Principles of Cyber-Physical Systems

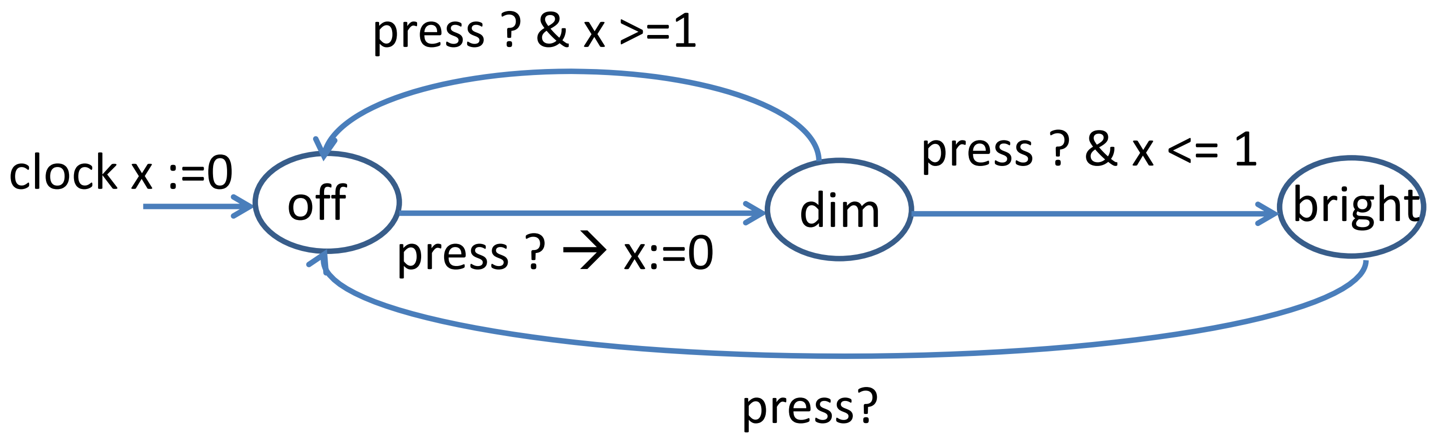
Chapter 8-3: Timed Model

Instructor: Lanshun Nie

Models of Reactive Computation

- Synchronous model
  - Components execute in a sequence of discrete rounds in lock-step
  - Computation within a round: Execute all tasks in an order consistent with precedence constraints
- Asynchronous model
  - Speeds at which different components execute are independent
  - Computation within a step: Execute a single task that is enabled
- Continuous-time model for dynamical system
  - Synchronous, but now time evolves continuously
  - Execution of system: Solution to differential equations
- Timed model
  - Like asynchronous for communication of information
  - Can rely on global time for coordination

Example Timed Model



Initial state = (mode = off, x = 0)

Timed transition: (off, 0) – 0.5 → (off, 0.5)

Input transition: (off, 0.5) – press? → (dim, 0)

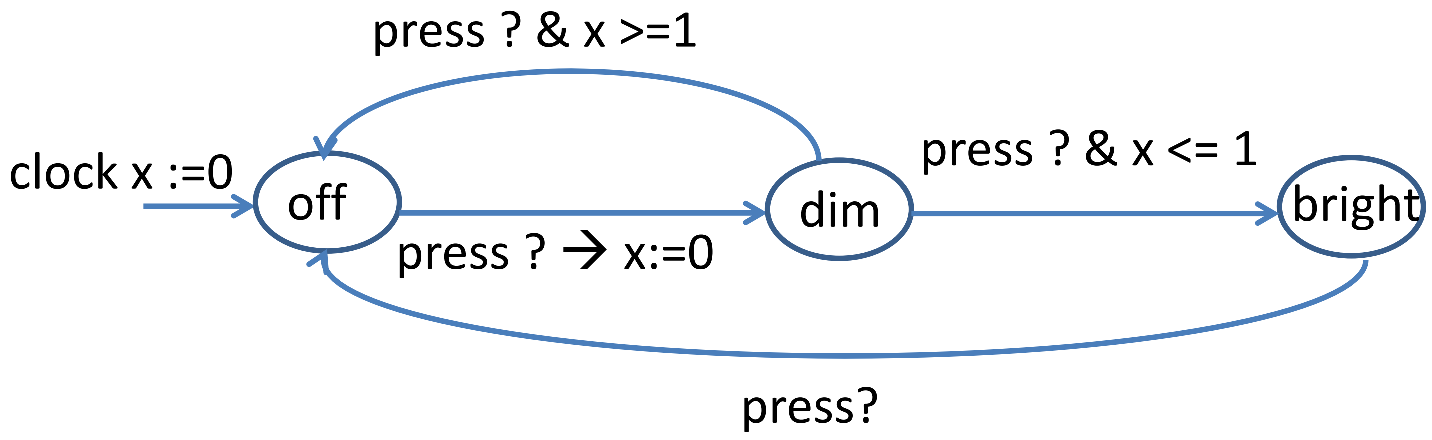
Timed transition: (dim, 0) – 0.8 → (dim, 0.8)

Input transition: (dim, 0.8) – press? → (bright, 0.8)

Timed transition: (dim, 0.8) – 1 → (dim, 1.8)

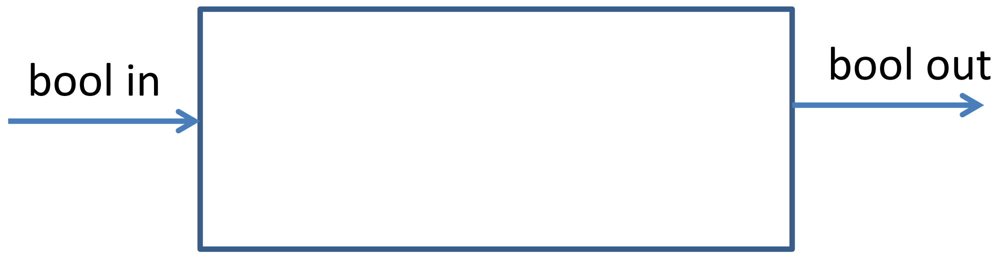
Input transition: (dim, 1.8) – press? → (off, 1.8)

Example Timed Model



- Clock variables
  - Tests and updates in mode-switches like other variables
  - New feature: During a timed transition of duration δ, the value of a clock variable increases by an amount equal to δ
- Timing constraint: Setting x to 0 for "off → dim" and guard x<=1 for "dim → bright" specifies that timing of these two transitions is <= 1 apart

