CS:4980 Foundations of Embedded Systems

Timed Model Part II

Timed Model

Timed model is sometimes called the *semi-synchronous* model (mix of asynchronous and synchronous)

Definitions/concepts that carry over naturally from those models:

- Executions of a timed process
- Transition system associated with a timed process
- Safety/liveness requirements

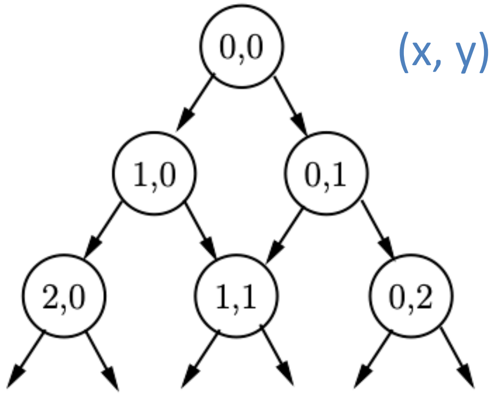
Distributed coordination problems: how can we exploit the knowledge of timing delays to design protocols?

Recall: Asynchronous Execution Model

```
nat x := 0 ; y := 0
```

A_x : $x := x + 1$

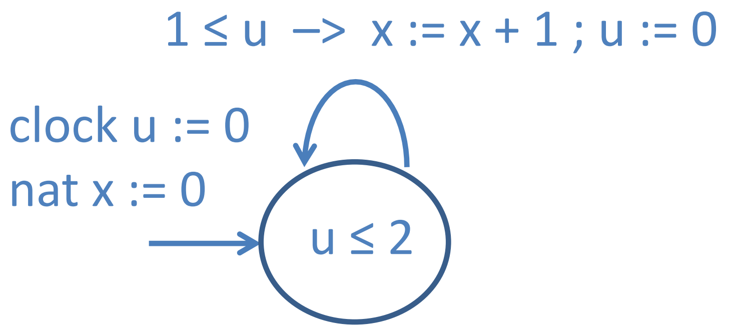
A_y : $y := y + 1$

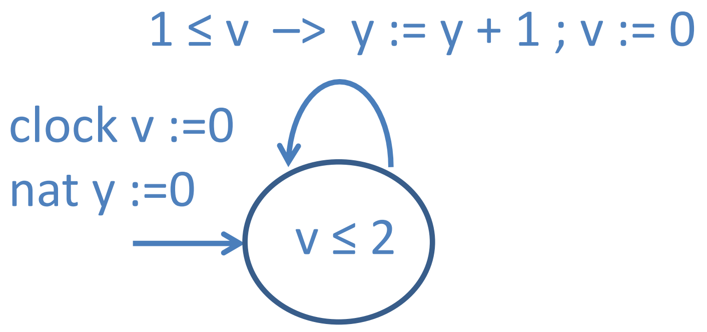


- Tasks A_x and A_y execute in an arbitrary order
- For every possible choice of numbers m and n, the state (m, n) is reachable
- Fairness assumptions can be used to rule out executions where one of the tasks is ignored forever
  (although this does not affect the set of reachable states)

What if we know how long each of these increments take?

Timed Increments

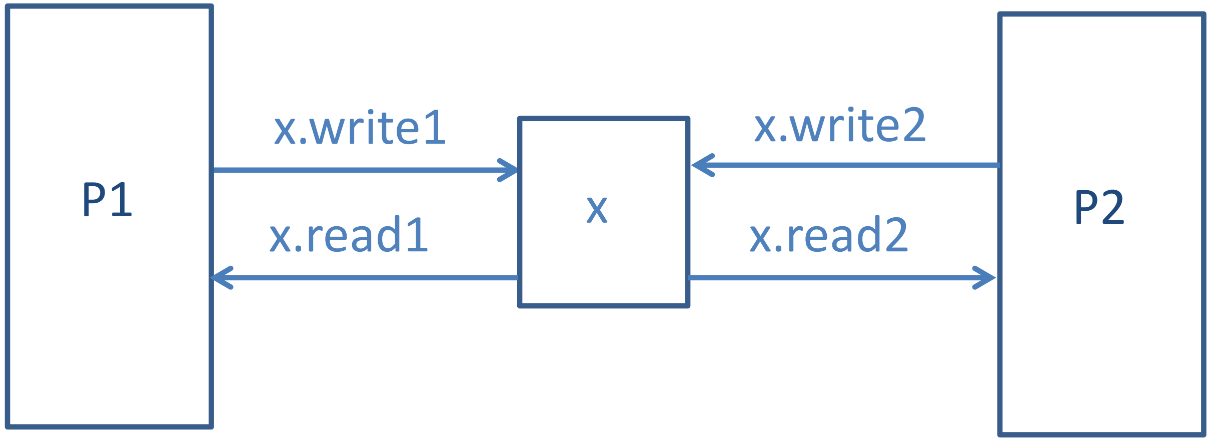




- ❑ Task A_x increments x, and this takes between 1 to 2 time units
- ❑ Task A_y increments y, and this also takes between 1 to 2 time units
- ❑ Tasks execute in parallel, asynchronously, but timing introduces loose coordination (since all clocks advance in unison)

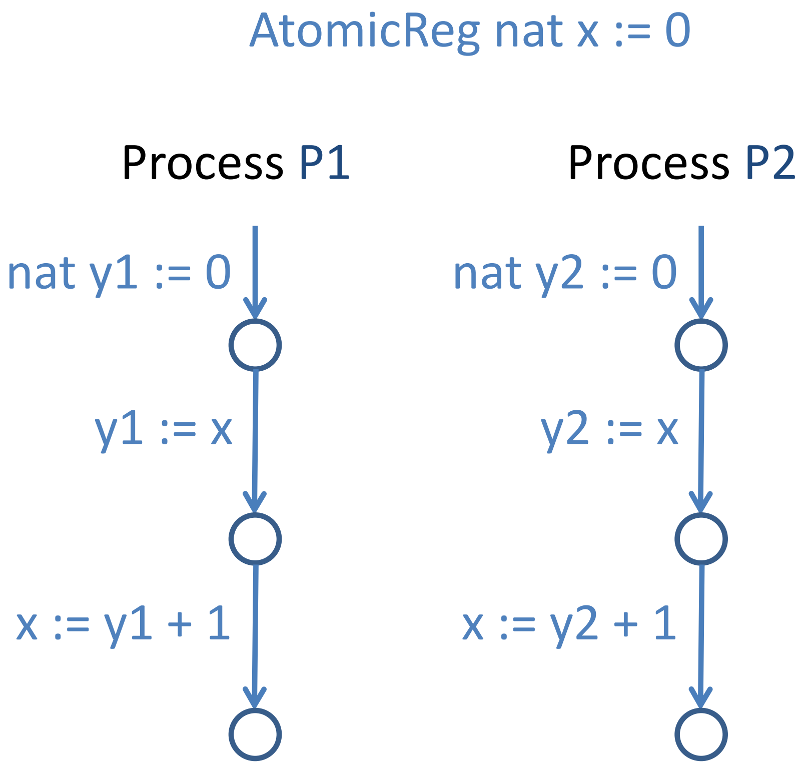
Which states are reachable? What is the relationship between m and n so that the state (m, n) is reachable?

