CS:5980
Foundations of Embedded Systems

Timed Model

Copyright 2014-26, Rajeev Alur and Cesare Tinelli.
Created by Cesare Tinelli at the University of Iowa from notes originally developed by Rajeev Alur at the University of Pennsylvania. These notes are copyrighted materials and may not be used in other course settings outside of the University of Iowa in their current form or modified form without the express written permission of one of the copyright holders. During this course, students are prohibited from selling notes to or being paid for taking notes by any person or commercial firm without the express written permission of one of the copyright holders.

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 task precedence constraints

Asynchronous model

- Speeds at which different components execute are independent
- Computation within a step: execute a single task that is enabled

Timed model (partially synchronous)

- Like asynchronous for communication of information
- But can rely on global time for coordination

Continuous-time model (for dynamical system)

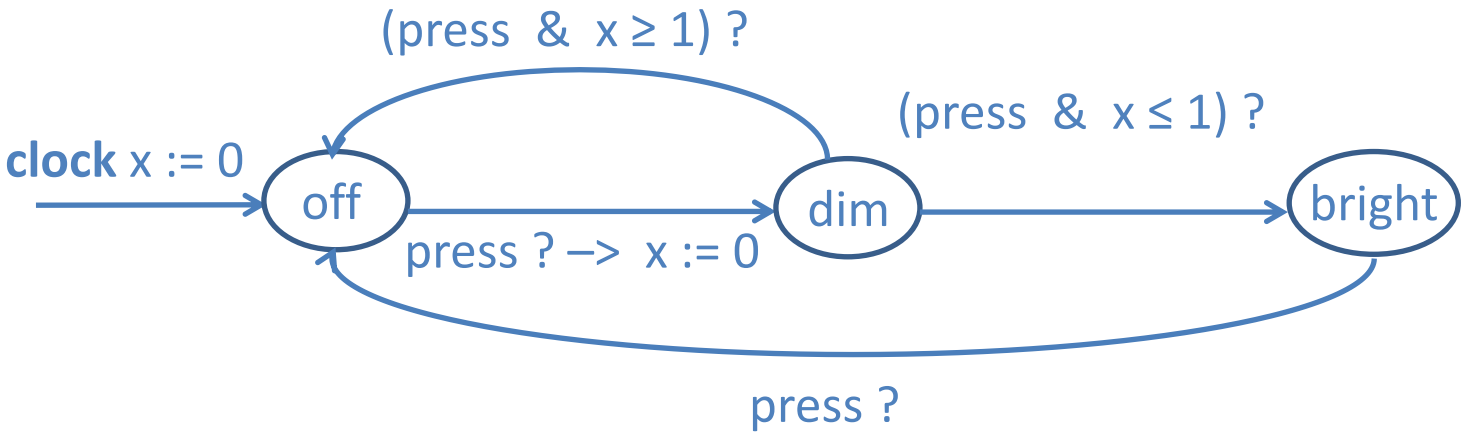
- Synchronous, but now time evolves continuously
- Execution of system: solution to differential equations

Timed Model

The timed model allows us to express phenomena such as:

- Task A is is executed every 5ms
- The delay between the reception of an input value and the corresponding output response is between 2ms to 4ms
- If an acknowledgment is not received within 5ms, the message is resent

Timed Model Example



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

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

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

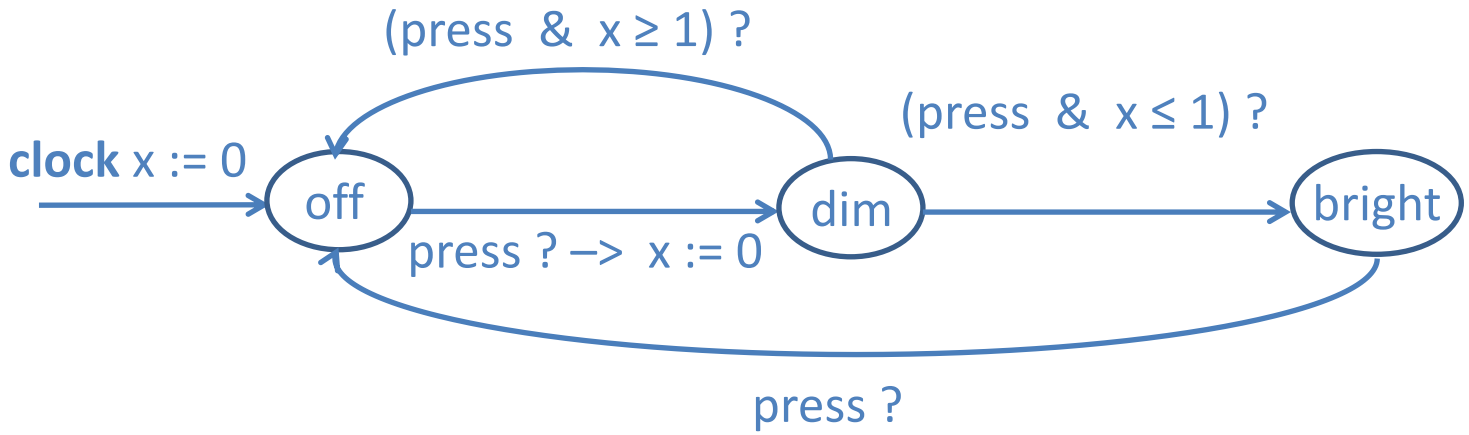
Timed transition: (dim, 0) –0.8–> (dim, 0.8)

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)

Timed Model Example



Clock variables:

- Tests and updates in mode-switches like other variables
- New: During a timed transition of duration d, the value of clock variables increases by an amount equal to d

Timing constraints:

Update x := 0 in off –> dim and guard x ≤ 1 in dim –> bright indicates that timing of these two transitions is ≤ 1 apart

Timed Buffer Example

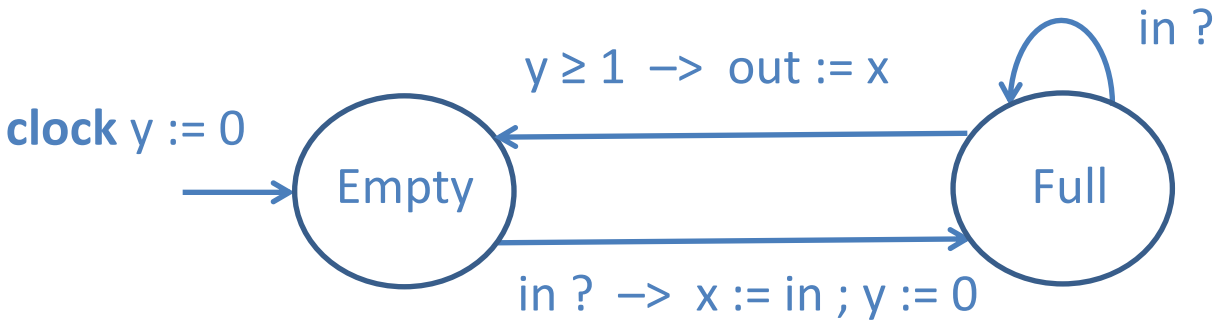


Buffer with a bounded delay

Behavior:

Input received on channel in is transmitted on output channel out after a delay of d time units, with LB ≤ d ≤ UB
(i.e., the delay has known lower and upper bounds)

