Verification of Real Time Systems - CS5270
8th lecture

Hugh Anderson

National University of Singapore
School of Computing

March, 2007

Duckburg...

Outline

1. Administration
   - Assignment 2
   - The road map...
2. Efficiency in TTS
   - From regions to zones
   - Matrix notation and zone operations
   - Closed zones and graph representation
3. Preliminaries to Model Checking
   - Behaviour, safety, liveness, automata, reachability...
   - Extensional and intensional logic
   - Linear and branching time

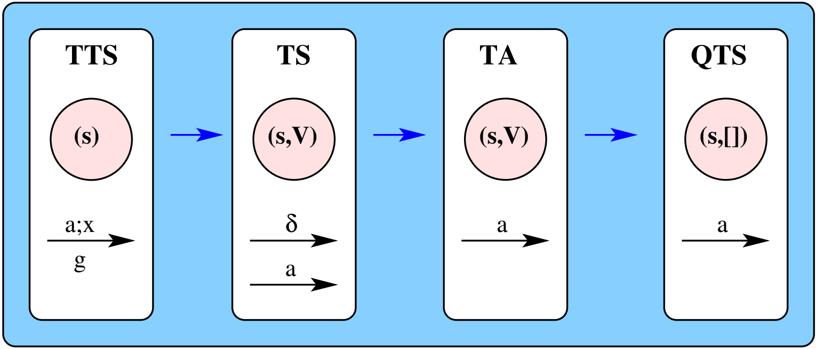
Assignment 2

A reminder... Assignment number 2:

- On the web site
- Due on 22nd March ...

The reduction...

What we did...



The immediate road map

The topics:

- **TTS: Timed transition systems**
  - Reduction: $\mathrm{TTS} \rightarrow \mathrm{TS}_{\mathrm{TTS}} \rightarrow \mathrm{TA}_{\mathrm{TTS}} \rightarrow \mathrm{RTS}$ (by quotienting)
- **Efficiency in TTS**
  - Regions

  - Zones
    - Notation
    - Operations
    - Optimizations
- **Preliminaries for Model Checking**
  - Behaviour, safety, liveness, automata, reachability
  - Temporal logic

What is wrong with regions?

Unwieldy:

- The number of regions can be very large:
  - It is exponential in the number of clocks, and in the size of the maximal constraints appearing in the clock constraints.
  - As a result, practical verification of transition systems based on regional transition systems becomes infeasible.

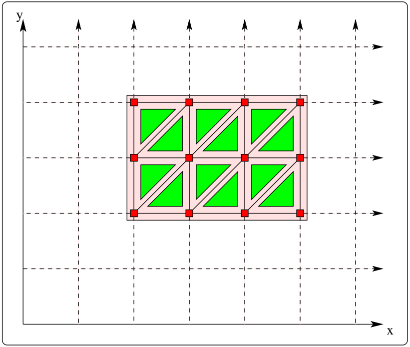
What is a zone?

A more compact representation:

- ...of equivalence classes of valuations....
  - Can be efficiently represented as Difference Bounded Matrices (edge weighted directed graphs).
  - DBMs admit a canonical representation.
  - DBMs can be manipulated efficiently.

Regions versus zones

47 regions versus 1 zone!



47 regions in zone $(2 \leq x \leq 5) \wedge (2 \leq y \leq 4)$

Formally:

Definition of zone:

- A zone $\mathcal{Z}$ is a clock constraint of the “two-variable difference” form

$$\mathcal{Z} ::= x \;\mathrm{op}\; c \mid x - y \;\mathrm{op}\; c \mid z_1 \wedge z_2$$

where $\mathrm{op} \in \{<, \leq, >, \geq\}$, and $c \in \mathbb{N}$.

Zone is a convex hull

What is this?

- A zone $\mathcal{Z}$ is a convex union (or *hull*) of all the regions $\mathcal{R}$: $\mathcal{Z} = \bigcup_i \mathcal{R}_i$.
- To encode zones in a DBM, we
  - construct a new clock variable x_0 which will always have the value 0, and then encode all constraints as $x_i - x_j < m$ or $x_i - x_j \leq m$ where $m \in \mathbb{Z}$.
  - For example the following terms on the left are translated to those on the right:

$$\begin{array}{rcl c rcl} x_2 & < & 3 & \Longrightarrow & x_2 - x_0 & < & 3 \\ x_5 & \geq & 7 & \Longrightarrow & x_0 - x_5 & \leq & -7 \\ x_2 - x_5 & > & 8 & \Longrightarrow & x_5 - x_2 & < & -8 \end{array}$$

Finiteness and hence termination

Ignore constraints bigger than C_x:

- To ensure termination:
  - Remove constraints of the form $x < m$, $x \leq m$, $x - y < m$ and $x - y \leq m$ if $m > C_x$.
  - Replace $x > m$, $x \geq m$ with $x > C_x$ if $m > C_x$.
  - Replace $y - x > m$, $y - x \geq m$ with $y - x > C_x$ and $y - x \geq C_x$ if $m > C_x$.

