds
- Every *a* symbol is followed by some *b* symbol after a delay of 1
  - Due to denseness, there can be unbounded number of *a* symbols in a unit interval
  - Complement of this language is timed regular
- Untimed language is $\{a^n b^n \mid n \text{ is an integer}\}$

Properties of Timed Regular languages

- Set of timed regular languages is closed under union, intersection, but not under complementation
- For every *k*, there is a timed regular language that cannot be expressed using only *k* clocks (strict hierarchy)
- Epsilon-labeled switches contribute to expressive power
  - the language "symbols occur only at integer times" crucially uses epsilon-labeled edges

Non-closure under complementation



- *L* contains timed words *w* s.t. there is *a* at some time *t*, and no event at time *t+1*
- Claim: *~L* is not timed regular
- Let *L'* contain timed words *w* s.t. untimed word is in *a*b**, all *a* symbols are before time *1*, and no two *a* events happen simultaneously
- A word $a^n b^m$ is in Untime*(~L & L')* iff *m>=n*
- *~L & L'* is not timed regular, but *L'* is. So *~L* cannot be timed regular

Undecidability

- Universality problem (given a timed automaton A, does it accept *all* timed words) is undecidable
  - Proof by reduction from halting problem for 2-counter machines
  - Symbols in time interval $[k, k+1)$ encode the k-th configuration of a run of the machine
  - Denseness of time ensures configurations can be of unbounded lengths
  - Crux: how to relate successive configurations
  - Copying of *a* symbols: every *a* at time t in one interval has a matching *a* in the next interval at time $t+1$
  - Absence of such copying can be guessed by a timed automaton

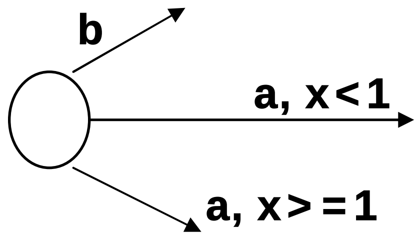
Do we have the "right" class?

- Corollary: Inclusion and Equivalence problems are undecidable for timed automata
  - Hierarchical verification using automata-theoretic setting not possible
- Closed under union, intersection, projection, concatenation, but not complementation
- Maybe the source of undecidability and non-closure under complementation is ability to model precise time constraints
  - some two *a* symbols are time 1 apart

Search for a “better” class

- Complementable subclasses
  - (Bounded two-way) Deterministic automata
  - (Recursive) Event-clock automata
- Semantics
  - (Inverse) Digitization, Open/closed automata
  - Robust timed automata
- Alternative characterizations
  - Timed regular expressions
  - Monadic second order theory + distance
  - Linear temporal logics with real-time

Deterministic Timed Automata



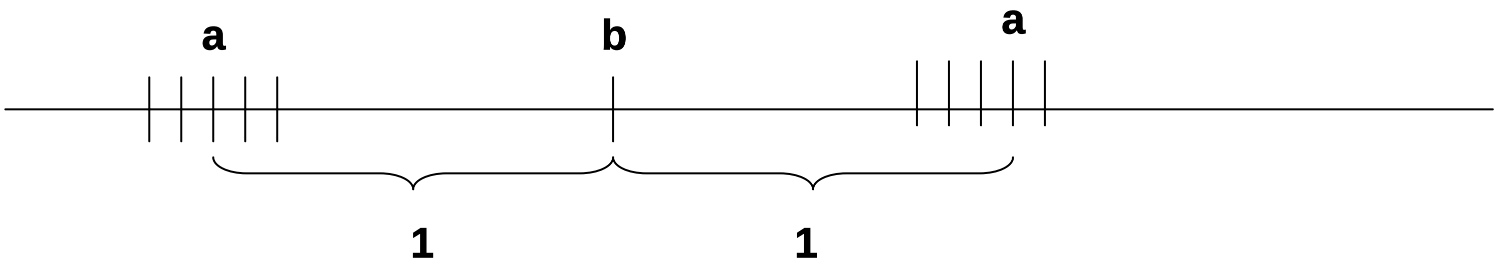
- A timed automaton is deterministic if
  - Only one initial location
  - No edges labeled with ε (some relaxation possible)
  - Two edges with same source and same label have disjoint guards
- Key property: At most one run on a given timed word
  - To complement, complete & complement final locations