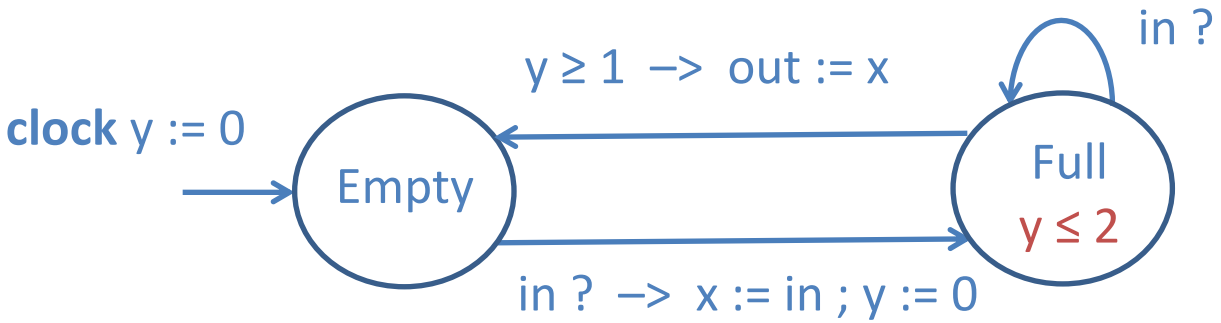
Modeling Timed Buffer



- **Input channel:** **msg** in **Output channel:** **msg** out
- Mode indicates 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 filled up
- When buffer is 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 clock y exceeds the upper bound 2?

Clock Invariants



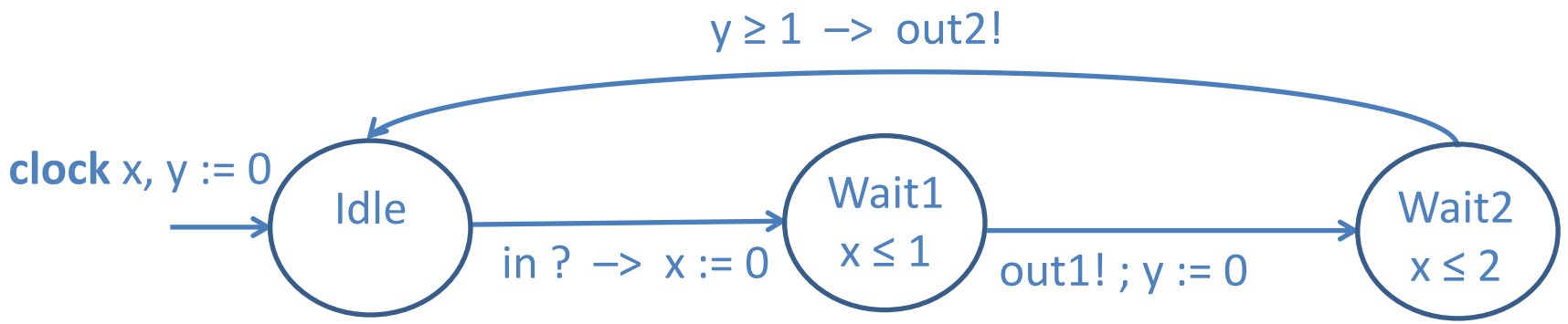
The constraint y ≤ 2 associated with mode Full is a *clock invariant*

A timed transition of duration d is allowed only if the clock invariant is satisfied for the entire duration of the transition

- (Full, x, 0.8) –0.7–> (Full, x, 1.5) allowed
- (Full, x, 0.8) –1.4–> (Full, x, 2.2) disallowed

Clock invariants limit how long a process stays in a mode

Example with Two Clocks

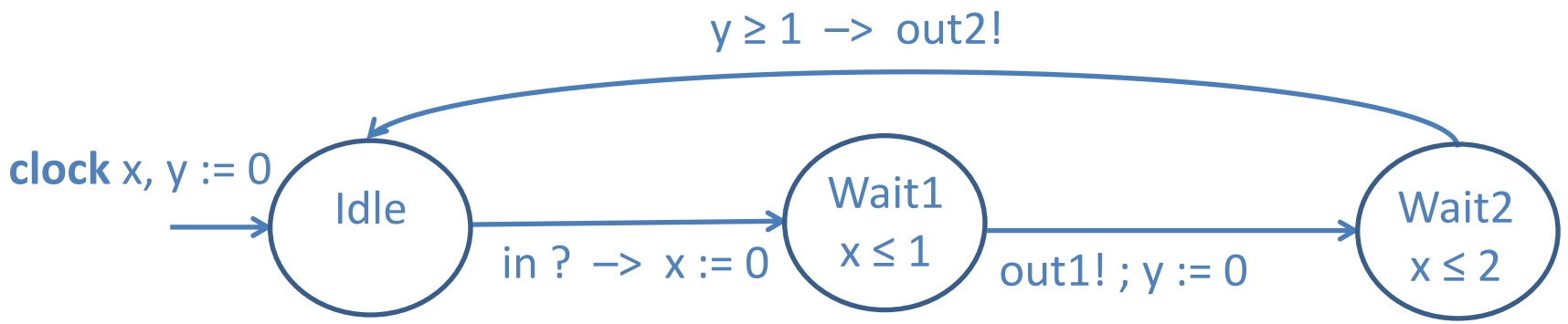


- **Input channel:** **event** in **Output channels:** **event** out1, out2
- Two clock variables: x, y
- As time passes, both clocks increase (and at the same rate)

Sample timed transitions from state (mode, x, y) = (Wait2, 0.8, 0) :

(Wait2, 0.8, 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 last input event
- Clock y tracks time elapsed since the output event

What is the behavior of this model?

Suppose an input event occurs at time t,

- the process produces an output event on channel out1 at time $t' \in [t, t+1]$
- then on channel out2 at time $t'' \in [t'+1, t+2]$

Example Specification

Consider a timed process with

Input: **event** x **Output:** **event** y, event z

Desired behavior:

- For each input event, produce both output events
- Time delay between x? and y! is in interval [2, 4]
- Time delay between x? and z! is in interval [3, 5]
- Ignore later inputs received in these intervals

Definition of Timed Process

Definition: A *timed process* TP consists of

1. An asynchronous process P, where some of the state variables can be of type *clock* (non-negative reals)
2. A *clock invariant* CI, a Boolean expression over P's state variables

Note: Inputs, outputs, states, initial states, internal actions, input actions, and output actions are exactly as in the asynchronous model

Definition of Timed Process

Notation: For a state s and time t, let $s+t$ denote the state such that

- $(s+t)(x) = s(x) + t$ for every clock variable x
- $(s+t)(y) = s(y)$ for every non-clock variable y

Definition: Given a state s and a time $d > 0$, *timed action* $s \text{ –d–> } s+d$ is a transition of duration d where state $s+t$ satisfies invariant CI for all $t \in [0, d]$