Matrix notation

Compact notation:

For $n - 1$ clock variables, we then write out an $n \times n$ matrix M, with elements drawn from $(\mathbb{Z} \times \{<, \leq\}) \cup \infty$ according to the following rules:

- For constraints like $x_i - x_j < c$, set $M_{i,j} = (c, <)$
- For constraints like $x_i - x_j \leq c$, set $M_{i,j} = (c, \leq)$
- Otherwise set $M_{i,j} = \infty$

Matrix notation

Consider this clock zone:

$$(0 \leq x_1 < 1) \wedge (0 < x_2 < 3) \wedge (x_2 - x_1 \geq 1)$$

then the DBM is

	x_0	x_1	x_2
x_0	$(0, \leq)$	$(0, \leq)$	$(0, <)$
x_1	$(1, <)$	$(0, \leq)$	$(-1, \leq)$
x_2	$(3, <)$	∞	$(0, \leq)$

Tightening constraints

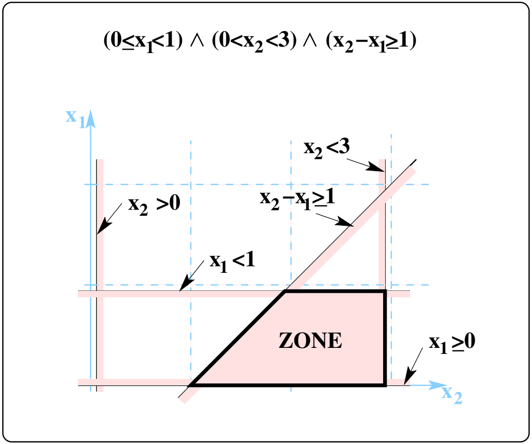
The canonical DBM:

- Obtained by strengthening/tightening all the constraints:

	x_0	x_1	x_2
x_0	$(0, \leq)$	$(0, \leq)$	$(-1, \leq)$
x_1	$(1, <)$	$(0, \leq)$	$(-1, \leq)$
x_2	$(3, <)$	$(3, <)$	$(0, \leq)$

Tightening constraints

Halfspace view:

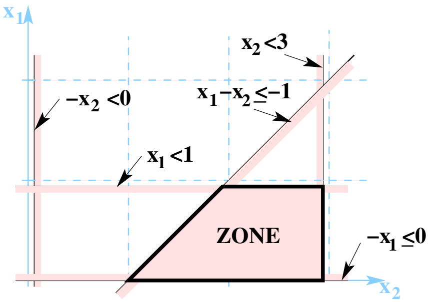


Tightening constraints

Halfspace view:

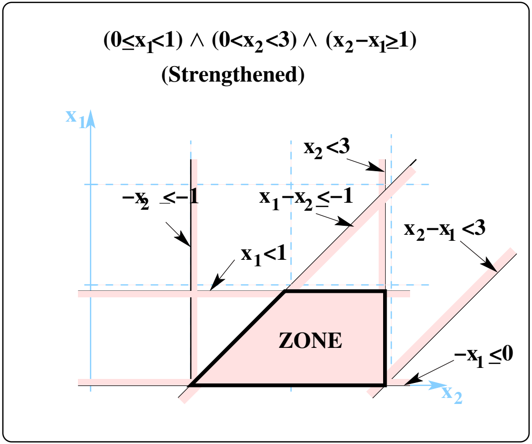
$(0 \leq x_1 < 1) \wedge (0 < x_2 < 3) \wedge (x_2 - x_1 \geq 1)$

(Normalised)



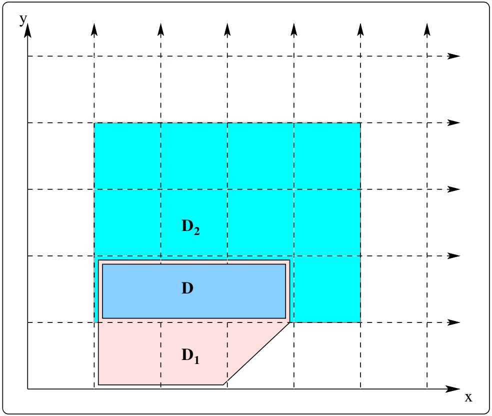
Tightening constraints

Halfspace view:



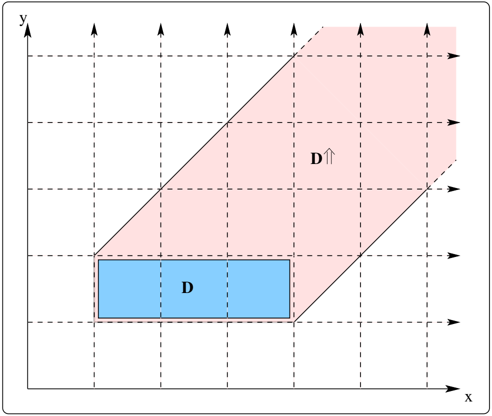
Operations on zones

The intersection:



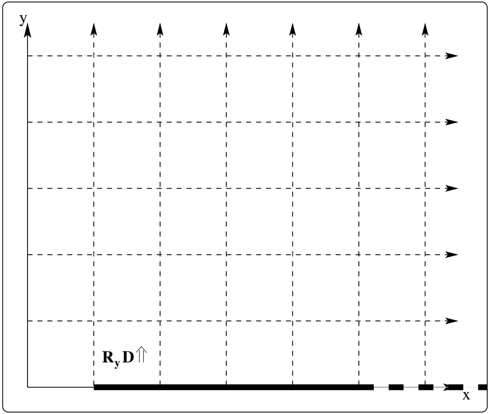
Operations on zones

Time elapses:



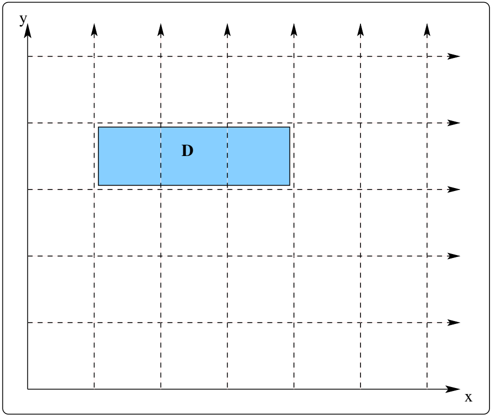
Operations on zones

A clock is reset:



Operations on zones

The PAST operation?



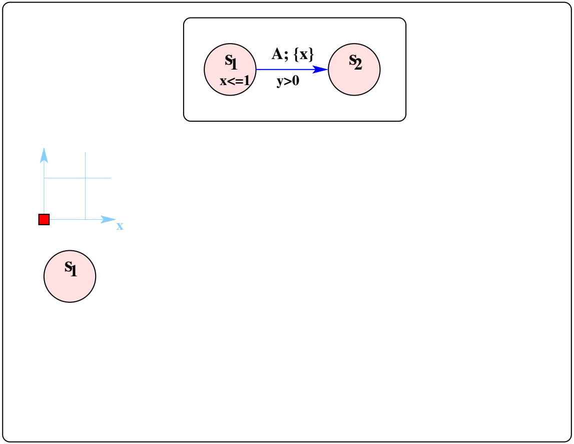
Constructing regional/zone transition systems

Practice versus mathematics:

- In the mathematics, en-route to the finite RTS or ZTS, we construct (infinite) transition systems.
- This is fine, but not actually possible (obviously).
- Instead we generate the transition systems in one step from the TTS.
- The following slides attempt to show the flavour of the algorithm in pictures...

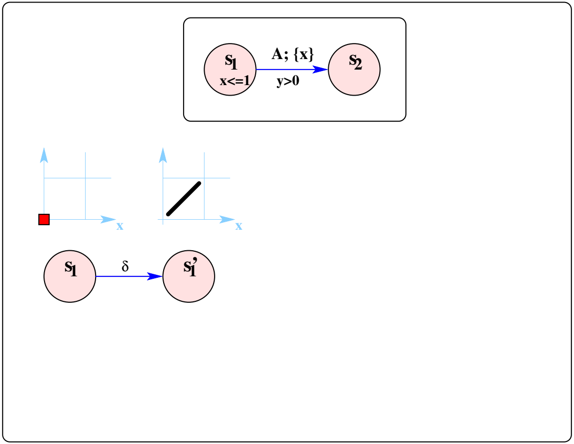
Drawing the operations (regions)