Recall: 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 x.write1 channel
    - To read x, P2 synchronizes with x on x.read2 channel

Shared Memory Programs with Atomic Registers



Declaration of shared variables

+ code for each process

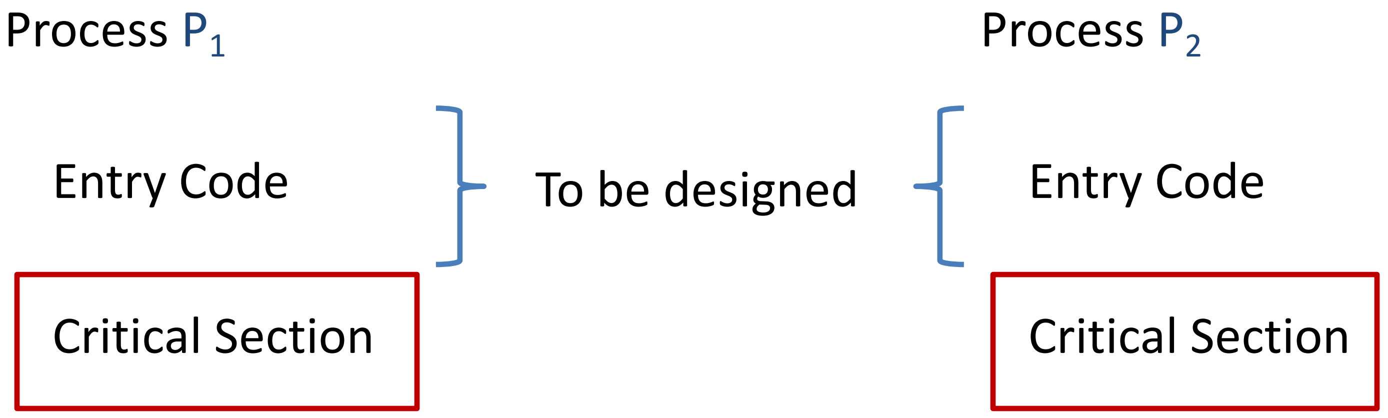
Key restriction: Each statement either

- 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? Could we solve coordination problems better?

Mutual Exclusion Problem



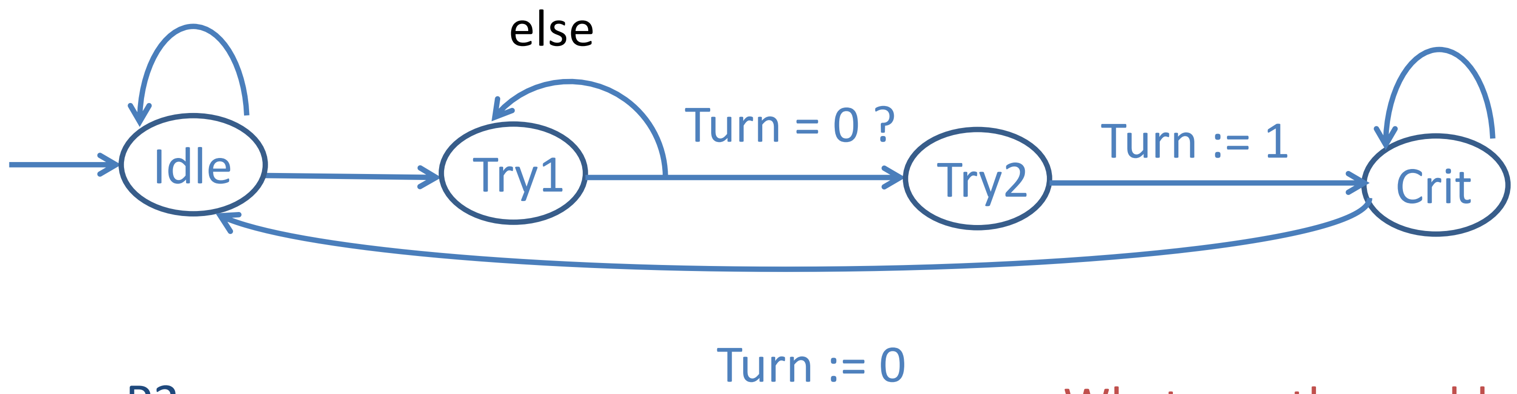
Safety: processes should not both be in critical section simultaneously (can be formalized using invariants)

Deadlock freedom: if any process is trying to enter, then some process should be able to enter