Note: If CI is a convex constraint, it suffices that s and $s+d$ satisfy CI

Definition of Parallel Composition

Let $TP_1 = (P_1, CI_1)$ and $TP_2 = (P_2, CI_2)$ be timed processes

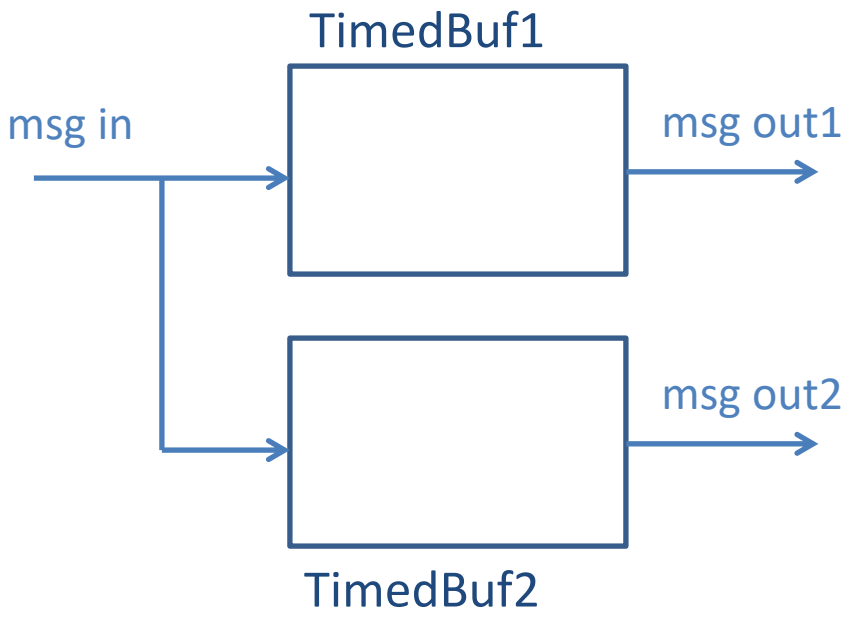
The parallel composition $TP_1 \mid TP_2$ is defined iff $P_1 \mid P_2$ is defined
(that is, iff the outputs of P_1 and P_2 are disjoint)

Composition: $TP_1 \mid TP_2 = (P_1 \mid P_2, CI_1 \wedge CI_2)$

- Asynchronous composition of P_1 and P_2 defines the internal, input and output actions of $TP_1 \mid TP_2$
- Conjunction of CI_1 and CI_2 defines the clock-invariant of $TP_1 \mid TP_2$

Consequence: The composite process allows a timed action of duration d exactly when both TP_1 and TP_2 can wait for time d

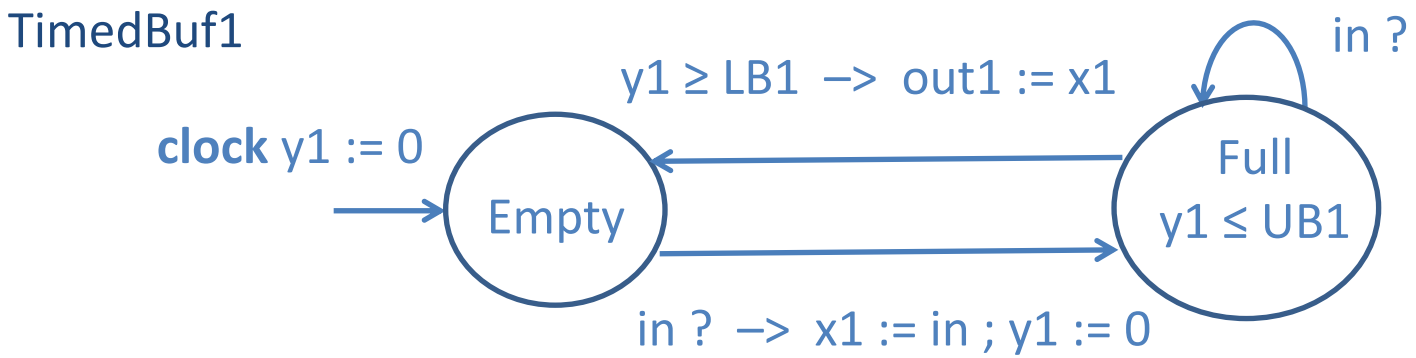
Composition of Processes

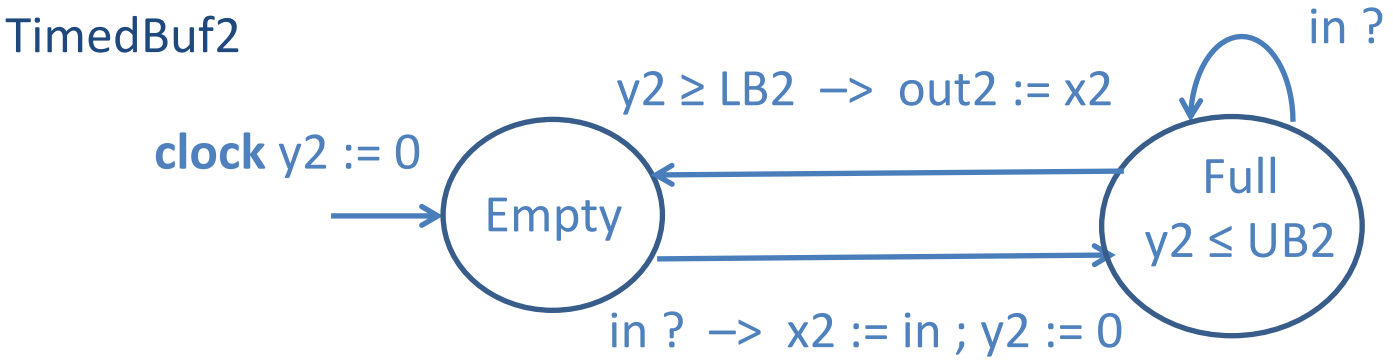


How to construct timed process corresponding to the composition of the two processes?

What are the possible behaviors of the composite process?