Show the regions in a diagram: Original state



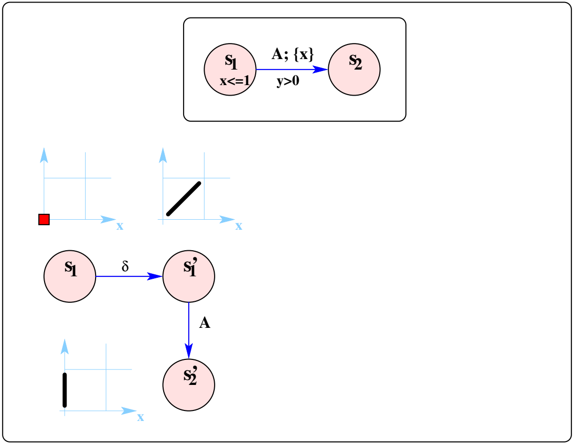
Drawing the operations (regions)

Show the regions in a diagram: Time passing



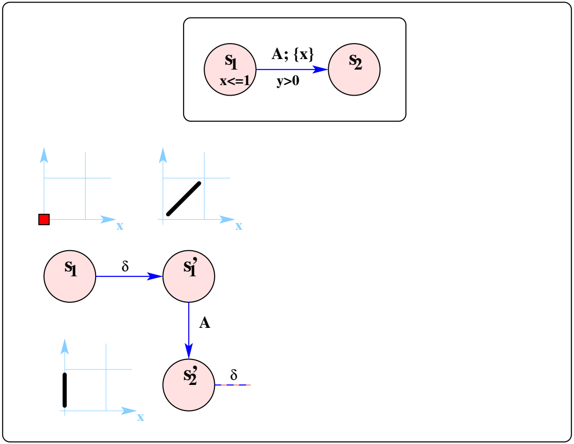
Drawing the operations (regions)

Show the regions in a diagram: Action move



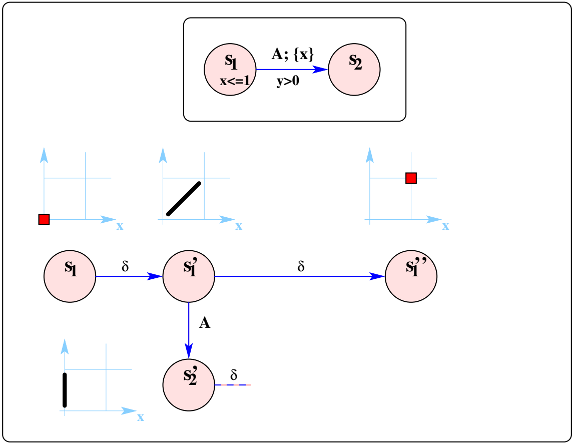
Drawing the operations (regions)

Show the regions in a diagram: Carry on time...



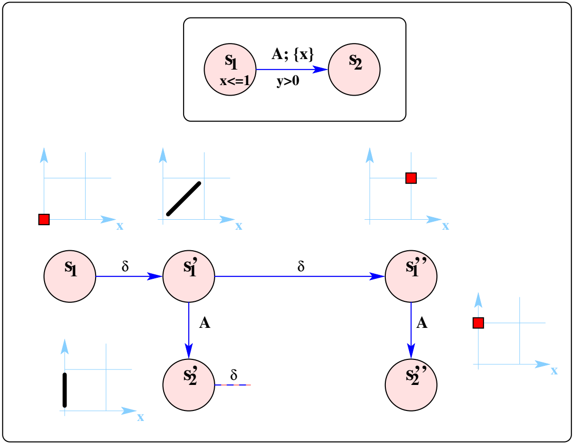
Drawing the operations (regions)

Show the regions in a diagram: Time passing move



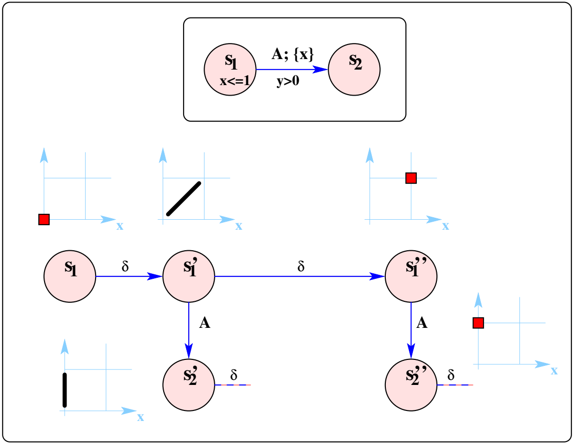
Drawing the operations (regions)

Show the regions in a diagram: Another action...