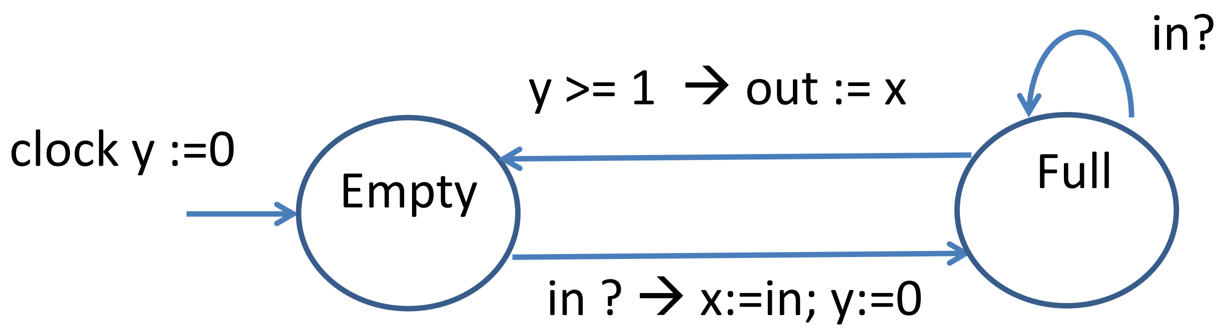
Example: Timed Buffer



- Buffer with a bounded delay

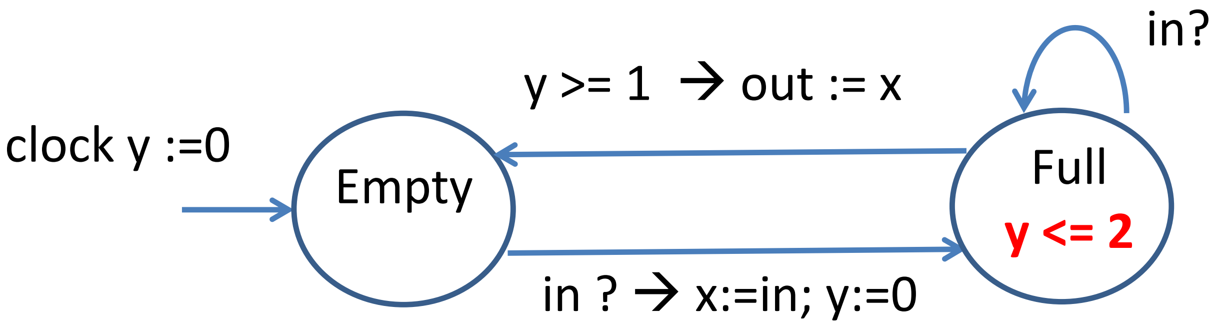
- Behavior to be modeled: Input received on the channel in is transmitted on the output channel after a delay of δ such that LB <= δ <= UB (i.e. we know lower/upper bounds on this delay)

Modeling Timed Buffer



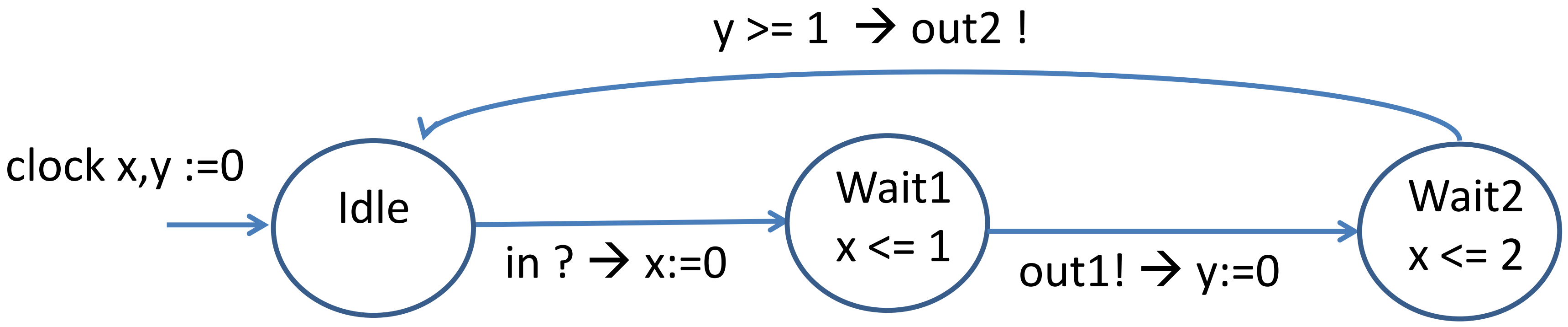
- Mode says whether the buffer is full or not
- State variable x remembers the last input value when buffer is full
- Clock variable y tracks the time elapsed since buffer got full
- When full, input events are ignored
- Guard y >= 1 ensures that at least 1 time unit elapses in mode Full
- How to ensure that mode-switch from Full to Empty is executed before the clock x exceeds the upper bound 2?

Clock Invariants



- The constraint “y <= 2” associated with the mode Full is called “clock invariant”
- A timed transition of duration δ is allowed only when the clock invariant remains true during the entire duration
  - Timed transition (Full, x, y=0.8) – 0.7 → (Full, x, y=1.5) allowed
  - Timed transition (Full, x, y=0.8) – 1.4 → (Full, x, y=2.2) disallowed
- Clock invariants useful to limit how long a process stays in a mode

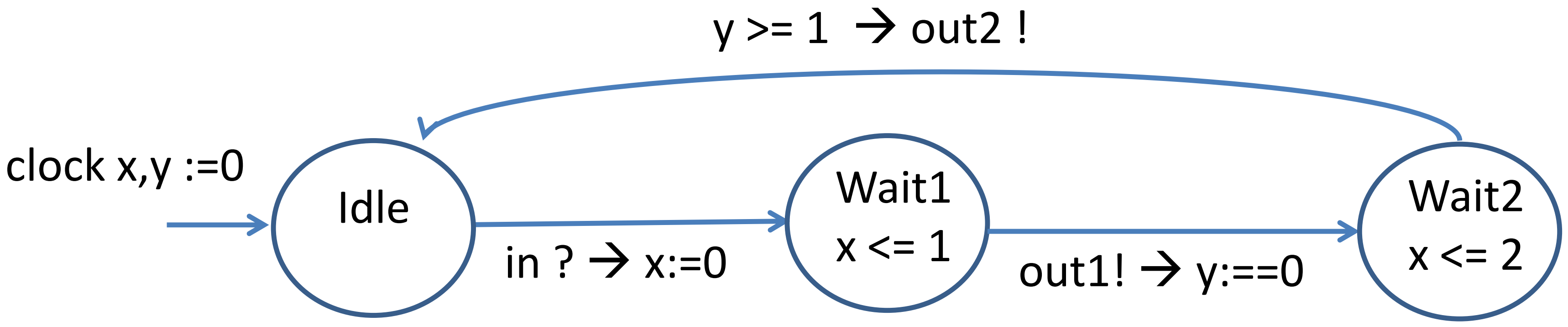
Example with Two Clocks



- Input event: in; Output events out1, out2
- Two clock variables x and y
- In mode Wait2, as time elapses both clocks increase (at same rate)
- Sample timed transitions:

  (Wait2, x=0.8, y=0) -0.3→(Wait2, 1.1, 0.3) -0.72→ (Wait2, 1.82, 1.02)

Two Clock Example



- Clock x tracks time elapsed since the occurrence of input event
- Clock y tracks time elapsed since the occurrence of output event out1
- What is the behavior of this model?
- If input event occurs at time t, the process issues an output on channel out1 at time t' within the interval [t, t+1], and then produces output on channel out2 at time t'' within the interval [t'+1, t+2]

Example Specification

Consider a timed process with

Input event x, Output events y and z

Desired behavior

Whenever it receives input, it produces both its output events

Time delay between x? and y! is in the interval [2, 4]

Time delay between x? and z! is in the interval [3,5]

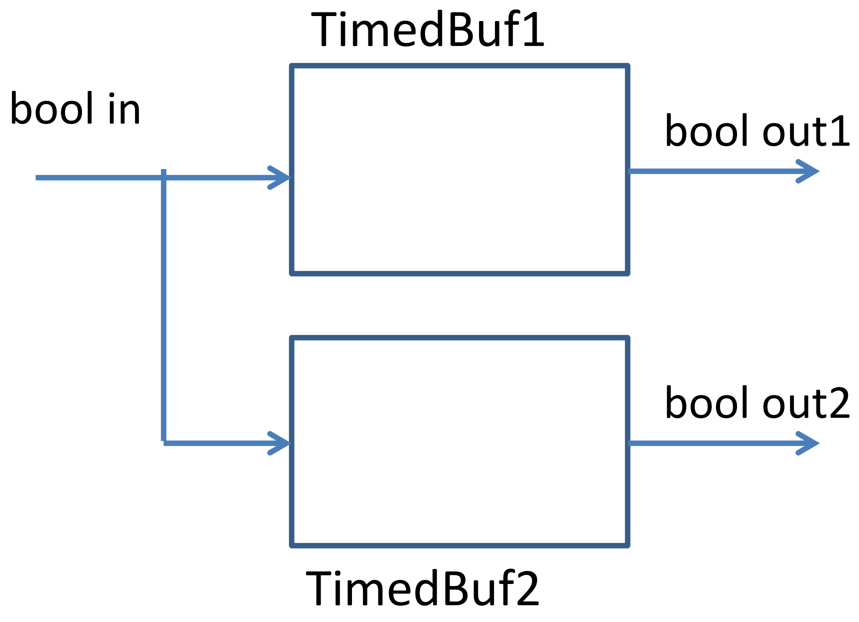
Input events received during this are ignored