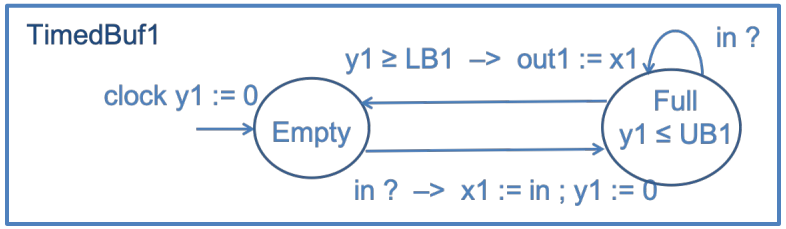
Composition of Timed Processes

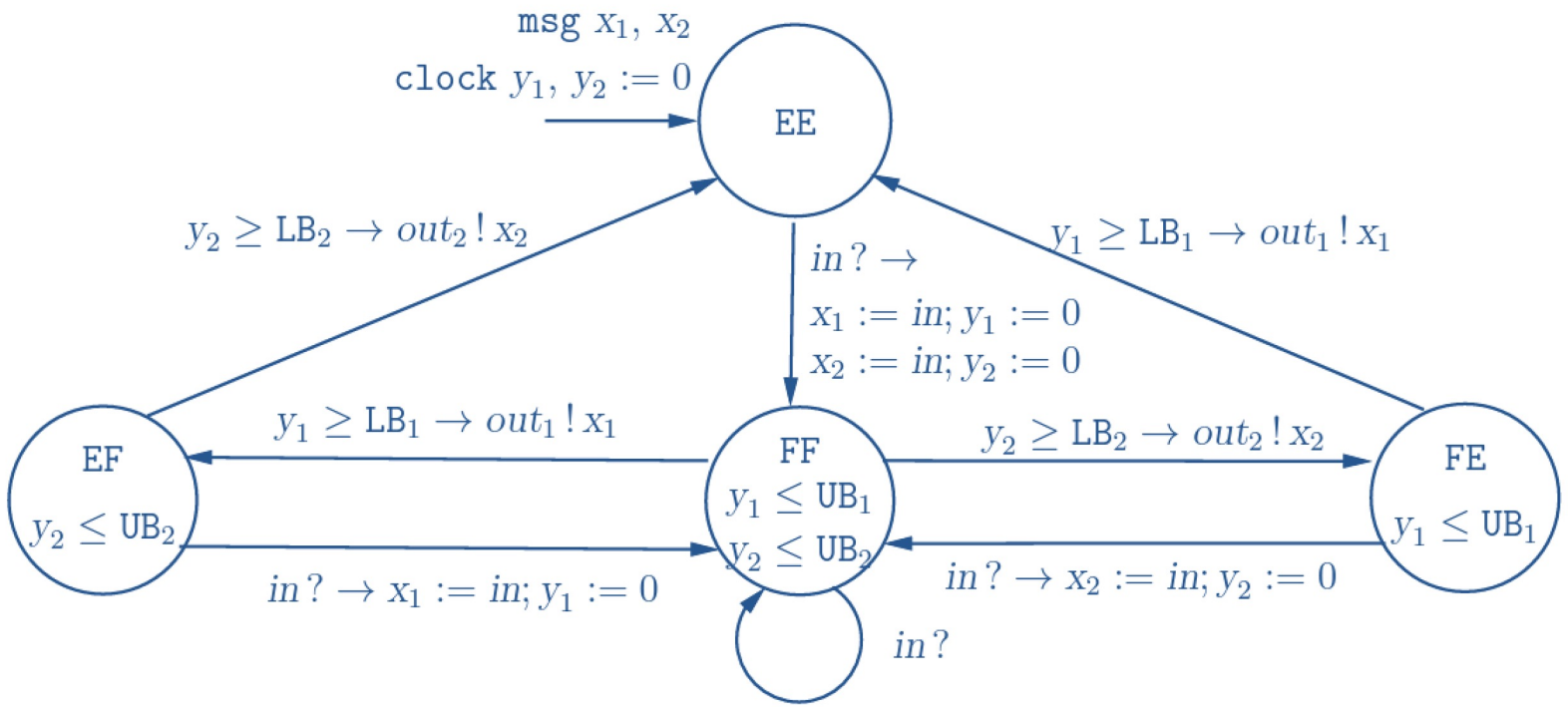




The composite process has four modes: (Empty, Empty), (Empty, Full), (Full, Empty), (Full, Full)

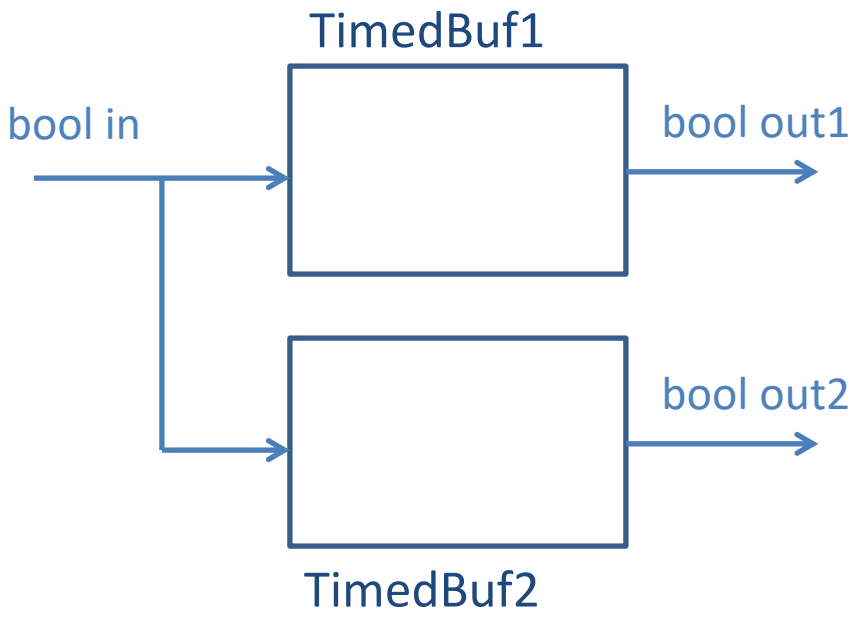
Composition of Timed Processes





$$CI = (\text{mode} = EF \Rightarrow y_2 \leq UB_2) \wedge (\text{mode} = FF \Rightarrow y_1 \leq UB_1 \wedge y_2 \leq UB_2) \wedge (\text{mode} = FE \Rightarrow y_1 \leq UB_1)$$

Composition of Processes



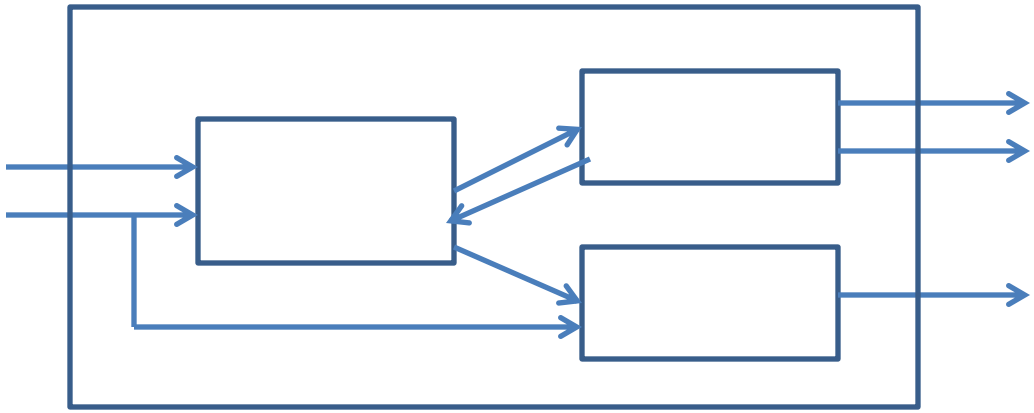
If UB1 < LB2 then out1 is guaranteed to occur before out2

- Implicit coordination based on bounds on delays

Is it possible to observe two out1 events without an intervening out2 event?

- Depends on relative magnitudes of bounds (need timing analysis!)

Block Diagrams



Components can be timed processes now

Operations: instantiation (I/O variable renaming), parallel composition, and variable hiding

Composite system step:

1. an internal step of one of the components,
2. a communication (I/O) step involving relevant sender and receivers, or
3. a timed step involving all the components

Timed Model

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

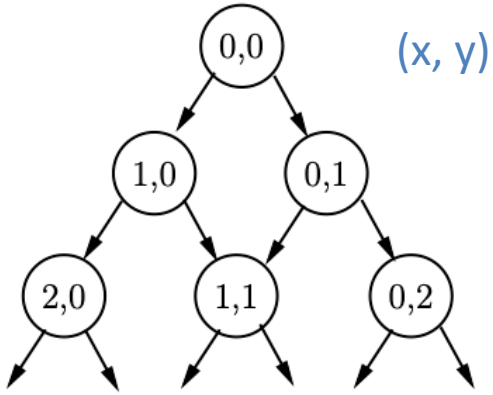
Definitions/concepts that carry over naturally from those models:

- 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



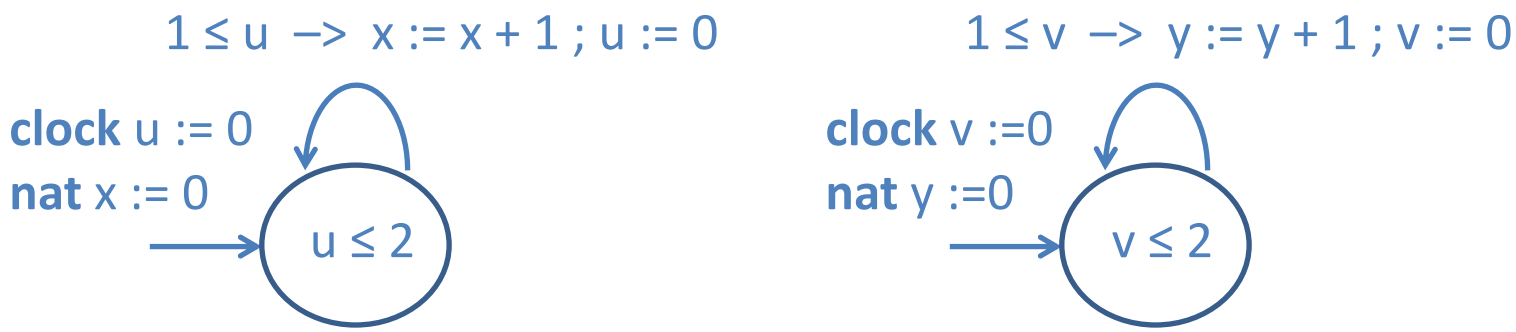
Tasks A_x and A_y execute in an arbitrary order

For every possible choice of numbers m and n, the state (m, n) is reachable

Fairness assumptions are used to rule out executions where a task 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

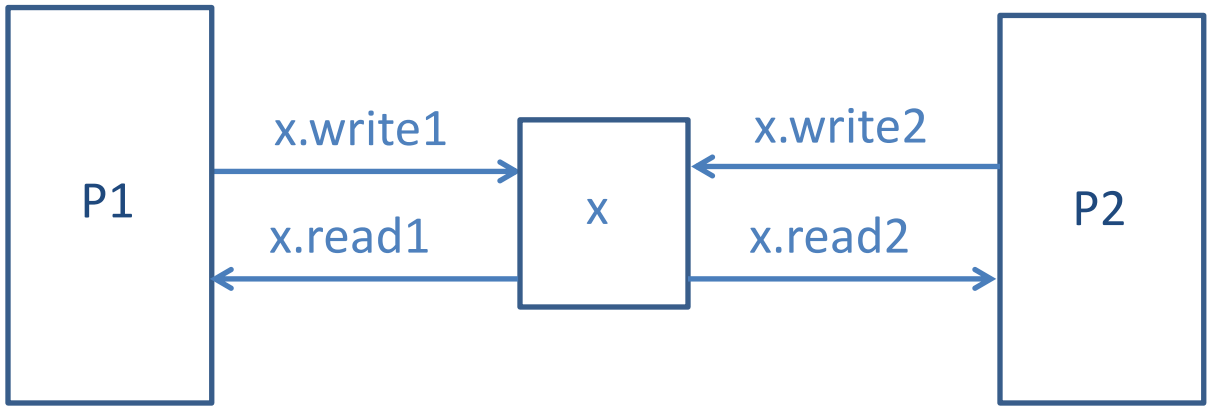
Task A_y increments y, and this also takes between 1 to 2 time units

Tasks execute in parallel, asynchronously, but timing introduces loose coordination (since all clocks advance in unison)

Which states are reachable?

What relationship between m and n guarantees that state (m, n) is reachable?

Recall: Shared Memory Asynchronous Processes



Processes P1 and P2 communicate by reading/writing shared variables

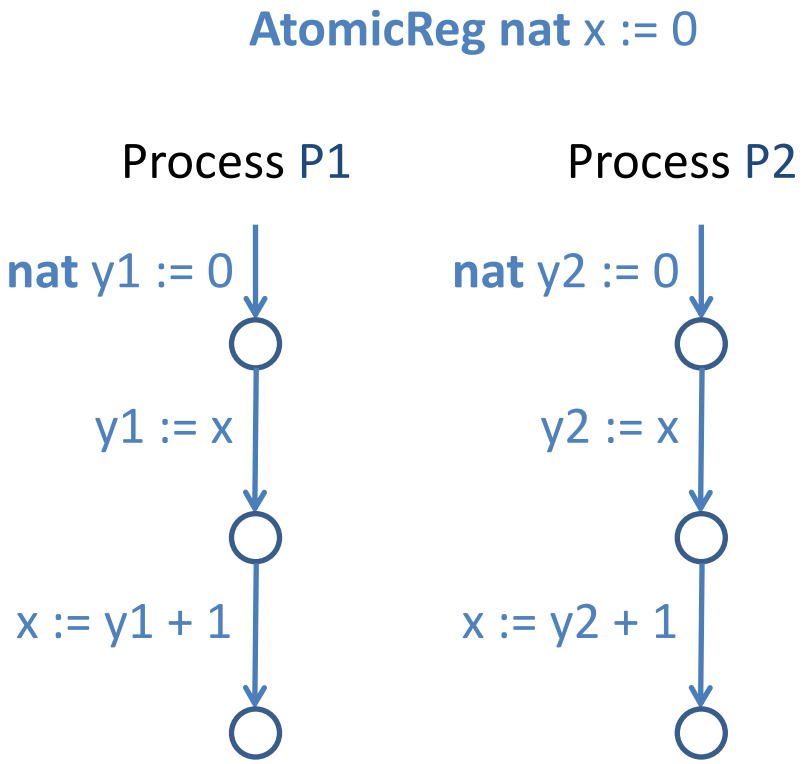
Each shared variable can be modeled as an asynchronous process

- 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



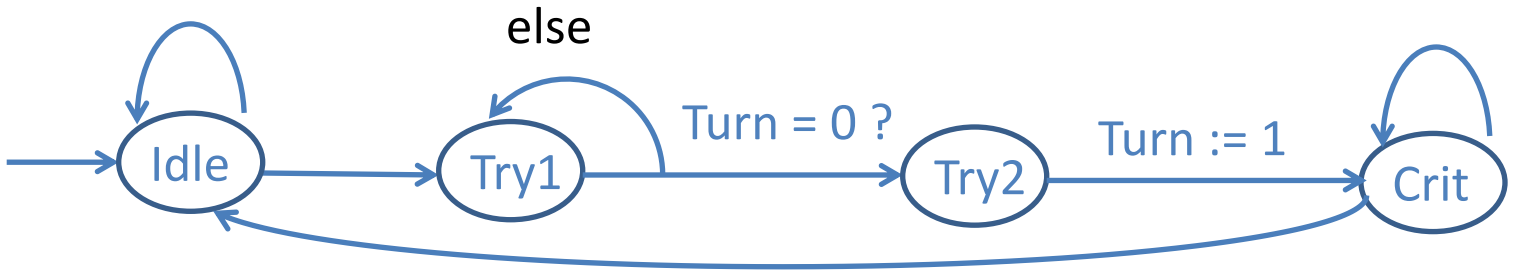
Safety: processes should not both be in critical section simultaneously
(can be formalized using invariants)

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

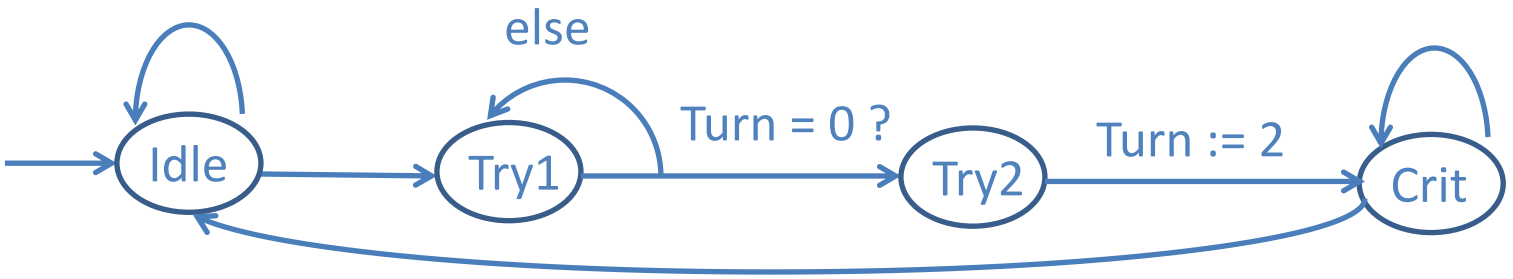
Process P1



Turn := 0

Process P2

What was the problem?