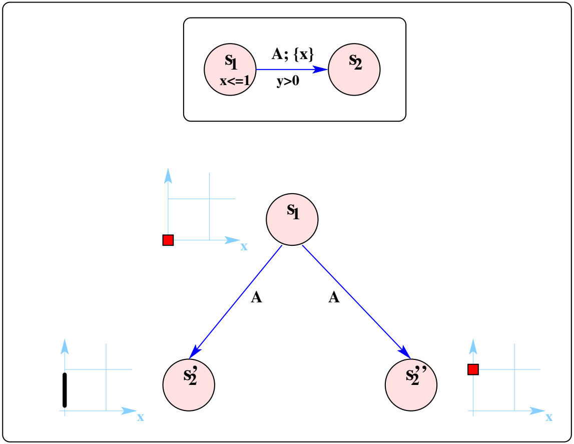
Drawing the operations (regions)

Show the regions in a diagram: and so on...



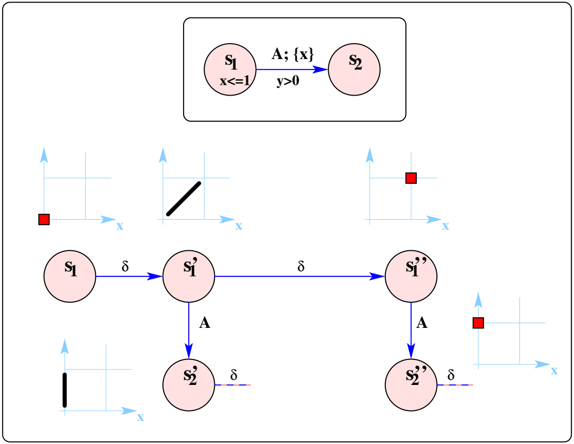
Drawing the operations (regions)

Show the regions in a diagram: TIME ABSTRACTED



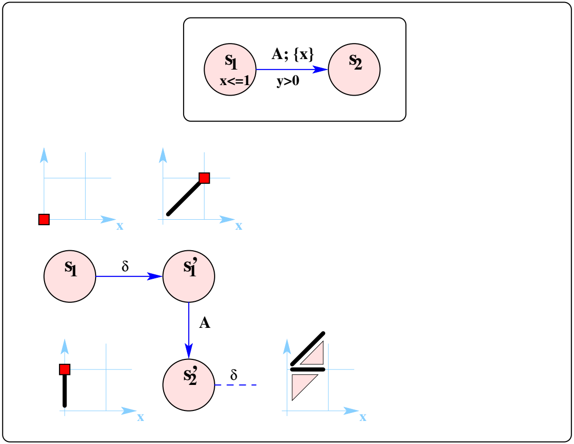
Drawing the operations (zones)

Show the regions in a diagram: from before...



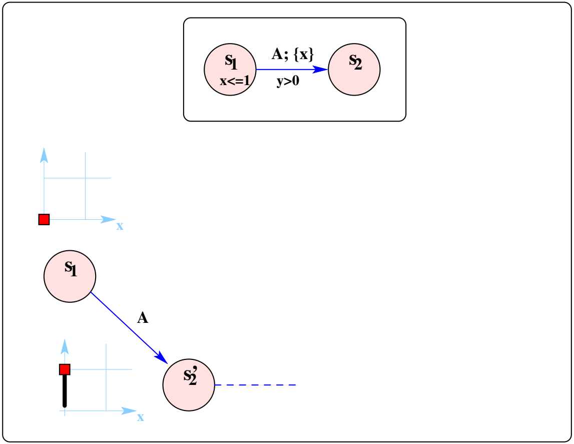
Drawing the operations (zones)

Show the zones in a diagram: smaller



Drawing the operations (zones)

Show the zones in a diagram: TIME ABSTRACTED



Operations on zones

Zones are relatively easily manipulated:

- Following three operations are needed for use in evaluating zone transitions:
  - If $\mathcal{D}_1$ and $\mathcal{D}_2$ are two clock zones, then the **intersection** of the zones is a new clock zone $\mathcal{D}_1 \wedge \mathcal{D}_2$.
  - $\mathcal{D} \Uparrow$ is the **time-elapsed zone** defined by $\mathcal{D} \Uparrow = \{V + \delta \mid V \in \mathcal{D}\}$ with $\delta \in \mathbb{R}_{\geq 0}$.
  - The **clock-reset zone** $R_X\mathcal{D}$ is defined by $R_X\mathcal{D} = \{R_X(V) \mid V \in \mathcal{D}\}$ where $R_X(V)(\lambda) = 0$ if $\lambda \in X$ or $R_X(V)(\lambda) = V(\lambda)$ otherwise.