Mutual Exclusion: Incorrect Solution

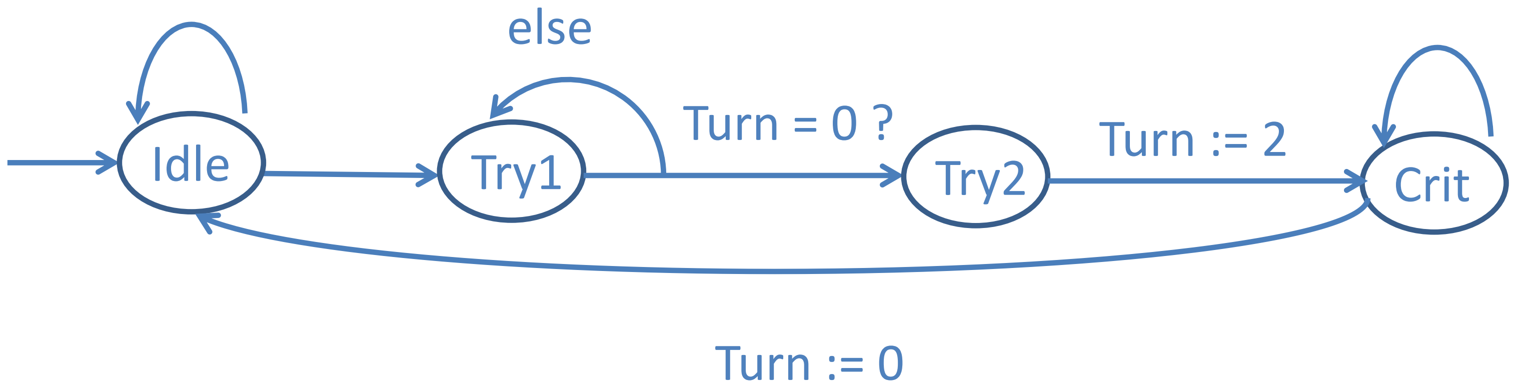
AtomicReg {0, 1, 2} Turn := 0

Process P1



Process P2

What was the problem?



Timing-based Mutual Exclusion

1. Before entering critical section, read the shared variable Turn
2. If Turn ≠ 0 then go to Step 1 and try again
3. If Turn = 0 then set Turn to your ID

Problem: Proceeding directly to critical section (the other process may also have concurrently read Turn to be 0, and updated Turn)

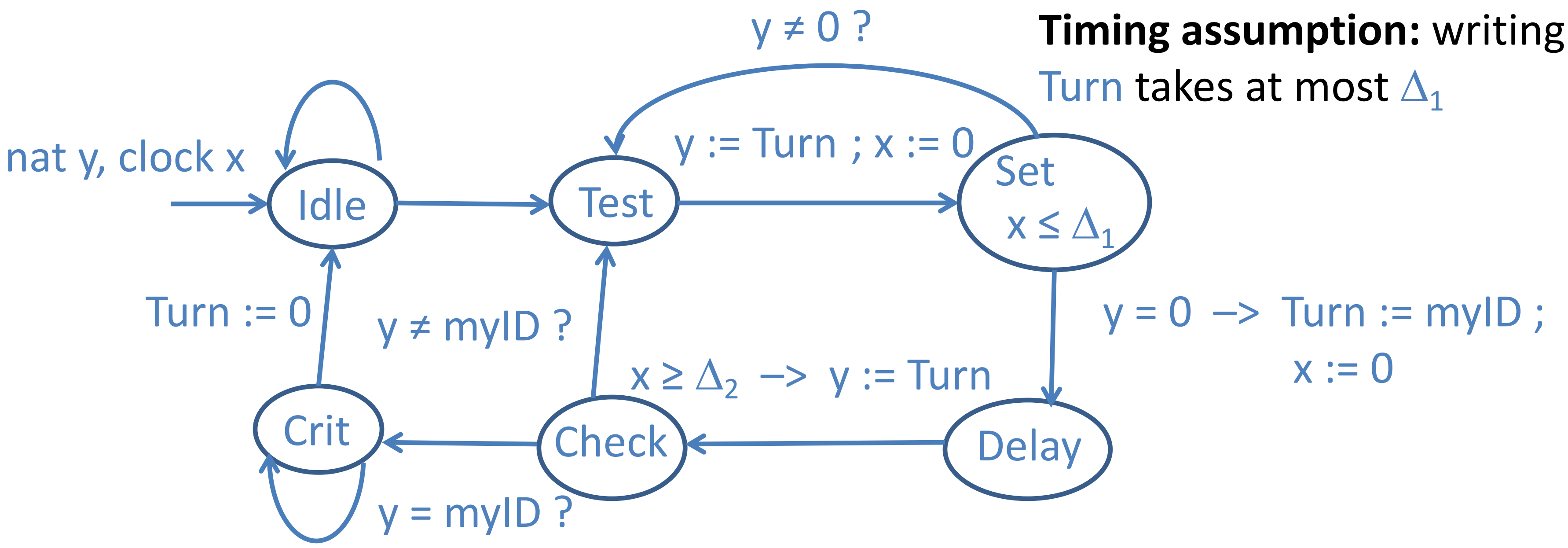
Solution: Delay and wait till you are sure that concurrent writes are finished

4. Read Turn again: if Turn equals your own ID then proceed to critical section; otherwise, go to Step 1 and try again
5. When done with critical section, set Turn back to 0

Fisher's Mutual Exclusion Protocol

AtomicReg nat Turn := 0

myID ∈ {1, 2, 3, …}



Timing assumption: writing Turn takes at most Δ_1

Wait for at least Δ_2 time units, and read Turn again

Does this work? Why?

Properties of Timed Fisher's Protocol

- ❑ If $\Delta_2 > \Delta_1$, the algorithm satisfies:
  1. Mutual exclusion (two processes cannot be in critical section simultaneously)
  2. 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

  In contrast, in the asynchronous model, mutual exclusion protocol for N processes is lot more complex than Peterson's algorithm

Exercise 1: Does the protocol satisfy the stronger property of *starvation freedom* (if a process wants to enter critical section then it eventually will)?

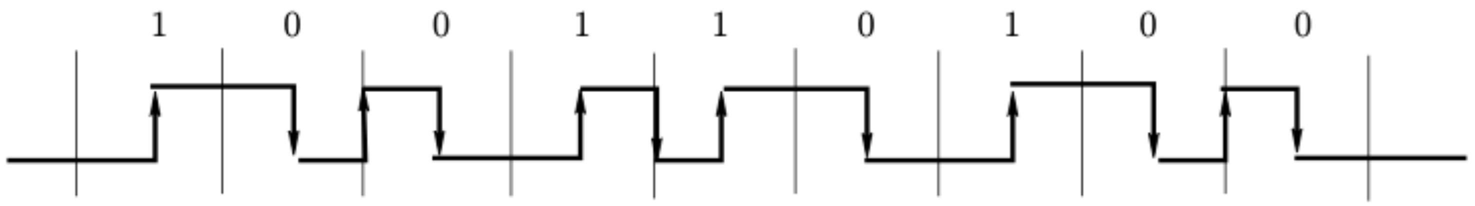
Exercise 2: If $\Delta_2 \leq \Delta_1$ does mutual exclusion hold? How about deadlock freedom?