Draw a timed state machine that captures this behavior

Definition of Timed Process

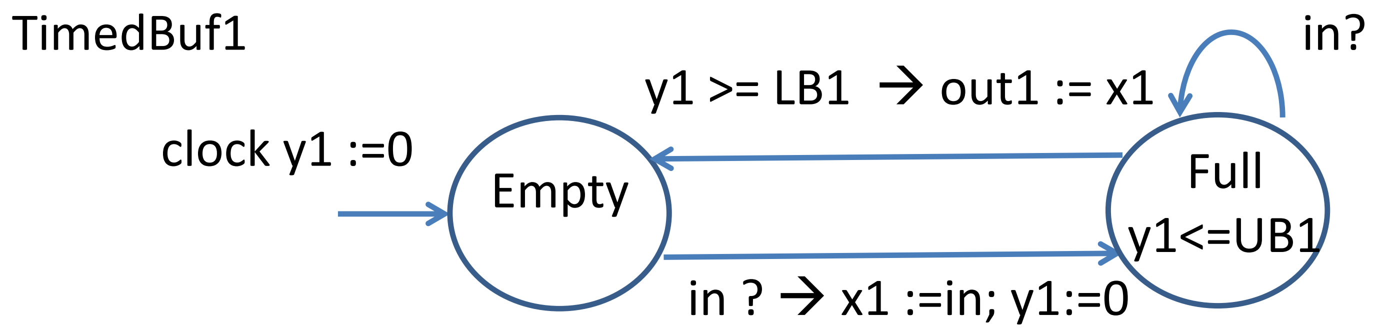
- A timed process TP consists of
  1. An asynchronous process P, where some of the state variables can be of type clock (ranging over non-negative real numbers)
  2. A clock invariant CI which is a Boolean expression over the state variables of P
- Define inputs, outputs, states, initial states, internal actions, input actions, output actions exactly the same as the asynchronous model
- Timed actions: Given a state s and real-valued time $\delta > 0$, $s \xrightarrow{\delta} s+\delta$ is a transition of duration δ if the state s+t satisfies the expression CI for all values of t in the interval $[0, \delta]$
  - For a state s and time t, s+t is another state which assigns the value s(x)+t to every clock variable x, and s(y) to every variable y of a type other than clock
  - If clock-invariant is a convex constraint then it is sufficient to check that the end-states s and $s+\delta$ satisfy CI

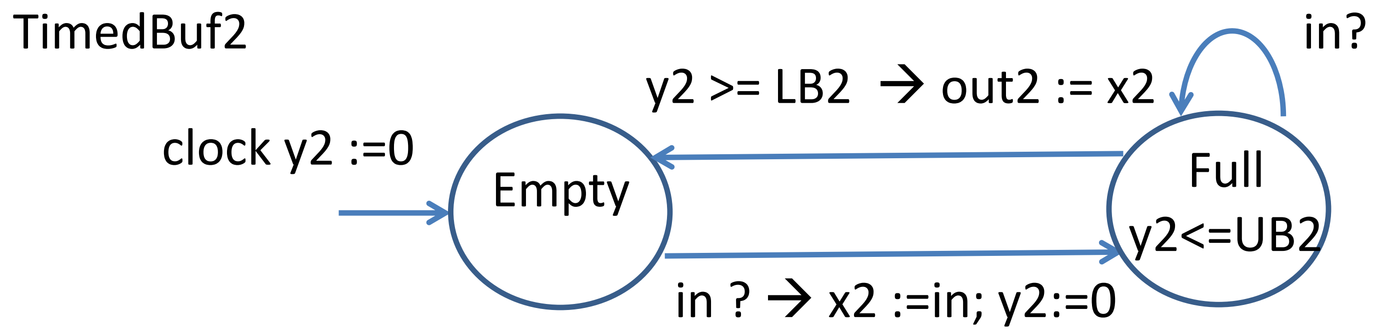
Composition of Processes



- ❑ How to construct timed process corresponding to the composition of the two processes?
- ❑ What are the possible behaviors of the composite process?

Composition of Timed Processes



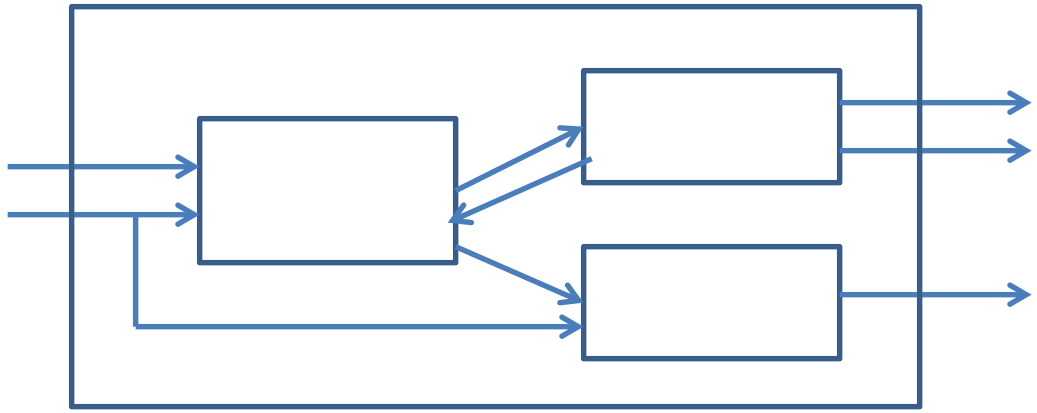


Construct the composite process with four modes

Definition of Parallel Composition

- Consider timed processes TP1 = (P1, CI1) and TP2 = (P2, CI2)
- When is the parallel composition TP1 | TP2 defined?
  - Exactly when the asynchronous parallel composition P1 | P2 is defined (that is, when the outputs of the two are disjoint)
- TP1 | TP2 = (P1 | P2, CI1 & CI2)
  - Asynchronous composition of P1 and P2 defines the internal, input and output actions of the composite
  - Conjunction of the clock-invariants defines the clock-invariant of the composite
- Consequence: The composite process can allow a timed action of duration δ exactly when both TP1 and TP2 are willing to wait for time δ
- Timed model is sometimes called the "semi-synchronous" model (mix of asynchronous and synchronous)

Block Diagrams



- Components can be timed processes now
  - Operations of Instantiation (input/output variable renaming), parallel composition, and Hiding
- A step of the system is either
  1. An internal step of one of components
  2. A communication (input/output) step involving relevant sender and receivers
  3. A timed step involving *all* the components

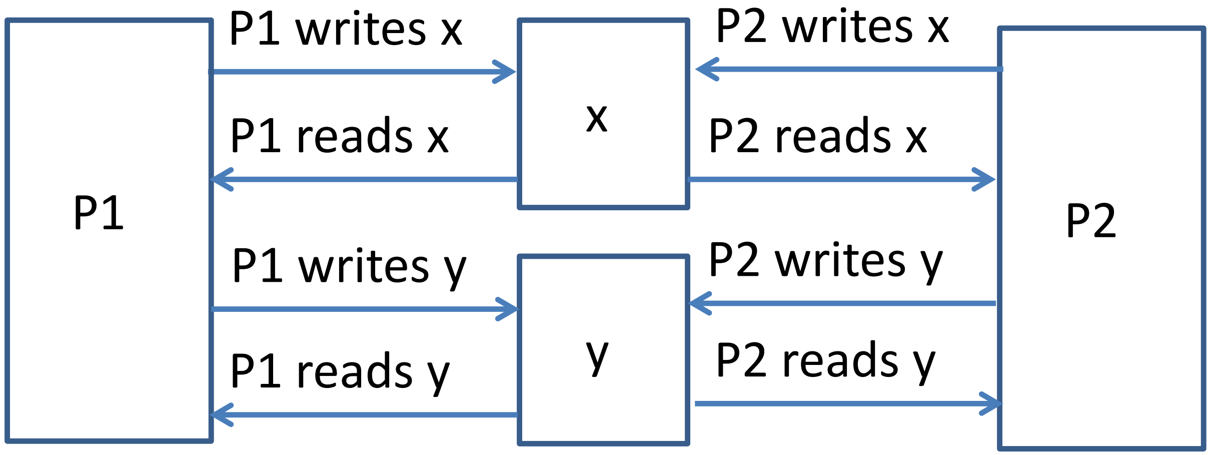
Timed Model

- Timed model is sometimes called the “semisynchronous” model (mix of asynchronous and synchronous)