Want a canonical representation

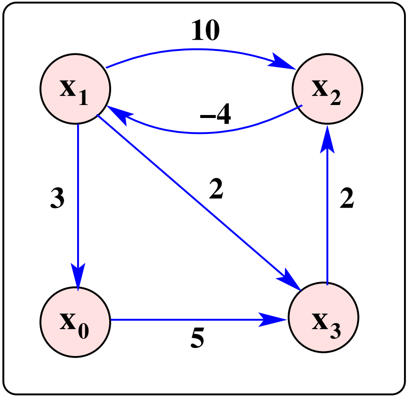
Equivalence of zones:

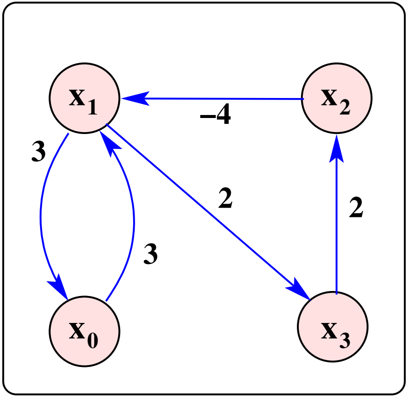
- We do not want two different zones to represent the same set of valuations (i.e. $(y - x \leq 3, x = 2, y = 4)$ the same as $(y - x = 2, x = 2, y = 4)$.

Definition: A zone is *closed* if no constraint can be strengthened without reducing the set of associated valuations.

Closed zones are equivalent iff identical

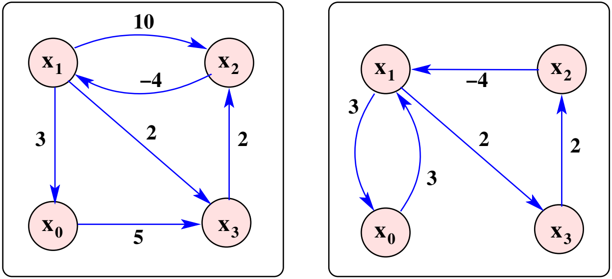
Graph representations (simplified)...





Closed zones are equivalent iff identical

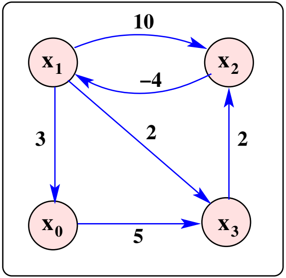
Graph shortest path reduction:

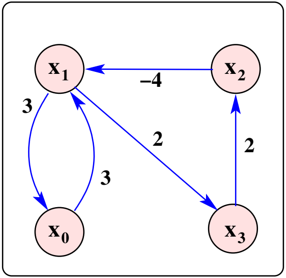


A shortest path reduction is performed on the graph (computed in $O(n^3)$ time), where redundant edges are removed when they can be.
For example $x_1 \xrightarrow{10} x_2$ is replaced by $x_1 \xrightarrow{2} x_3 \xrightarrow{2} x_2$.

Closed zones are equivalent iff identical

Graph shortest path reduction:



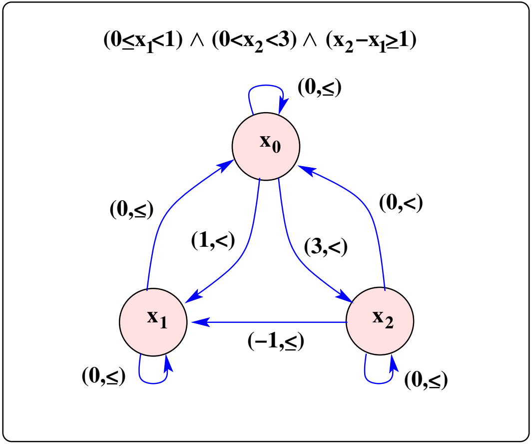


If D is closed then D is a subset of D' iff for every constraint $x - y \leq m'$ in D' there is $x - y \leq m$ in D with $m \leq m'$.

If D is closed then D is non-empty iff there are no negative weight cycles in the graph.

DBM example repeated

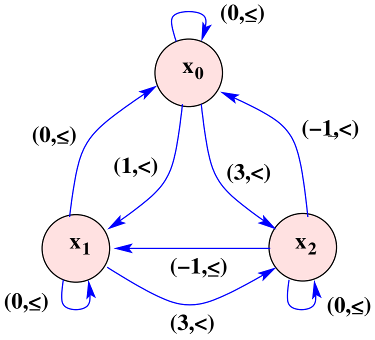
Graphs for DBMs: graph reduction...



DBM example repeated

Graphs for DBMs: graph reduction...

$(0 \leq x_1 < 1) \wedge (0 < x_2 < 3) \wedge (x_2 - x_1 \geq 1)$



Behaviours

The behaviour of a TS...