Timed Communication

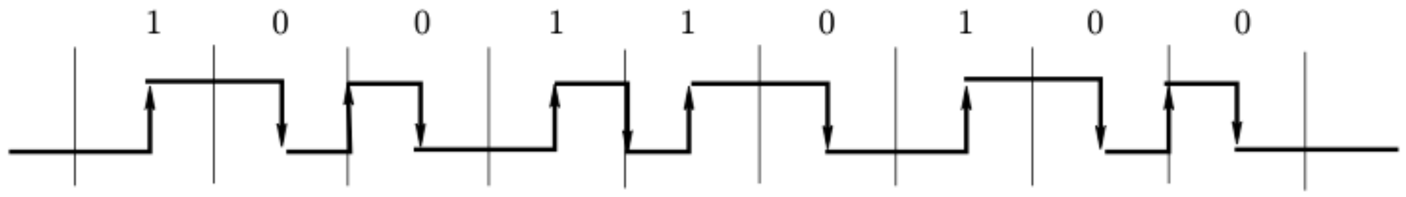
Suppose a sender wants to transmit a sequence of bits to a receiver connected by a communication bus

Natural strategy: Divide time into slots, and in each slot transmit a bit using low/high voltage values to encode 0/1

Manchester encoding: 0 encoded as a falling edge, and 1 encoded as a rising edge



Timed Communication Challenges



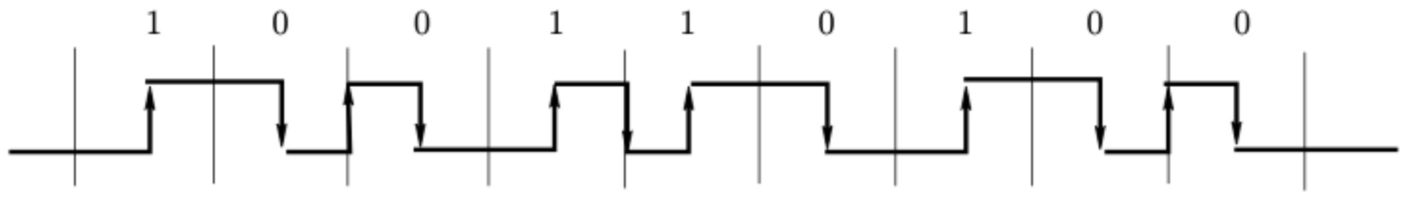
Sender and receiver know the duration of each time slot, but …

1. When idle, the voltage is set to low.
   So receiver doesn't know when the communication begins
2. Receiver cannot reliably detect falling edges
3. Sender and receiver clocks are synchronized imperfectly due to *drift* (when a clock x is 1, actual elapsed time is in interval $[1 - \varepsilon, 1 + \varepsilon]$)

Addressing the challenges:

1. All messages start with 1 and end with 00
2. Processes use timing information to transmit 0s
3. We use constraints like $x \leq 1 + \varepsilon$ instead of $x \leq 1$, and $1 - \varepsilon \leq x$ instead of $1 \leq x$

Audio Control Protocol



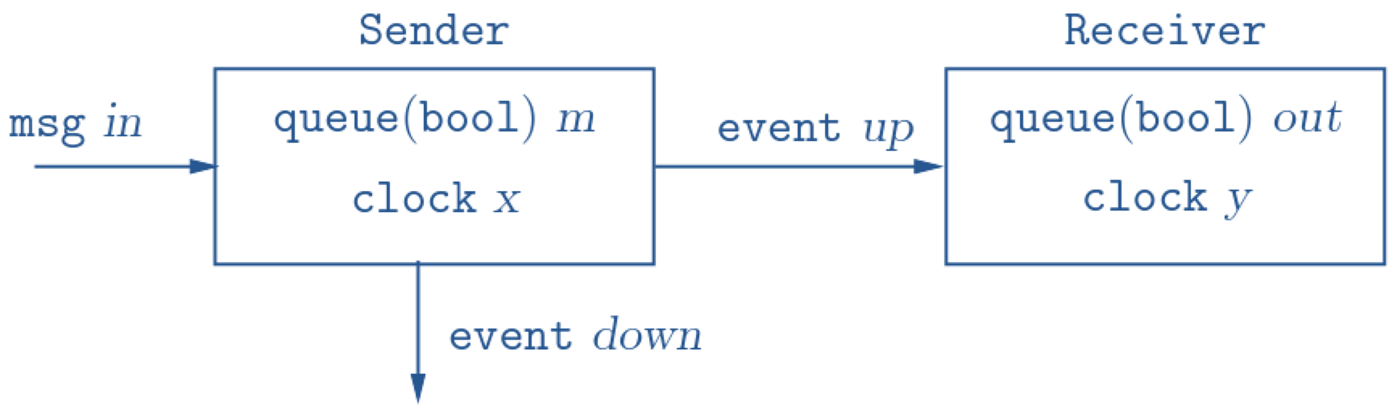
Protocol developed by Philips to reliably transmit messages in presence of imperfect clocks

Design logic for receiver to map measured delays between successive raising edges to sequence of bits

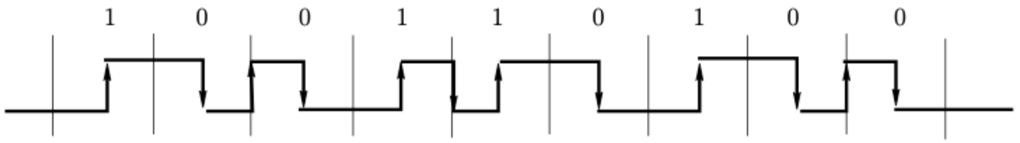
Verification: Prove that message transmission is reliable for a given drift rate ε