- Definitions/concepts below carry over in a natural way
  - Executions of a timed process
  - Transition system associated with a timed process
  - Safety/liveness requirements

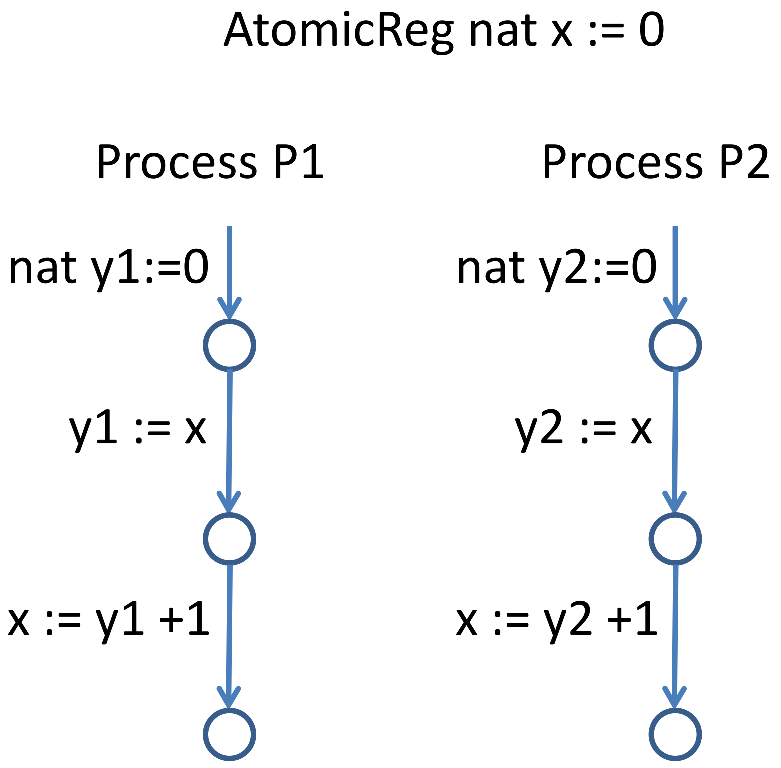
- Distributed coordination problems: how can we exploit the knowledge of timing delays to design protocols?

Recap: Shared Memory Asynchronous Processes



- Processes P1 and P2 communicate by reading/writing shared variables
- Each shared variable can be modeled as an asynchronous process
  - State of each such process is the value of corresponding variable
  - In implementation, shared memory can be a separate subsystem
- Read and write channel between each process and each shared variable
  - To write x, P1 synchronizes with x on "P1 writes x" channel
  - To read y, P2 synchronizes with y on "P2 reads y" channel

Shared Memory Programs with Atomic Registers



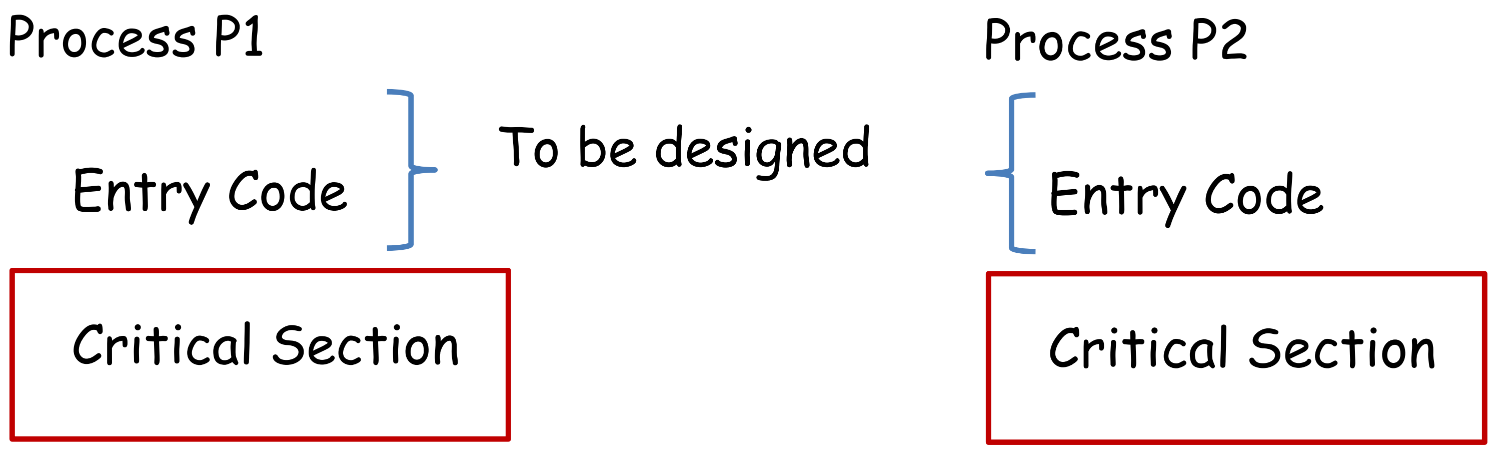
Declaration of shared variables
+ Code for each process

Key restriction: Each statement of a process either
- changes local variables,
- reads a single shared var, or
- writes a single shared var

Execution model: execute one step of one of the processes

What if we knew lower and upper bounds on how long a read or a write takes? Can we solve coordination problems better?

Mutual Exclusion Problem



- Safety Requirement: Both processes should not be in critical section simultaneously (can be formalized using invariants)

- Absence of deadlocks: If a process is trying to enter, then some process should be able to enter

Mutual Exclusion: Attempted Solution

AtomicReg {0, 1, 2} Turn := 0

Process P1

$$\begin{tikzcd}
{} \arrow[r] & \text{Idle} \arrow[loop above] \arrow[r] & \text{Try1} \arrow[loop above, "\text{else}"] \arrow[r, "\text{Turn=0 ?}"] & \text{Try2} \arrow[r, "\text{Turn : = 1}"] & \text{Crit} \arrow[loop above] \arrow[lll, bend left, "\text{Turn := 0}"]
\end{tikzcd}$$

What's the bug?

Process P2

$$\begin{tikzcd}
{} \arrow[r] & \text{Idle} \arrow[loop above] \arrow[r] & \text{Try1} \arrow[loop above, "\text{else}"] \arrow[r, "\text{Turn=0 ?}"] & \text{Try2} \arrow[r, "\text{Turn : = 2}"] & \text{Crit} \arrow[loop above] \arrow[lll, bend left, "\text{Turn := 0}"]
\end{tikzcd}$$