- ...is its set of *runs*, or its set of *computations*.
- To *verify* behaviours against a property, we can consider questions like:
  - Does every computation (run) of the transition system have a desired property X ? or
  - Is it true that in no computation, C is immediately followed by on-ac?.

Safety and Liveness

Two types of behaviours:

- In handouts, the ideas of safety and liveness were introduced, identifying two types of behaviours that require different analysis methods.
  - A safety property is like “something bad doesn’t happen”, whereas
  - liveness is like “something bad (or good) must eventually happen".
- We can often formulate safety properties in terms of the reachability of a state.

Checking TS

What is an automata?

- An automata is a state transition system with some set of accepting states, which may be used to distinguish between *good* and *bad* computations.
- We can use automata matching a particular transition system to specify *desired* behaviour of the system, in a form like "Is there a run of the automaton that leads to the (desired) accepting state?", or "Is there a run of the automaton that leads to an accepting state in which property *P* holds?".
- These are examples of a reachability problem.

Finite automaton

Definition:

A finite automaton is a 5-tuple $(Q, \Sigma, \Delta, q_0, F)$, where

- Q is a finite set called the **states**
- Σ is a finite set called the **alphabet**
- $\Delta : Q \times \Sigma \rightarrow Q$ is the **transition function**
- q_0 is the **start state**
- $F \subseteq Q$ is the set of **accepting states**

Checking TS

Automata theory...

- These sort of problems have clear links to automata theory, and
- we could easily cast a lot of this discussion in terms of the languages accepted by (finite) automata.
- To reason about liveness properties, we need to consider *infinite* sequences.
- A Büchi automaton is an extension of a finite state automaton to one which accepts an infinite input sequence if, and only if, there is a run of the automaton which has infinitely many states in the set of final states.

Büchi automata

Definition:

A Büchi automaton is a 5-tuple $(Q, \Sigma, \Delta, q_0, F)$, as for a regular automaton, but with F interpreted differently. In particular $s_0\, a_0\, s_1\, a_1\, s_2\, \ldots$ is an accepting infinite trace if

- $s_0 \in Q$
- $(s_i, a_i, s_{i+1}) \in \Delta$ for all i
- For infinitely many j, the state s_j is in F

Büchi automata

What are they good for?

- They are useful for specifying behavior of nonterminating systems, such as
  - hardware (electronic circuits) or
  - operating systems.
- For example, you may want to specify a property like
  - “for every measurement, a recording eventually follows”, or
  - the reverse “there is a measurement which is not followed by a recording".
- For the second example, an argument limited to finite sequences cannot satisfy this property.

Properties: Reachability and deadlock

Are related...

- For example, is there a run leading to *deadlock*?
- A deadlocked system can do no more computation, more formally:

Definition: The run $s_0 \stackrel{*}{\Longrightarrow} s_k$ with $s_0 \in S_{\mathrm{in}}$ is in **deadlock** if no action is enabled at s_k.

Properties: qualitative

Questions such as...

- Every request is eventually served.
- The sensor signal x11 is sensed infinitely often.
- From any stage of the computation the all clear state can be reached within 3 steps

Properties: quantitative

Questions such as...

- Every request is served within 3 microseconds.
- The sensor signal x11 is sensed every 10 milliseconds for ever.
- From any stage of the computation the all clear state can be reached within 1 second.

The universe is not black and white

Consider...

Please answer YES or NO: Will the next answer you give me be NO?

You are either going to die in a bomb raid or you are not...

- *Extensional* logic means that you can determine the truth of a formula from the truth values of its parts.
- *Intensional/modal* logic refers to QUALIFIED truth (words like *could*, *eventually*, *possibly* and so on).

Modal logic

QUALIFIED truth

- The basic modal operators are
  - $\Box$ which represents necessity and
  - its dual $\Diamond$ which represents possibility ($\Diamond A = \neg\Box\neg A$).
- The language of modal logic consists of
  - propositional variables,
  - a set of Boolean connectives such as $\{\wedge, \vee, \neg\}$, and
  - the modal operators.

Temporal logic (a modal logic)

Consider...

"The engine is too hot."

- The *meaning* is clear, it does not vary with time, but ...
- the truth value of the assertion *can* vary in time.
  - Sometimes it is true, and sometimes it is false, and
  - it is never true and false simultaneously.
- Temporal logics are a good mechanism for expressing *qualitative* temporal properties of reactive systems.
- Operators related to TIME, so that (for example) $\Box\phi$ means that propositional variable ϕ must hold in all the following (later) states.