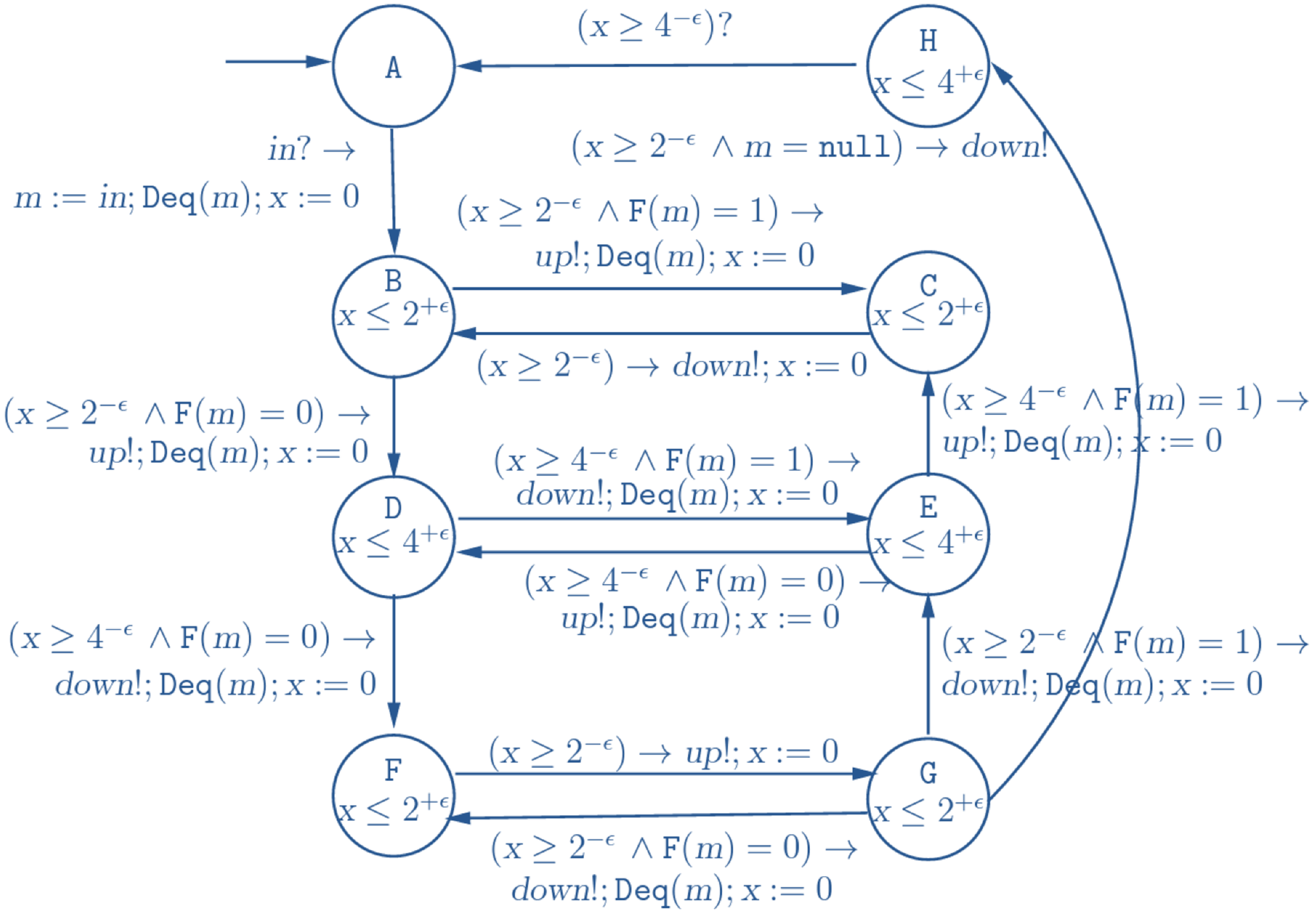
Optimization: Find the largest drift rate that the protocol tolerates