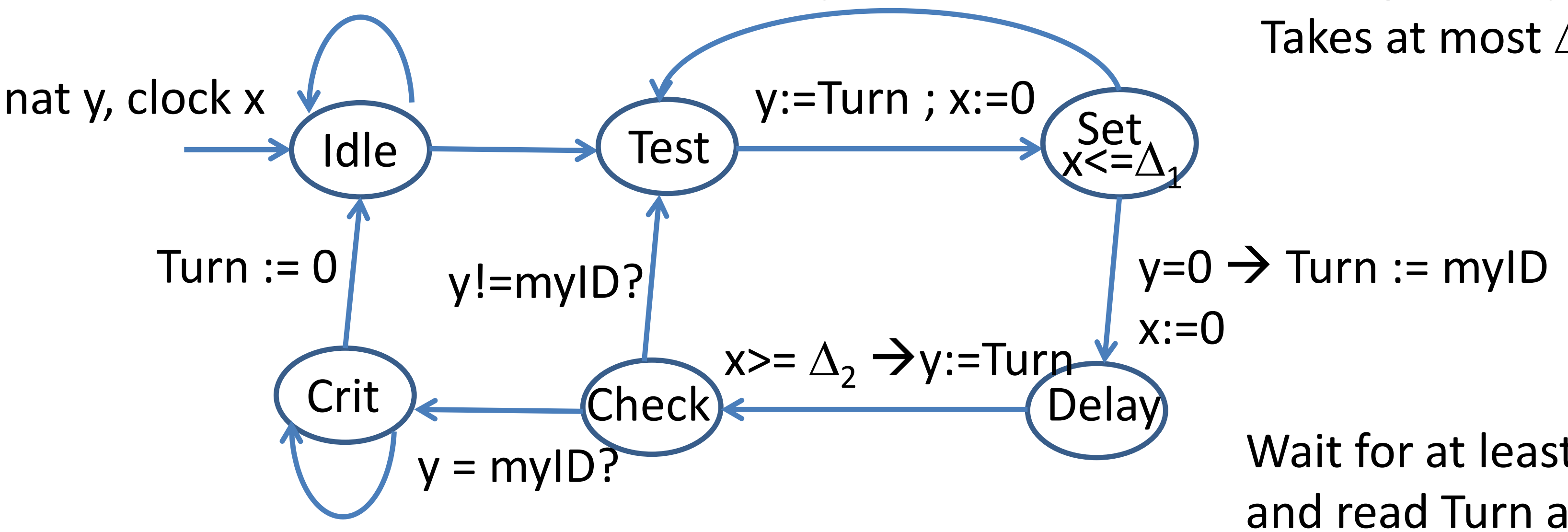
Timing-based Mutual Exclusion

1. When a process wants to enter critical section, it reads the shared variable Turn
2. If Turn != 0 then try again (goto step 1)
3. If Turn = 0 then set Turn to your ID
4. Proceeding directly to critical section is a problem (since the other process may also have concurrently read Turn to be 0, and updating Turn to its own ID)
5. Solution: Delay and wait till you are sure that concurrent writes are finished
6. Read Turn again: if Turn equals your own ID then proceed to critical section, if not goto step 1 and try again
7. When done with critical section, set Turn back to 0

Fisher's Protocol for Mutual Exclusion

AtomicReg Turn := 0

Timing assumption:
Takes at most Δ_1 to write



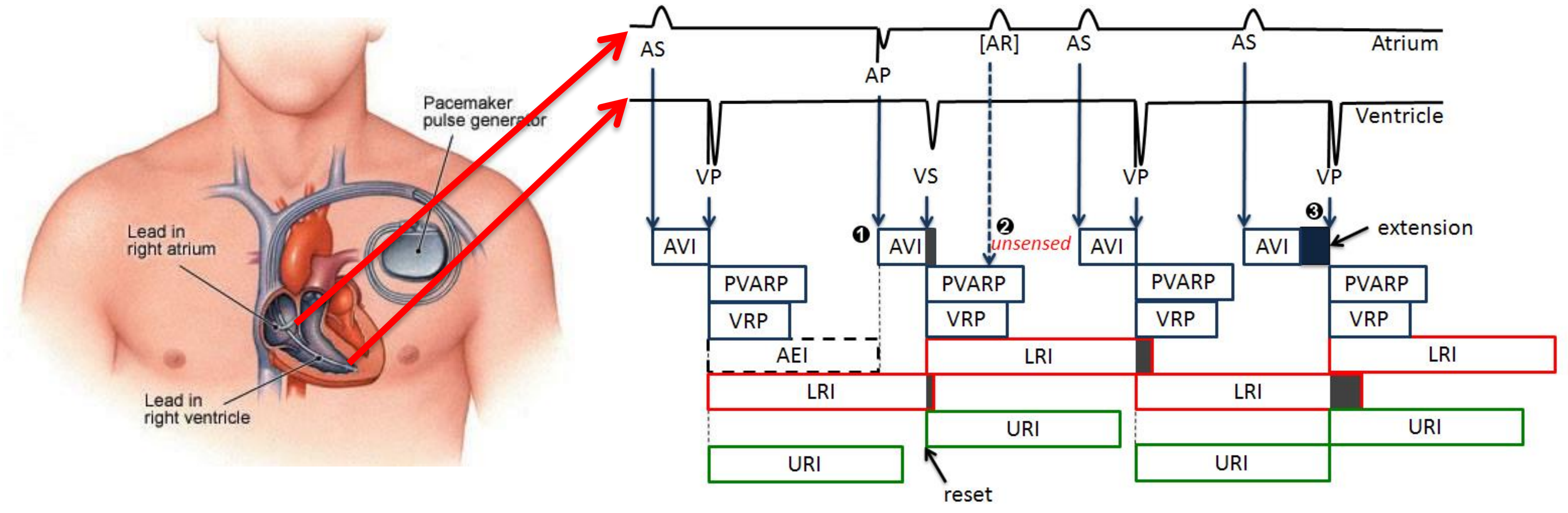
Wait for at least Δ_2 time, and read Turn again

Why does it work ?

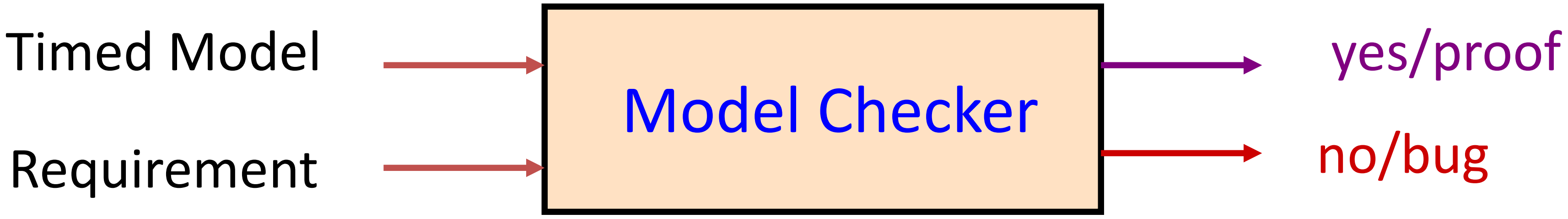
Properties of Fisher's Protocol

- Assuming $\Delta_2 > \Delta_1$, the algorithm satisfies:
  - Mutual exclusion: Two processes cannot be in critical section simultaneously
  - Deadlock freedom: If a process wants to enter critical section then some process will enter critical section
- Protocol works for arbitrarily many processes (not just 2)
  - Note: In the asynchronous model, mutual exclusion protocol for N processes is lot more complex than Peterson's algorithm
- Does the protocol satisfy the stronger property of starvation freedom: If a process P_i wants to enter critical section then it eventually will ?
- If $\Delta_2 <= \Delta_1$ then does mutual exclusion hold? Does deadlock freedom hold?

Implantable Pacemaker Modeling

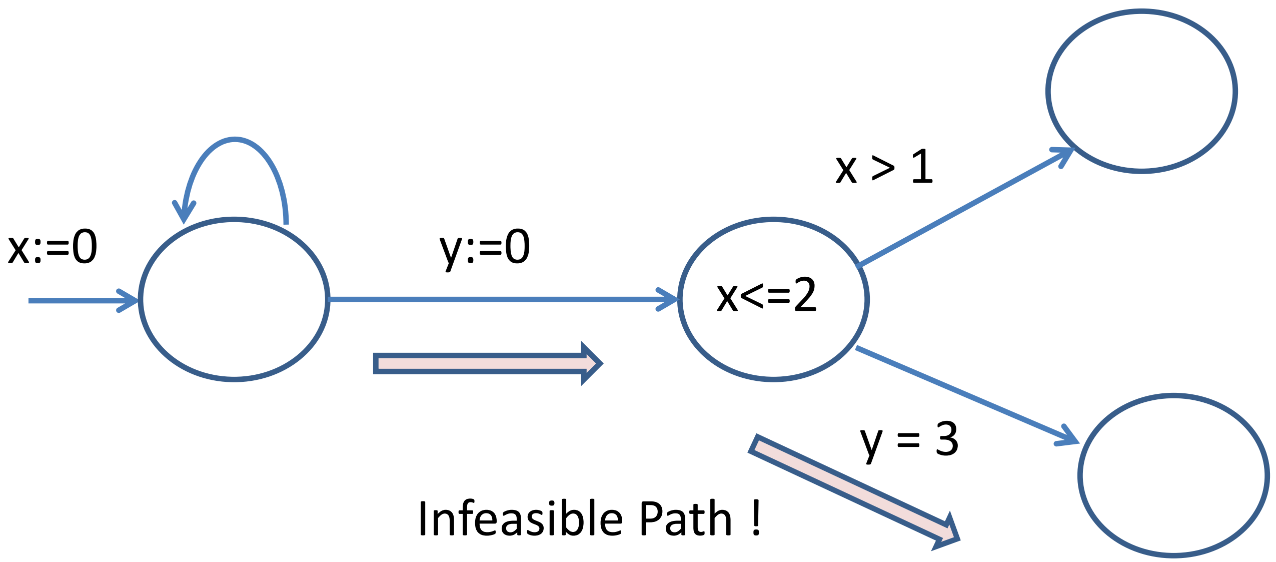


Timing Analysis



- How to adapt algorithms for searching through the state-space of a model in presence of clock variables and timing constraints?
- Application: Formal analysis of timing-based coordination and communication protocols
- Must handle the space of clock valuations symbolically!
- Popular model checker: Uppaal