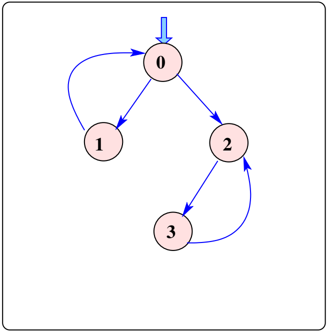
Temporal operators

Common to use mnemonic letters X,G,F,U,R...

- **Operators:**
  - $\mathrm{X}\,\phi$ indicating that ϕ must hold in the next state.
  - $\mathrm{G}\,\phi$ (or $\Box\,\phi$) indicating that ϕ must hold in all the following states.
  - $\mathrm{F}\,\phi$ (or $\Diamond\,\phi$) indicating that eventually (finally) ϕ must hold somewhere.
  - $\phi\,\mathrm{U}\,\psi$ indicating that ϕ has to hold until ψ holds at the current or a future position.
  - $\phi\,\mathrm{R}\,\psi$ The dual of U, ψ holds until the first state where ϕ holds.
- **Quantification**
  - A for all paths
  - E there exists a path...

LTL: Linear time view

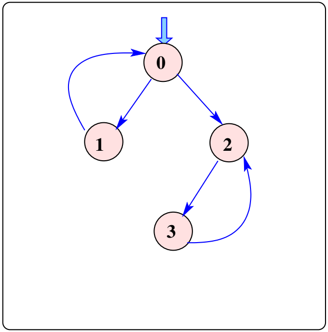
The set of runs...

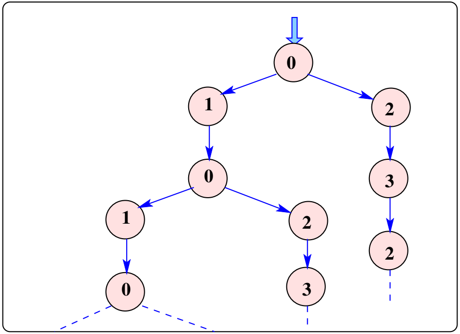


$\{01010101\ldots,\ 01023232\ldots,$

CTL: Branching time view

Branches into the future...





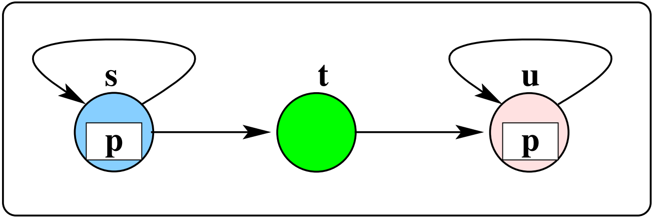
LTL versus CTL

Linear Temporal Logic, Computation Tree Logic

- In LTL, one can encode formulæ about events along a single computation path.
- By contrast, CTL is a modal *branching-time* temporal logic. The operators quantify over all possible future paths from a given state.
- CTL and LTL are both subsets of a more general temporal logic CTL*.
- There are expressions in CTL that cannot be expressed in LTL and *vice versa*.
- In CTL formulæ each of the temporal operators must be preceded by a *path* quantifier: A, or E.

LTL$\neq$CTL

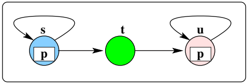
Consider this TS:



- $\mathrm{A}(\mathrm{FG}\,p)$, is an LTL formula representing: for all paths, eventually p holds globally (i.e. from then on).
- $\mathrm{AF}(\mathrm{AG}\,p)$ is CTL for: for all paths, eventually you get to a state where for all paths p holds globally (i.e. from then on).
- LTL formula is not the same thing as CTL formula.

LTL≠CTL

LTL: All runs that start in *s* have *p* holding eventually:



The possible (infinite) runs from s are

$$\left.\textit{sssssssss}\ldots\right\}\quad i.e.\ s^{\infty}$$

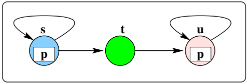
or...

$$\left.\begin{array}{l}\textit{stuuuuuu}\ldots\\ \textit{sstuuuuu}\ldots\\ \textit{ssstuuuu}\ldots\\ \ldots\end{array}\right\}\quad i.e.\ ss^{*}tu^{\infty}$$

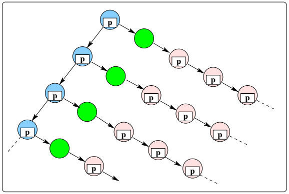
so in the linear time view, for state s, $\mathrm{A}(\mathrm{FG}\,p)$.

LTL$\neq$CTL

CTL: Eventually you get to where *p* holds from then on:



The CTL counterexample for $\mathrm{AF}(\mathrm{AG}\,p)$ is



so in the CTL view, for state *s*, $\mathrm{AF}(\mathrm{AG}\,p)$ is not true.