Turn := 0

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)

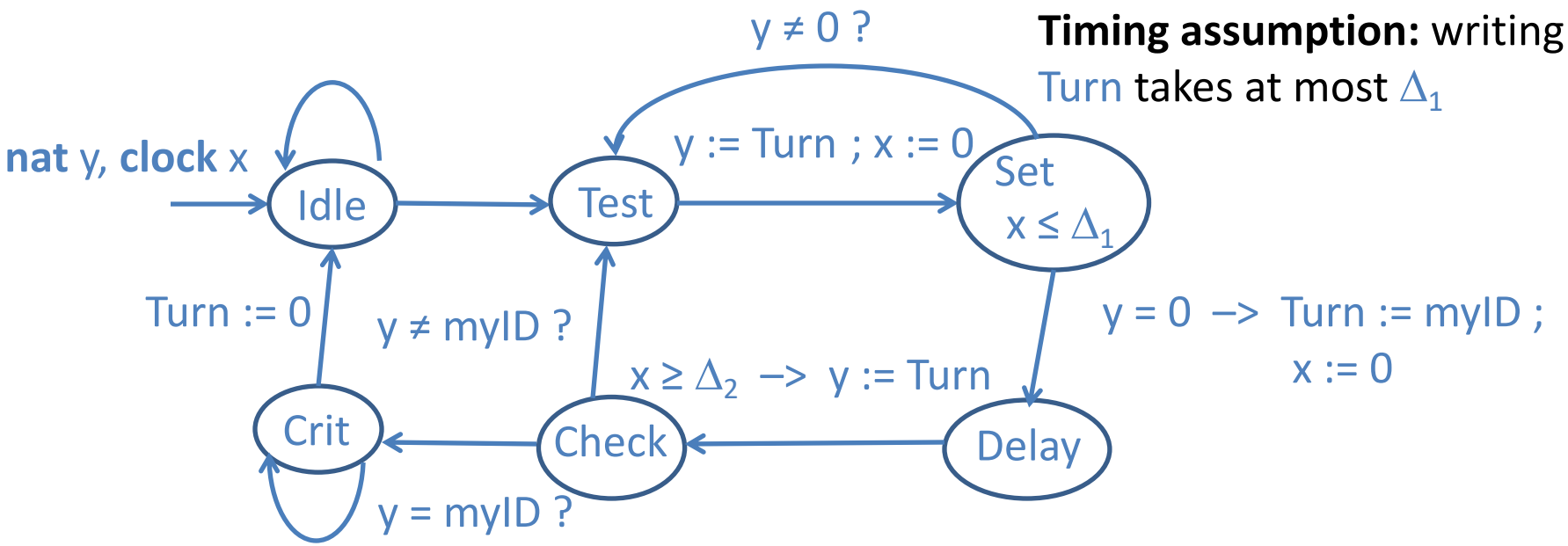
Solution: Delay and wait till you are sure that concurrent writes are finished

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 or why not?

Properties of Timed Fisher's Protocol

1. If $\Delta_2 > \Delta_1$, the algorithm satisfies:
   a. Mutual exclusion (two processes cannot be in critical section simultaneously)
   b. Deadlock freedom (if a process wants to enter critical section, then some process will enter critical section)
2. 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)?

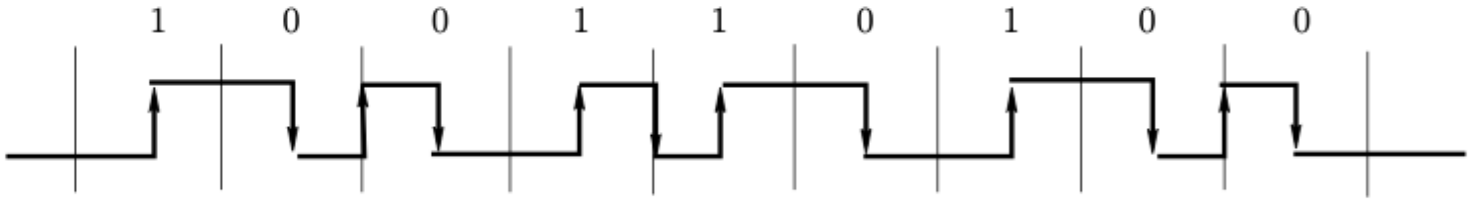
Exercise 2: If $\Delta_2 \leq \Delta_1$ does mutual exclusion hold? How about deadlock freedom?

Timed Communication

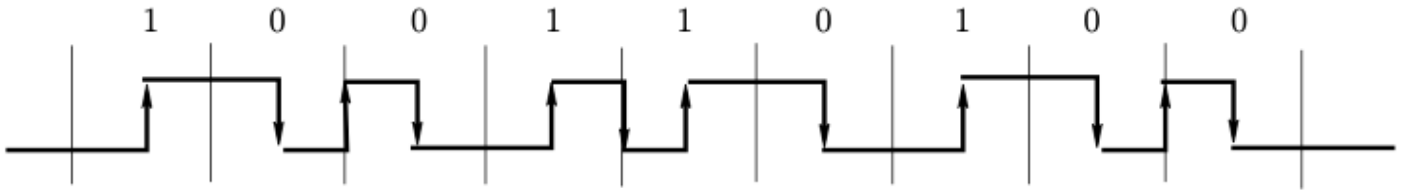
Suppose a sender wants to transmit a sequence of bits to a receiver connected by a communication bus

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



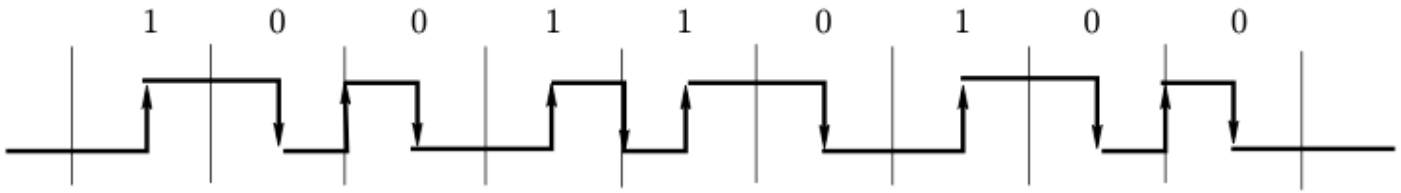
Sender and receiver know the duration of each time slot, but …