Properties of DTA Languages

- Closed under union, intersection, complement, but not projection
- Emptiness, universality, inclusion, equivalence all decidable in PSPACE
- Strictly less expressive than nondeterministic
  - There exists i and j s.t. $t_j=t_i+1$
- Open problem: Given a timed automaton A, is L(A) a DTA-language? (see Tripakis00)

Alur, Henzinger, 92

Two-way Deterministic Timed Automata



- Languages of deterministic timed automata not closed under “reverse”
  - Deterministically identified *b* is *followed* by *a* after 1 unit is a DTA-language
  - Deterministically identified *b* is *preceded* by *a* before 1 unit is not a DTA language
- More tricky example: Every *a* is followed by some *b* within a delay of [1,2] (see AFH96)

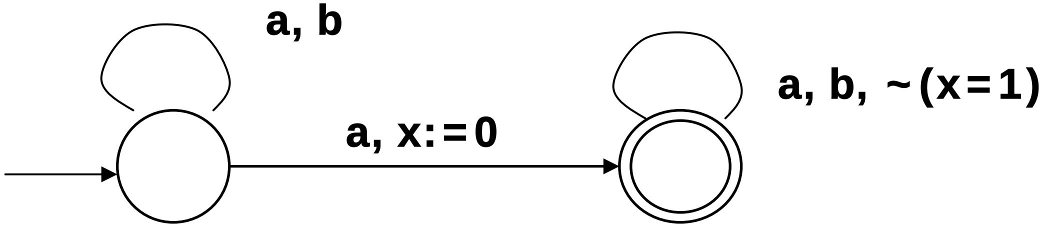
Properties of two-way automata

- Bounded reversal two-way timed automata: k-bounded automaton visits any symbol at most k times
- Every k-bounded automaton can be simulated by a forward non-deterministic one
- DTA^k: Languages of k-bounded deterministic timed automata
- DTA^k is closed under union, intersection, complementation, and has decidable inclusion/equivalence problems
- DTA^k forms a strict hierarchy with increasing k

Robust Timed Automata

- Intuition: Rule out the ability to relate events "accurately" by forcing fuzziness in semantics
- Accept/reject a word only if a dense subset around it is accepted/rejected
- For two timed words *w* and *w'* with same untimed word, $d(w,w') = max_i \, |t_i - t'_i|$
- Use this metric to define open/closed sets
- Robust language of A is interior of the smallest closed set containing L(A)

Robust acceptance



- Robust language of this automaton is all timed words
- Isolated words cannot be accepted/rejected
- Open timed automata: Timed automata where all guards are strict (x<c, x>c)
- Given a timed automaton A, one can construct an open timed automaton B with the same robust language, which is empty iff L(B) is empty
- Emptiness of robust language is decidable

Robust timed automata

- Robustness unfortunately does not solve non-complementability and undecidability of inclusion [HR00]
- *L* contains timed words *w* s.t. untimed word is a^*b^*, and there exist consecutive *a* symbols at times *t* and *t'* with no b in $[t+1,t'+1]$
- *L* is a robust timed language, but its complement is not
- Universality of robust timed automata is undecidable

Back to Language Inclusion

- Given timed automata A and B, checking if L(A) is contained in L(B) is decidable if
  - B has only 1 clock or
  - All constraints in B use the constant 0
- B cannot be determinized, and one has to consider potentially unbounded copies of the clock of B, but termination uses well-founded ordering on the configurations
- Any relaxation on resources of B leads to undecidability

Resource-bounded Inclusion

- Critical resources of a timed automaton
  - Granularity 1/m (all constants are multiples of this granularity)
  - Number of clocks k
- An observer C distinguishes automata A and B if L(A)&L(C) is non-empty but L(B)&L(C) is empty
- Resource bounded inclusion: Given A, B, and resource bound (k,1/m) check if there is an observer C with k clocks, granularity 1/m, and distinguishes A and B
- Resource bounded inclusion is decidable

Topics Not Covered

- Timed ω-languages
- Linear/Branching-time real-time logics
- Connections to monadic logics, regular expressions, circuits
- Timed branching-time equivalences
- Efficient implementations, tools, applications
- Adding probabilities
- Concurrency: Process algebras, Petri nets
- Timed automata + Parameters
- Games and controller synthesis

Open Problems

- There is no “final” answer to “what is the right class of timed languages”
  - Perturbation by adding drifts to clocks?
- Are there subclasses of timed automata for which reachability is less than PSPACE
  - Automata with “small” strongly-connected components
- Games on weighted timed graphs
  - See a recent paper ABM04 [ICALP]