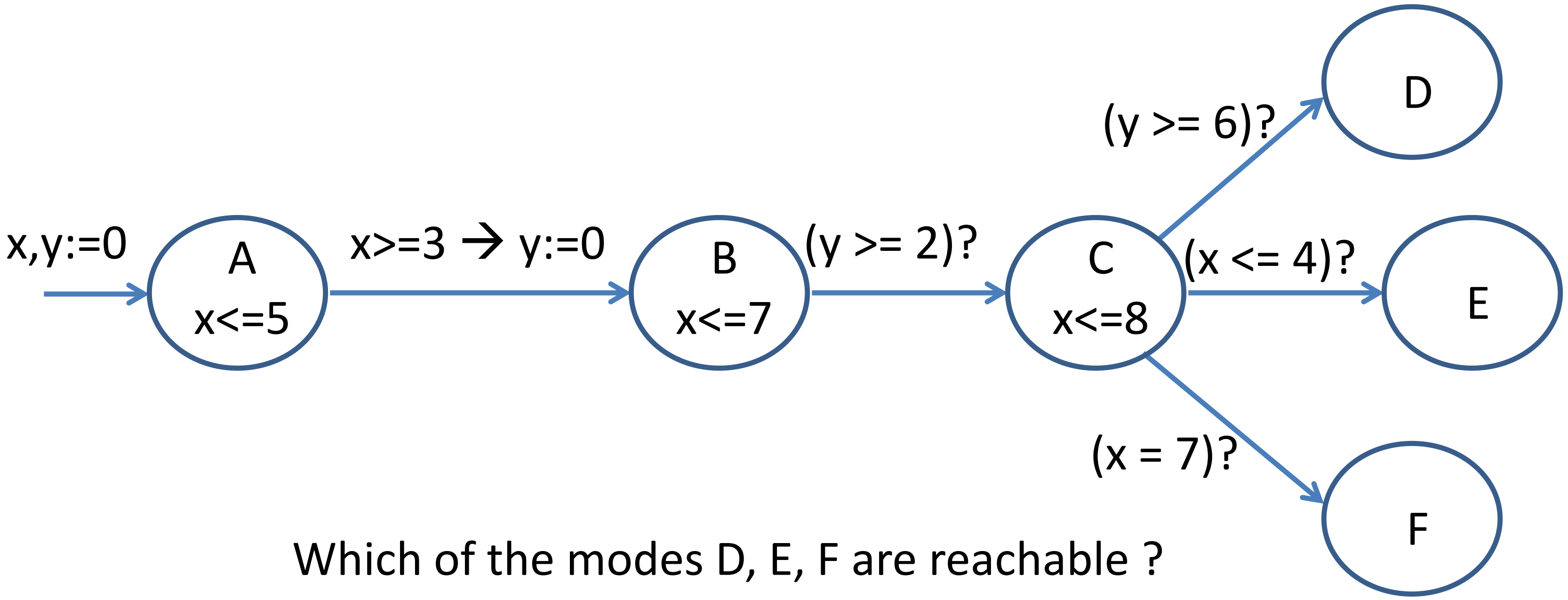
Timing Analysis Example



Timed Automata

- Motivation: When is exact analysis of timing constraints possible?
- A timed process TP is a timed automaton if for every clock variable x
  - Assignments to x in the description of TP are of the form x:=0
  - An atomic expression involving x in the description of TP (in clock-invariants or in guards) must of the form "x~ k", where k is a constant and ~ is a comparison operation (=, <=, >, <, >=)
- In such a model, one can express constant lower and upper bounds on timing delays
  - Closed under parallel composition: If TP1 and TP2 are timed automata then TP1 | TP2 is also a timed automaton
- Finite-state timed automaton: A timed automaton where all variables other than clock variables have finite types (e.g. Boolean, enumerated)
  - State-space is still infinite due to clock variables, but verification is solvable exactly

Timing Analysis Example

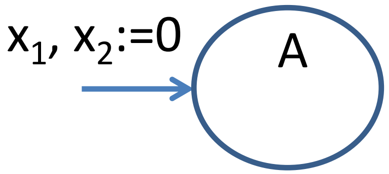


Which of the modes D, E, F are reachable ?

Requires propagation of the reachable combinations of x and y symbolically

Timing Analysis Example

Clock-zone R_0
$0 <= x_1 <= 0$
$0 <= x_2 <= 0$
$0 <= x_1 - x_2 <= 0$



Initial set of clock-valuations: $x_1 = 0$ & $x_2 = 0$

Clock-zone: Uniform representation of constraints that arise during analysis

Constraints of two types:

1. Lower/upper bound on value of a clock variable
2. Lower/upper bound on difference of two clock variables

Timing Analysis Example

Clock-zone R_0

$0 <= x_1 <= 0$

$0 <= x_2 <= 0$

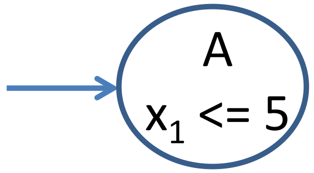
$0 <= x_1 - x_2 <= 0$

Clock-zone R'_0

$0 <= x_1 <=$ Infty

$0 <= x_2 <=$ Infty

$0 <= x_1 - x_2 <= 0$



Starting from a state in R_0, as time elapses, which clock-valuations are reachable ?

During a timed transition, values of all clocks increase.
How are the constraints impacted? What's the effect of clock-invariant ?

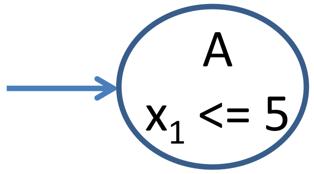
Step 1: Compute effect of timed transitions ignoring clock-invariants

Constraints on individual clocks: Change upper bound to Infty

Constraints on differences between clock values: unchanged (why?)

Timing Analysis Example

Clock-zone R'_0	Clock-zone R''_0	Clock-zone R_1
$0 <= x_1 <=$ Infty	$0 <= x_1 <= 5$	$0 <= x_1 <= 5$
$0 <= x_2 <=$ Infty	$0 <= x_2 <=$ Infty	$0 <= x_2 <= 5$
$0 <= x_1-x_2 <= 0$	$0 <= x_1-x_2 <= 0$	$0 <= x_1-x_2 <= 0$



Desired clock-zone R_1: Set of clock-valuations reachable while in mode A

Intersection of constraints in R'_0 and the clock-invariant

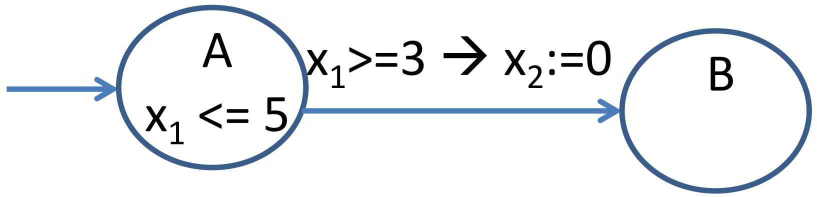
Canonicalization: Tighten all bounds to reflect implied constraints

Each lower bound should be as high as possible

Each upper bound should be as low as possible

Timing Analysis Example

Clock-zone R_1			Clock-zone R_2
$0 <= x_1 <= 5$	$3 <= x_1 <= 5$	$3 <= x_1 <= 5$	$3 <= x_1 <= 5$
$0 <= x_2 <= 5$	$0 <= x_2 <= 5$	$3 <= x_2 <= 5$	$0 <= x_2 <= 0$
$0 <= x_1-x_2 <= 0$	$0 <= x_1-x_2 <= 0$	$0 <= x_1-x_2 <= 0$	$3 <= x_1-x_2 <= 5$



Desired clock-zone R_2: What are set of clock-valuations upon entry to B ?