Audio Control System

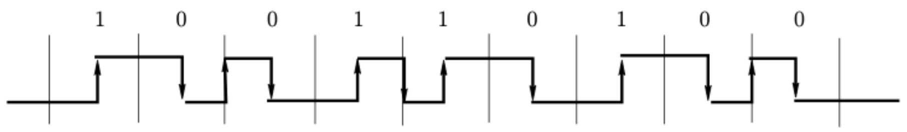


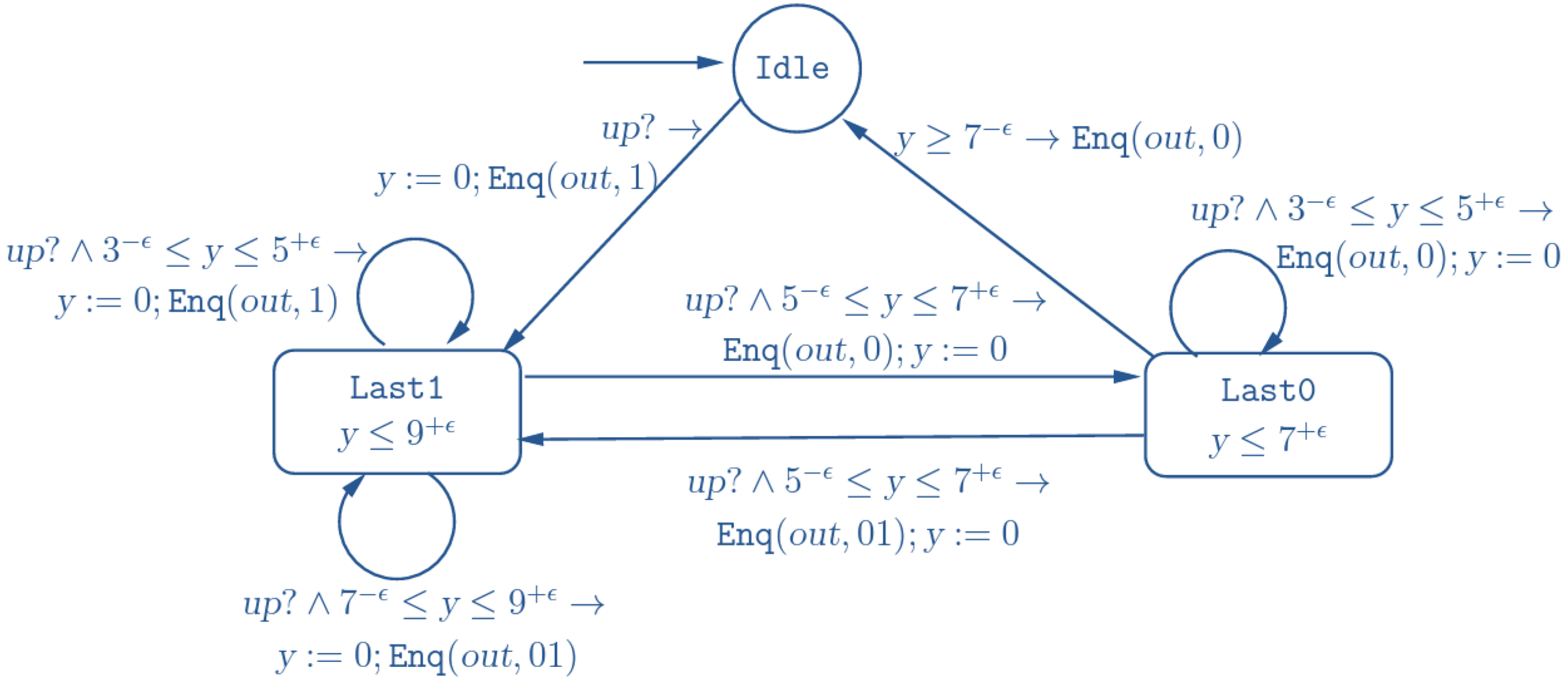
Sender Process



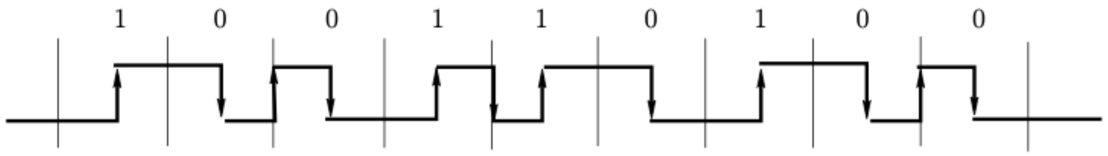

A
H
$x \leq 4^{+\epsilon}$
B
$x \leq 2^{+\epsilon}$
C
$x \leq 2^{+\epsilon}$
D
$x \leq 4^{+\epsilon}$
E
$x \leq 4^{+\epsilon}$
F
$x \leq 2^{+\epsilon}$
G
$x \leq 2^{+\epsilon}$
$(x \geq 4^{-\epsilon})?$
$in? \rightarrow$
$m := in; \mathtt{Deq}(m); x := 0$
$(x \geq 2^{-\epsilon} \wedge m = \mathtt{null}) \rightarrow down!$
$(x \geq 2^{-\epsilon} \wedge \mathtt{F}(m) = 1) \rightarrow$
$up!; \mathtt{Deq}(m); x := 0$
$(x \geq 2^{-\epsilon}) \rightarrow down!; x := 0$
$(x \geq 2^{-\epsilon} \wedge \mathtt{F}(m) = 0) \rightarrow$
$up!; \mathtt{Deq}(m); x := 0$
$(x \geq 4^{-\epsilon} \wedge \mathtt{F}(m) = 1) \rightarrow$
$up!; \mathtt{Deq}(m); x := 0$
$(x \geq 4^{-\epsilon} \wedge \mathtt{F}(m) = 1) \rightarrow$
$down!; \mathtt{Deq}(m); x := 0$
$(x \geq 4^{-\epsilon} \wedge \mathtt{F}(m) = 0) \rightarrow$
$up!; \mathtt{Deq}(m); x := 0$
$(x \geq 4^{-\epsilon} \wedge \mathtt{F}(m) = 0) \rightarrow$
$down!; \mathtt{Deq}(m); x := 0$
$(x \geq 2^{-\epsilon} \wedge \mathtt{F}(m) = 1) \rightarrow$
$down!; \mathtt{Deq}(m); x := 0$
$(x \geq 2^{-\epsilon}) \rightarrow up!; x := 0$
$(x \geq 2^{-\epsilon} \wedge \mathtt{F}(m) = 0) \rightarrow$
$down!; \mathtt{Deq}(m); x := 0$

Receiver Process



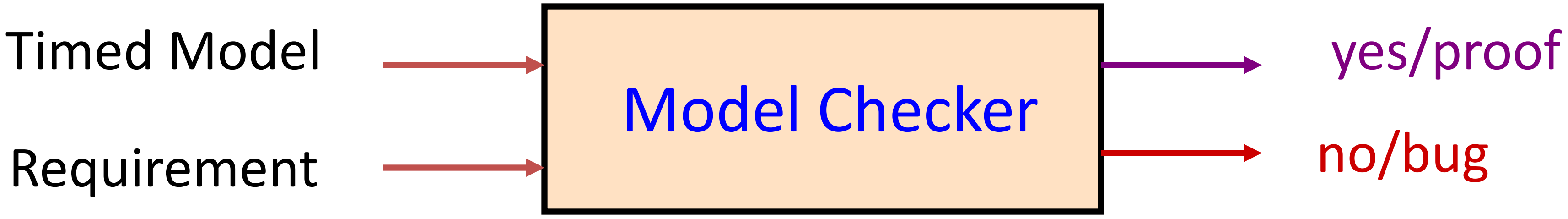

Idle
Last1
$y \leq 9^{+\epsilon}$
Last0
$y \leq 7^{+\epsilon}$
$up? \rightarrow$
$y := 0; \mathtt{Enq}(out, 1)$
$y \geq 7^{-\epsilon} \rightarrow \mathtt{Enq}(out, 0)$
$up? \wedge 3^{-\epsilon} \leq y \leq 5^{+\epsilon} \rightarrow$
$\mathtt{Enq}(out, 0); y := 0$
$up? \wedge 3^{-\epsilon} \leq y \leq 5^{+\epsilon} \rightarrow$
$y := 0; \mathtt{Enq}(out, 1)$
$up? \wedge 5^{-\epsilon} \leq y \leq 7^{+\epsilon} \rightarrow$
$\mathtt{Enq}(out, 0); y := 0$
$up? \wedge 5^{-\epsilon} \leq y \leq 7^{+\epsilon} \rightarrow$
$\mathtt{Enq}(out, 01); y := 0$
$up? \wedge 7^{-\epsilon} \leq y \leq 9^{+\epsilon} \rightarrow$
$y := 0; \mathtt{Enq}(out, 01)$