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



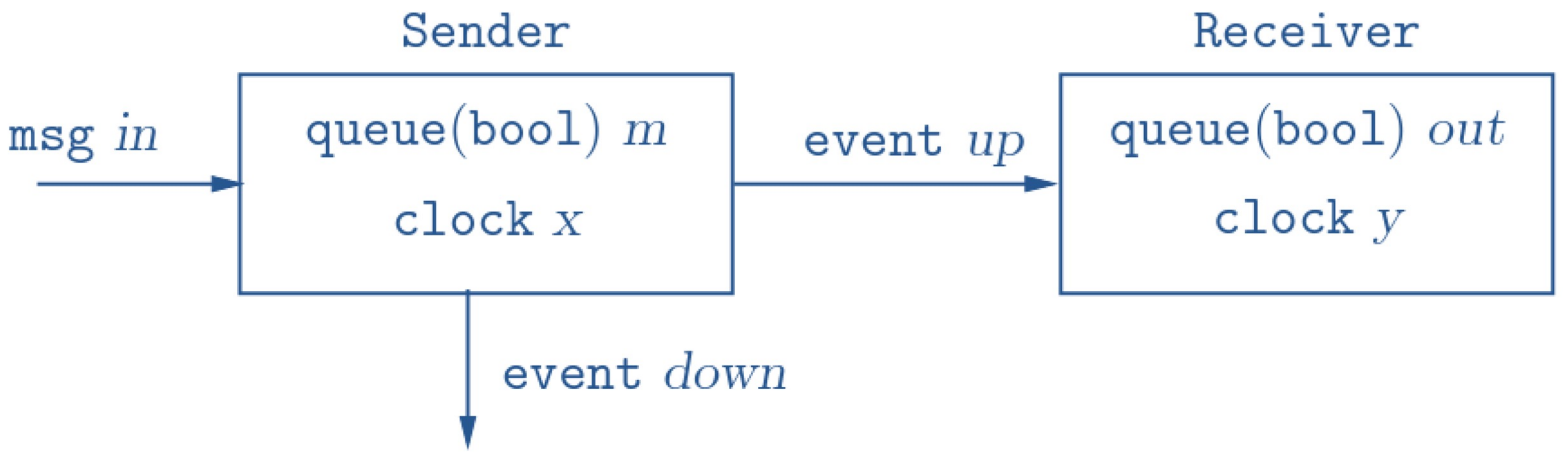
Protocol developed by Philips to reliably transmit messages in presence of imperfect clocks

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

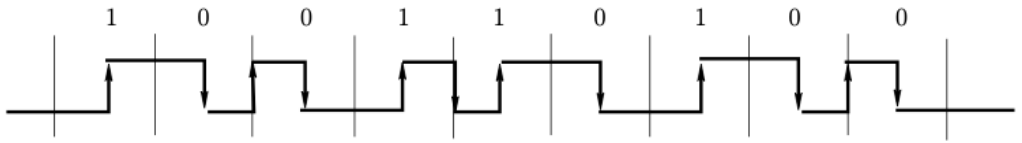
Verification: Prove that message transmission is reliable for a given drift rate ε

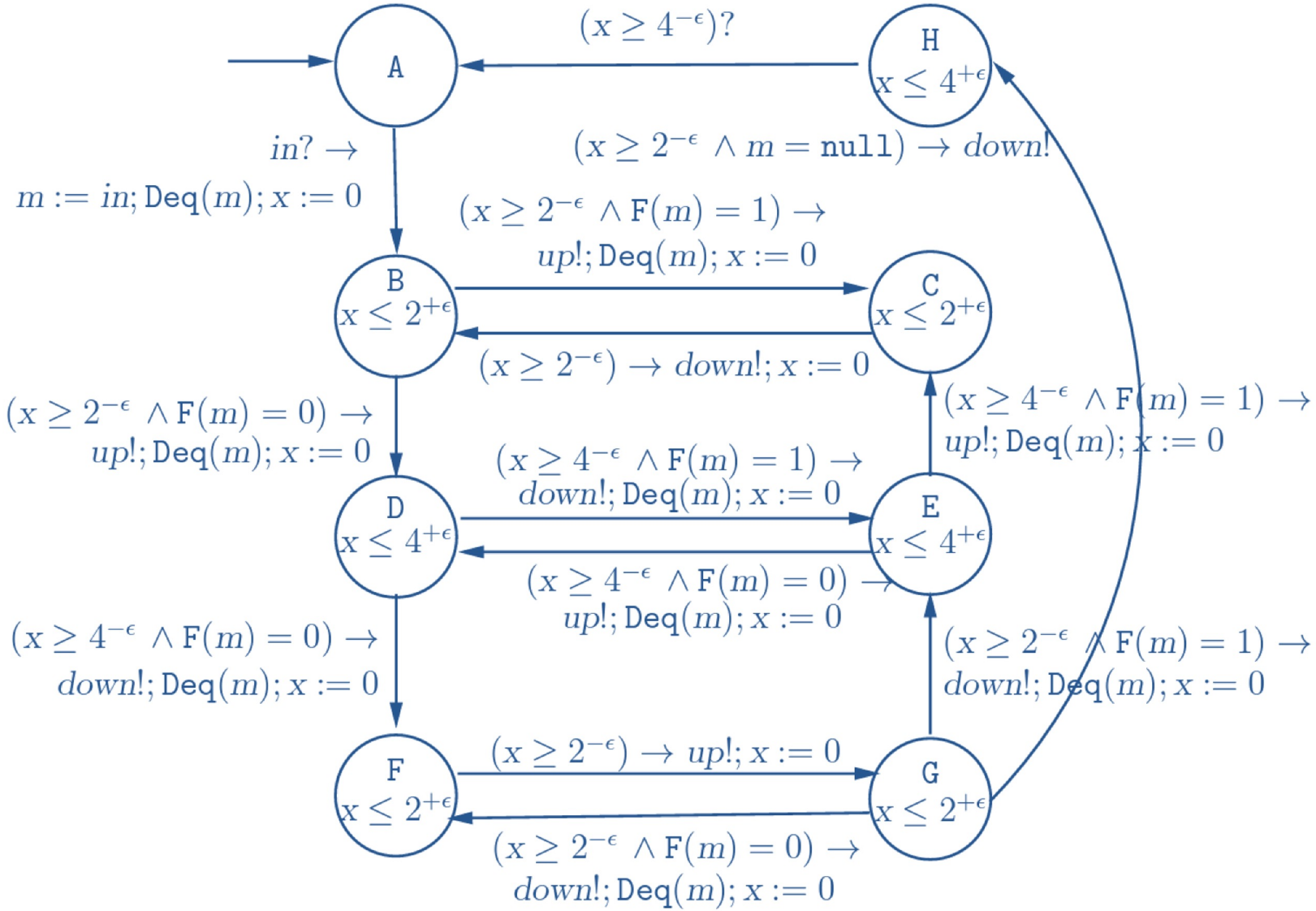
Optimization: Find the largest drift rate that the protocol tolerates