Step 1: Intersect guard $3 <= x_1$ with the clock-zone R_1

Step 2: Canonicalize by tightening constraints

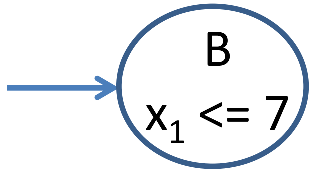
Step 3: Capture the effect of assignment $x_2 := 0$

Bounds on x_2 change, and so do bounds on x_1-x_2

Step 4: Canonicalize. In this case, constraints are already as tight as possible

Timing Analysis Example

Clock-zone R_2			Clock-zone R_3
$3 <= x_1 <= 5$	$3 <= x_1 <= \text{Infty}$	$3 <= x_1 <= 7$	$3 <= x_1 <= 7$
$0 <= x_2 <= 0$	$0 <= x_2 <= \text{Infty}$	$0 <= x_2 <= \text{Infty}$	$0 <= x_2 <= 4$
$3 <= x_1-x_2 <= 5$	$3 <= x_1-x_2 <= 5$	$3 <= x_1-x_2 <= 5$	$3 <= x_1-x_2 <= 5$



Starting from a state in R_0, as time elapses, which clock-valuations are reachable ?

Step 1: Set upper bounds on individual clock values to Infty

Step 2: Intersect with the clock-invariant $x_1 <= 7$

Step 3: Canonicalize by tightening all the bounds

- What is a good data structure to represent clock-zones?
- What are algorithms for operations such as intersection, canonicalization?

Difference Bounds Matrix

- Data structure for representing constraints, where each constraint expresses a bound on difference of values of two variables
- Suppose clocks are named $x_1, x_2, \ldots x_m$
- Let us introduce a dummy clock x_0 that is always 0. Then instead of the constraint $L <= x_i <= U$, we have $L <= x_i - x_0 <= U$
- Lower bound constraint $L <= x_i - x_j$ can be rewritten as upper bound constraint $x_j - x_i <= -L$
- DBM R is (m+1) x (m+1) matrix representing

  for $0 <= i <= m$, for $0 <= j <= m$, $x_i - x_j <= R[i,j]$
- Diagonal entries should be 0: $x_i - x_j <= 0$
- There is a one-to-one correspondence between DBMs and clock-zones
- Entries of DBM: Integers plus a special symbol Infty (to represent absence of a bound)

DBM Representation of Constraints

$3 <= x_1 <= 7$
$0 <= x_2 <= \text{Infty}$
$3 <= x_1 - x_2 <= 5$

$3 <= x_1 - x_0 <= 7$
$0 <= x_2 – x_0 <= \text{Infty}$
$3 <= x_1 - x_2 <= 5$

$x_1 - x_0 <= 7$
$x_0 – x_1 <= -3$
$x_2 - x_0 <= \text{Infty}$
$x_0 – x_2 <= 0$
$x_1 - x_2 <= 5$
$x_2 – x_1 <= -3$

	X0	X1	X2
X0	0	-3	0
X1	7	0	5
X2	Infty	-3	0

DBM Canonicalization

	X0	X1	X2
X0	0	-3	0
X1	7	0	5
X2	Infty	-3	0

	X0	X1	X2
X0	0	-3	0
X1	7	0	5
X2	4	-3	0

Clock-zone R_3

$3 <= x_1 <= 7$

$0 <= x_2 <= 4$

$3 <= x_1 - x_2 <= 5$

We know: $x_2 - x_1 <= -3$ and $x_1 - x_0 <= 7$, so we can conclude $x_2 - x_0 <= 4$

But current R[2,0] entry is Infty, to reflect tighter implied constraint, change it to 4

General canonicalization step:

If R[i,k] > R[i,j] + R[j,k] then R[i,k] = R[i,j] + R[j,k]

Canonicalization: repeatedly apply above rule.

A matrix is called canonical if all i, j, k, R[i,k] <= R[i,j] + R[j,k]

Canonical DBMs

- Every canonicalization step does not change the set of clock-valuations that the DBM represents
  - Canonical DBM represents the "tightest" possible constraints
- If R is canonical, for every i and j, R[i,j] is the least upper bound on the difference $x_i - x_j$ for clock-valuations satisfying constraints in R
- Alternative interpretation
  - Consider a graph with m+1 vertices corresponding to $x_0, x_1, \ldots x_m$
  - Entry R[i,j] = Cost labeling the edge from vertex x_i to x_j
  - Canonicalization: Compute costs of shortest paths in this graph
  - R is canonical if R[i,j] is the cost of shortest path from x_i to x_j
- What if constraints are unsatisfiable ?

Checking Satisfiability/Non-emptiness

	X0	X1	X2
X0	0	0	0
X1	5	0	10
X2	Infty	-6	0

	X0	X1	X2
X0	0	-6	0
X1	5	0	10
X2	Infty	-6	0

	X0	X1	X2
X0	0	-6	0
X1	5	-1	10
X2	Infty	-6	0

Suppose we know: $0 <= x_1 <= 5$; $0 <= x_2 <=$ Infty; $6 <= x_1 – x_2 <= 10$

Canonicalization step 1: Compare R[0,1] to R[0,2] + R[2,1], and change to -6

Canonicalization step 2: Compare R[1,1] to R[1,0] + R[0,1], and change to -1

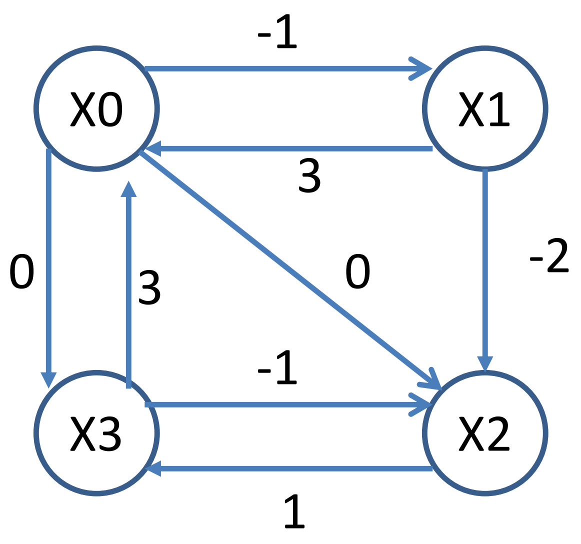
R[1,1] entry says $x_1 – x_1 <= -1$, not possible! Means original constraints unsatisfiable!