Execution Example



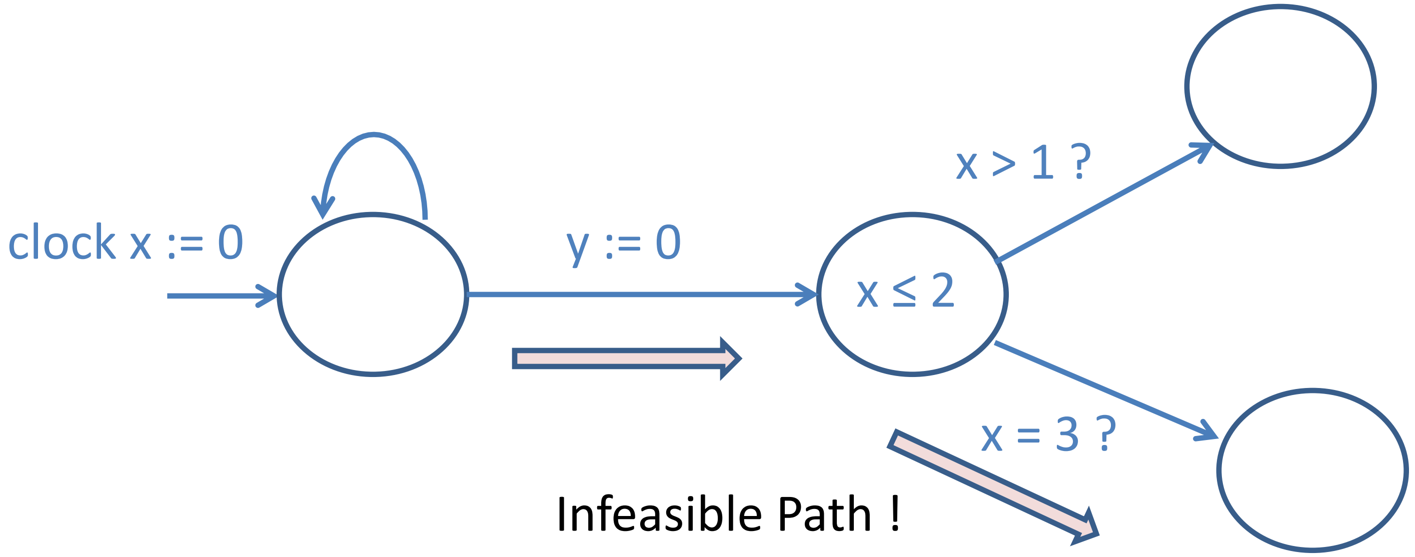
Time	Event	x	Sender	Queue m	y	Receiver	Queue *out*
0			B	00110100		Idle	null
2.07	*up*	2.07	D	0110100		Last1	1
5.97	*down*	3.9	F	110100	3.9	Last1	1
7.97	*up*	2	G	110100	5.9	Last0	10
9.92	*down*	1.95	E	10100	1.95	Last0	10
14.08	*up*	4.16	C	0100	6.11	Last1	1001
16.1	*down*	2.02	B	0100	2.02	Last1	1001
18	*up*	1.9	D	100	3.92	Last1	10011
22.05	*down*	4.05	E	00	4.05	Last1	10011
25.91	*up*	3.86	D	0	7.91	Last1	1001101
30.01	*down*	4.1	F	null	4.1	Last1	1001101
32.11	*up*	2.1	G	null	6.2	Last0	10011010
34.16	*down*	2.05	H	null	2.05	Last0	10011010
38.29		4.13	A	null	6.18	Last0	10011010
39.39		1.1	A	null	7.28	Idle	100110100

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 feasible?

Definition: A timed process TP is a *timed automaton* if for every clock variable x,

1. assignments to x in the description of TP are of the form $x := 0$

2. atomic expressions involving x (in clock-invariants or in guards) are of the form

   $$x \bowtie k$$

   where k is a constant and $\bowtie \in \{ =, \leq, >, <, \geq \}$

   (can express only constant lower/upper bounds on timing delays)

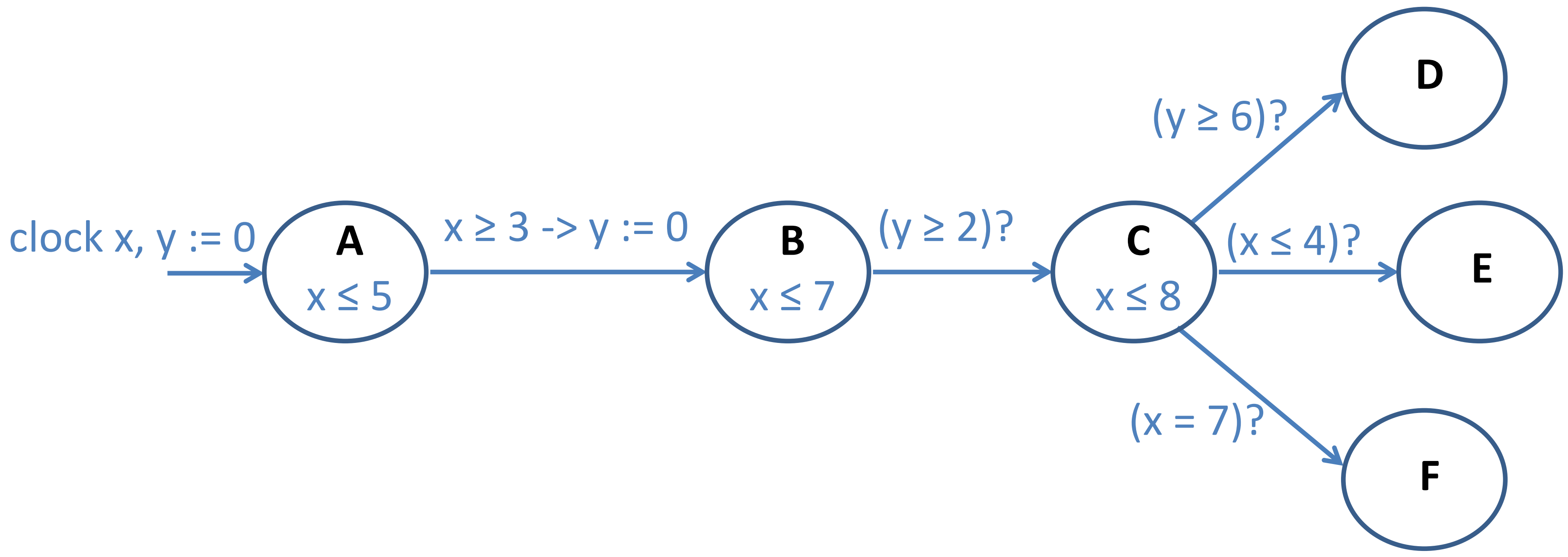
Timed Automata

Properties: Closed under parallel composition: If TP1 and TP2 are timed automata then TP1 | TP2 is also a timed automaton

A time automaton is *finite-state* if all its variables other than clocks have finite types (e.g. Boolean, enumerated)

Note: State space is still infinite due to the clock variables, but verification of safety properties is decidable

Timing Analysis Example



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

Requires propagation of the reachable combinations of x and y symbolically

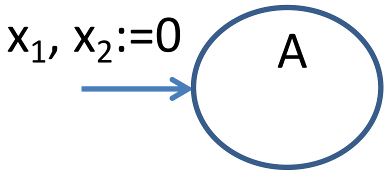
Timing Analysis Example

Clock-zone R_0

$0 \leq x_1 \leq 0$

$0 \leq x_2 \leq 0$

$0 \leq x_1 - x_2 \leq 0$



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

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

$0 \le x_1 \le 0$

$0 \le x_2 \le 0$

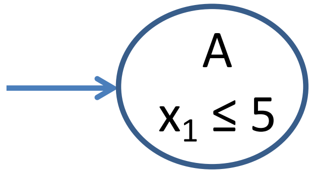
$0 \le x_1 - x_2 \le 0$

Clock-zone R'_0

$0 \le x_1 \le$ Infty

$0 \le x_2 \le$ Infty

$0 \le x_1 - x_2 \le 0$



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 ?

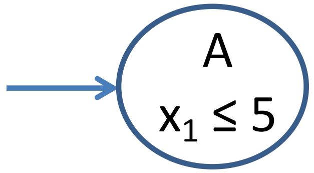
Step 1: Compute effect of timed transitions ignoring clock-invariants

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
$0 \le x_1 \le \text{Infty}$	$0 \le x_1 \le 5$	$0 \le x_1 \le 5$
$0 \le x_2 \le \text{Infty}$	$0 \le x_2 \le \text{Infty}$	$0 \le x_2 \le 5$
$0 \le x_1 - x_2 \le 0$	$0 \le x_1 - x_2 \le 0$	$0 \le x_1 - x_2 \le 0$



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

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

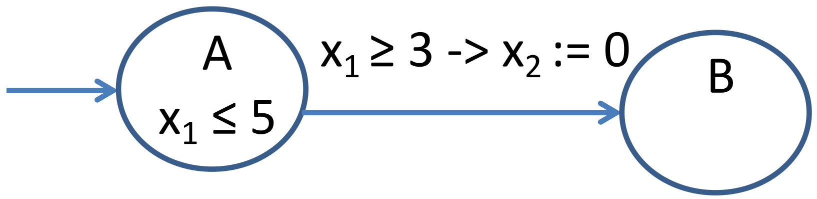
Canonicalization: Tighten all bounds to reflect implied constraints

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
$0 \leq x_1 \leq 5$	$3 \leq x_1 \leq 5$	$3 \leq x_1 \leq 5$	$3 \leq x_1 \leq 5$
$0 \leq x_2 \leq 5$	$0 \leq x_2 \leq 5$	$3 \leq x_2 \leq 5$	$0 \leq x_2 \leq 0$
$0 \leq x_1-x_2 \leq 0$	$0 \leq x_1-x_2 \leq 0$	$0 \leq x_1-x_2 \leq 0$	$3 \leq x_1-x_2 \leq 5$



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

Step 1: Intersect guard $3 \leq x_1$ with the clock-zone R_1

Step 2: Canonicalize by tightening constraints

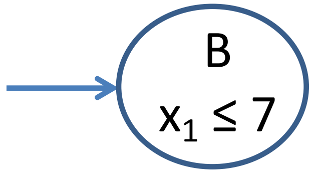
Step 3: Capture the effect of assignment $x_2 := 0$

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
$3 \leq x_1 \leq 5$	$3 \leq x_1 \leq \text{Infty}$	$3 \leq x_1 \leq 7$	$3 \leq x_1 \leq 7$
$0 \leq x_2 \leq 0$	$0 \leq x_2 \leq \text{Infty}$	$0 \leq x_2 \leq \text{Infty}$	$0 \leq x_2 \leq 4$
$3 \leq x_1-x_2 \leq 5$	$3 \leq x_1-x_2 \leq 5$	$3 \leq x_1-x_2 \leq 5$	$3 \leq x_1-x_2 \leq 5$



Starting from a state in R_2, 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 $x_1 \leq 7$

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?

DBM Representation of Constraints

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

$3 \leq x_1 - x_0 \leq 7$
$0 \leq x_2 - x_0 \leq \text{Infty}$
$3 \leq x_1 - x_2 \leq 5$

$x_1 - x_0 \leq 7$
$x_0 - x_1 \leq -3$
$x_2 - x_0 \leq \text{Infty}$
$x_0 - x_2 \leq 0$
$x_1 - x_2 \leq 5$
$x_2 - x_1 \leq -3$

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

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 $L \leq x_i \leq U$, we have $L \leq x_i - x_0 \leq U$
- Lower bound constraint $L \leq x_i - x_j$ can be rewritten as upper bound constraint $x_j - x_i \leq -L$
- DBM R is (m+1) x (m+1) matrix representing

  for $0 \leq i \leq m$, for $0 \leq j \leq m$, $x_i - x_j \leq R[i,j]$
- Diagonal entries should be 0: $x_i - x_i \leq 0$
- 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)

Timing Analysis Example

Clock-zone R_5

$5 \leq x_1 \leq 8$

$2 \leq x_2 \leq 5$

$3 \leq x_1 - x_2 \leq 5$

$$\begin{tikzcd}
 & & & & D \\
{} \arrow[r, "{x_1, x_2:=0}"] & \boxed{\begin{matrix} A \\ x_1 \leq 5 \end{matrix}} \arrow[r, "x_1 \geq 3 \text{ -> } x_2 := 0"] & \boxed{\begin{matrix} B \\ x_1 \leq 7 \end{matrix}} \arrow[r, "x_2 \geq 2"] & \boxed{\begin{matrix} C \\ x_1 \leq 8 \end{matrix}} \arrow[ur, "x_2 \geq 6"] \arrow[r, "x_1 \leq 4"] \arrow[dr, "x_1 = 7"'] & E \\
 & & & & F
\end{tikzcd}$$

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

Intersection of R_5 and $x_2 \geq 6$ unsatisfiable; means mode D not reachable

Intersection of R_5 and $x_1 \leq 4$ unsatisfiable; means mode E not reachable

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

Credits

Notes based on Chapter 7 of

Principles of Cyber-Physical Systems
by Rajeev Alur
MIT Press, 2015