Graph-based representation

Given: $1 <= x_1 <= 3$; $x_2 >= 0$; $0 <= x_3 <= 3$; $x_2 – x_3 = 1$; $x_2 – x_1 >= 2$

	X0	X1	X2	X3
X0	0	-1	0	0
X1	3	0	-2	infty
X2	infty	infty	0	1
X3	3	Infty	-1	0

	X0	X1	X2	X3
X0	0	-1	-3	-2
X1	2	0	-2	-1
X2	4	3	0	1
X3	3	2	-1	0



Canonicalization Algorithm

- Problem same as computing shortest paths among all pairs of vertices in a directed graph

- Need to account for detection of negative cycles (that is, unsatisfiable constraints)

- Classical algorithm with time complexity cubic in the number of vertices/clocks

Canonicalization Algorithm

Input: (m+1) x (m+1) DBM R

Output: Check if constraints represented by R are satisfiable, and if so, return "canonical" version of R

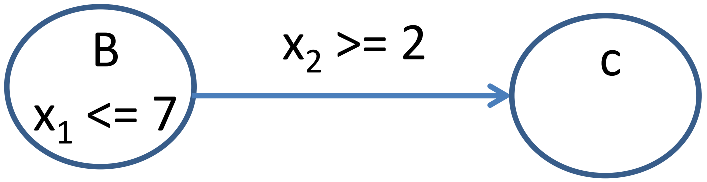
```
for l = 0 to m {
 /* R[i,j] reflects shortest path going through vertices < l */
        for i = 0 to m {
            for j = 0 to m { /*Check if visiting vertex l improves i→j path */
                R[i, j] = min ( R[i,j], R[i,l] + R[l,j] )
            };
            if R[i, i] < 0 then return "Unsatisfiable";
        };
};
return R
```

Timing Analysis Example

	X0	X1	X2
X0	0	-3	0
X1	7	0	5
X2	4	-3	0

	X0	X1	X2
X0	0	0	-2
X1	Infty	0	Infty
X2	Infty	Infty	0

	X0	X1	X2
X0	0	-3	-2
X1	7	0	5
X2	4	-3	0



	X0	X1	X2
X0	0	-5	-2
X1	7	0	5
X2	4	-3	0

Goal: Compute set of clock-valuations upon entry to C

Step 1: Represent guard condition $x_2 >= 2$ as a canonical DBM

Step 2: Intersect the two DBMs

[i,j] entry of the result = Minimum of [i,j] entries of the given DBMs

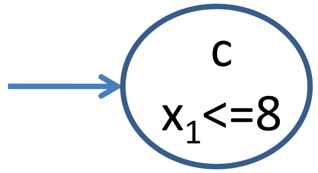
Step 3: Canonicalize.

Timing Analysis Example

	X0	X1	X2
X0	0	-5	-2
X1	7	0	5
X2	4	-3	0

	X0	X1	X2
X0	0	-5	-2
X1	Infty	0	5
X2	Infty	-3	0

	X0	X1	X2
X0	0	-5	-2
X1	8	0	5
X2	Infty	-3	0



	X0	X1	X2
X0	0	-5	-2
X1	8	0	5
X2	5	-3	0

Goal: Compute set of clock-valuations reachable due to timed transitions in mode C

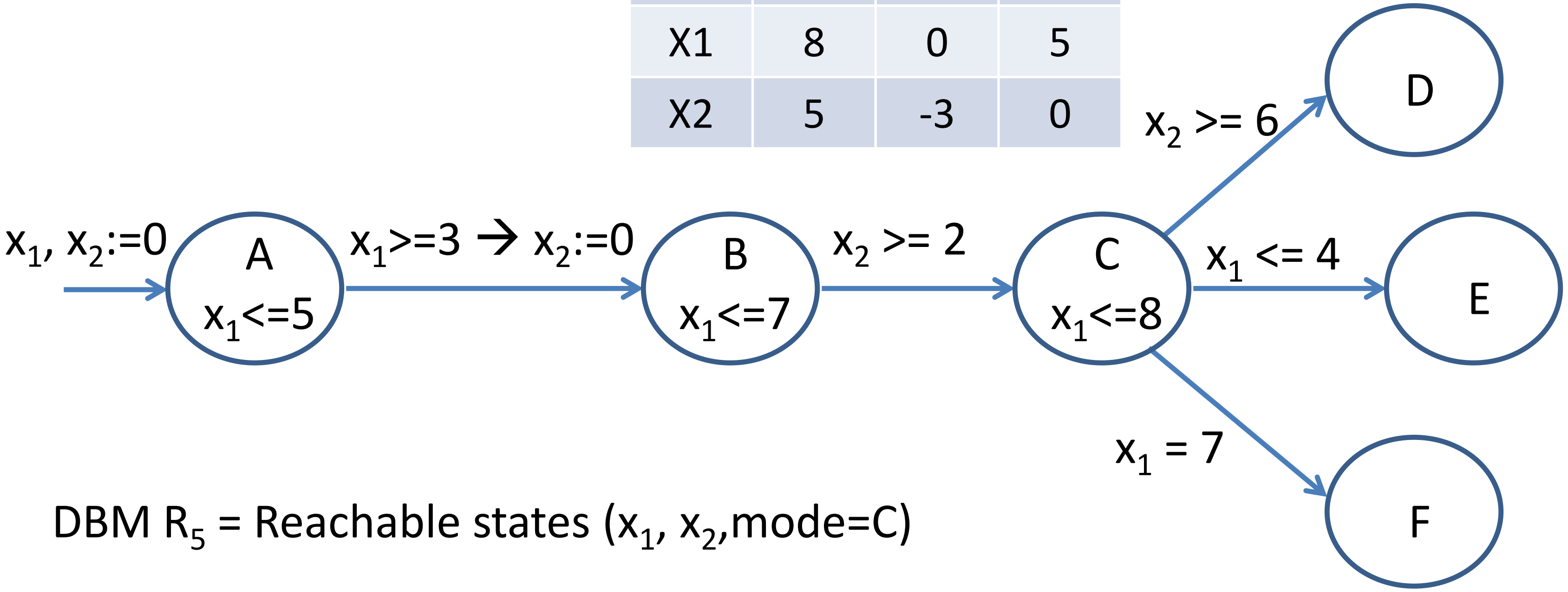
Step 1: Set upper bounds on all clocks, that is, first column, other than X0, to Infty

Step 2: Represent clock-invariant of C as a DBM, and take intersection

Step 3: Canonicalize.

Timing Analysis Example

	X0	X1	X2
X0	0	-5	-2
X1	8	0	5
X2	5	-3	0



DBM R_5 = Reachable states (x_1, x_2,mode=C)

Intersection of R_5 and $x_2 >= 6$ unsatisfiable; means mode D not reachable

Intersection of R_5 and $x_1 <= 4$ unsatisfiable; means mode E not reachable

Intersection of R_5 and $x_1 = 7$ satisfiable; means mode F is reachable

Symbolic BFS Algorithm

Given region Init over S, region Trans over S U S', and region φ over S, if φ is reachable in T then return 1, else return 0

```
reg Reach := Init; /* States found reachable */
reg New := Init; /* States not yet explored for outgoing transitions */
while IsEmpty(New) = 0 {   /* while there are states to be explored */
   if IsEmpty(Conj(New,φ)) =0  /* Property φ found reachable */
   then return 1 (and stop);
   New := Diff(Post(New,Trans),Reach);
               /*These are states in post-image of New, but not
                     previously found reachable, so to be explored */
   Reach := Disj(Reach, New); /* Update Reach by newly found states*/
};
return 0; /* All states explored without encountering φ */
```

Symbolic Search Using DBMs

- State of a timed automaton = (Discrete state, clock valuation)
  - Discrete state assigns values to all variables other than clocks
- Symbolic representation = (Discrete state s, DBM R)
- Operations on DBMs used to implement symbolic search
- When do (s, R) and (s', R') represent same set of states?
  - s = s' and R = R' (assuming they are canonical)
- Image computation corresponding to timed transitions: already studied
- Image computation corresponding to mode-switches
  - Intersect with guard, and reset clocks as needed
  - What's the algorithm to compute, given DBM R, DBM for R[x_i := 0]
- Cannot implement "Union" operation directly on DBMs
  - Union of DBMs R and R' need not be a DBM, as each DBM represents a convex set of valuations

Symbolic Search Using DBMs

- Data structure: discrete state and a list of DBMs (s, R_1, R_2, …)
- Search is partly enumerative and partly symbolic
- Multiple paths ending up at same discrete state s would cause the list of DBMs associated with s to grow
  - Efficient implementation needs a way to keep this list compact
- If number of reachable discrete states is finite, then search algorithm is guaranteed to terminate
- Alternative algorithm for analyzing timed automata
  - Region equivalence: Finite partitioning of the set of all possible clock-valuations such that valuations belonging to the same partition behave similarly
  - See Section 7.3.2 for details