Audio Control System

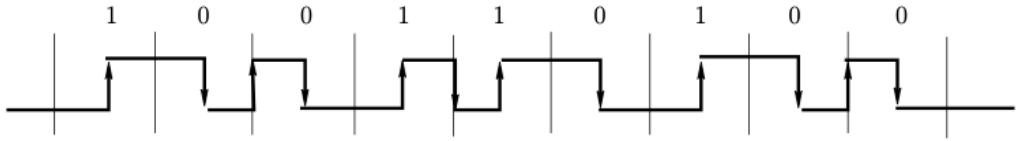


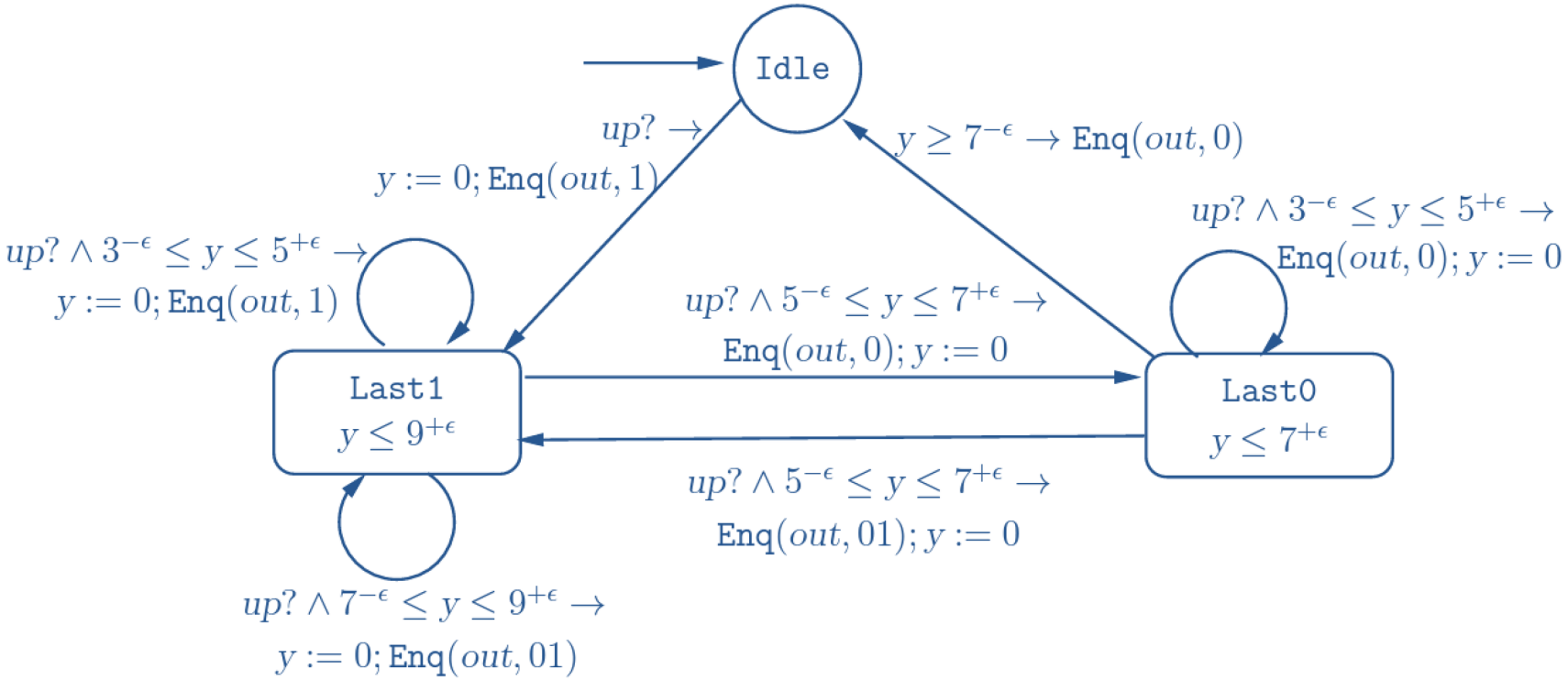
Sender Process



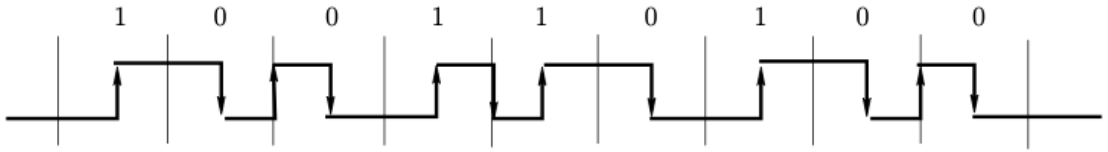


Receiver Process



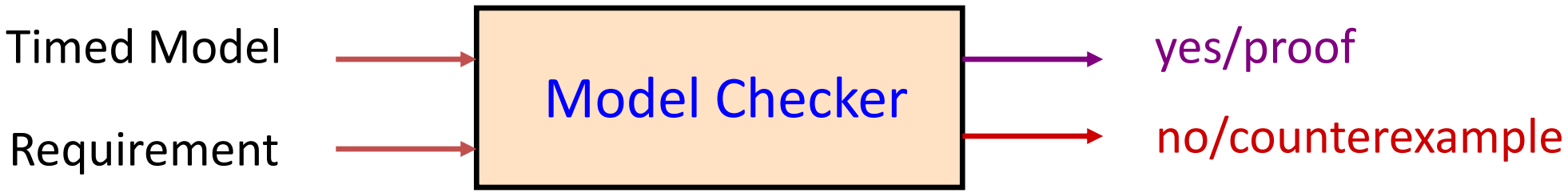


Execution Example



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



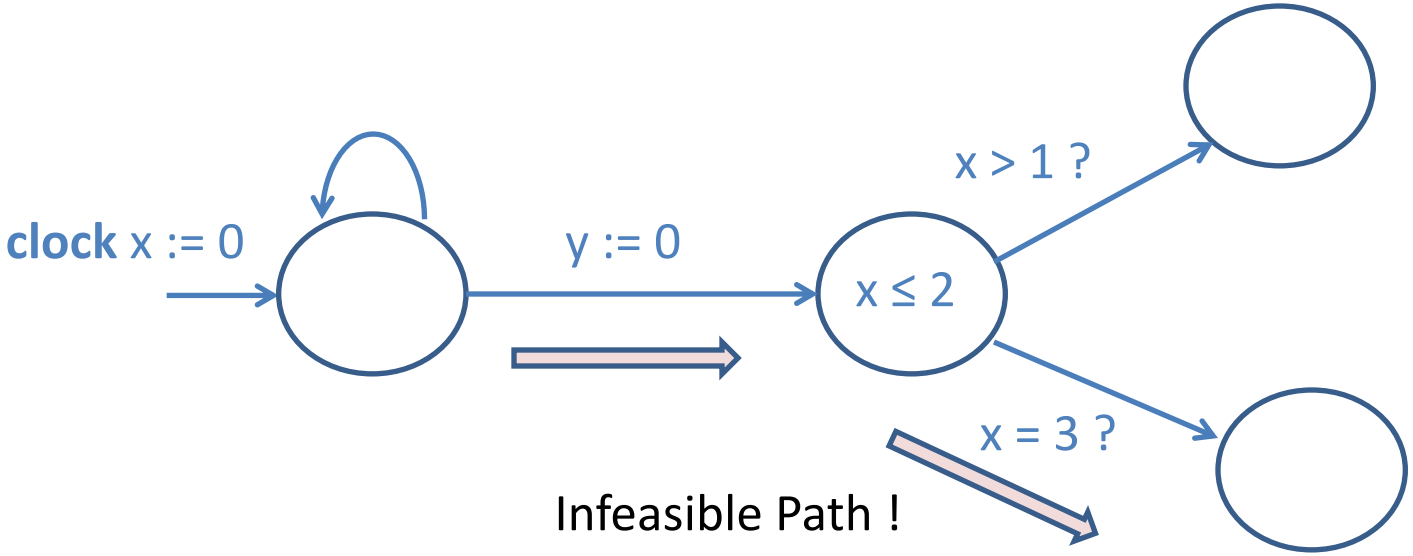
How to adapt algorithms for searching through the state-space of a model in presence of clock variables and timing constraints?

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)

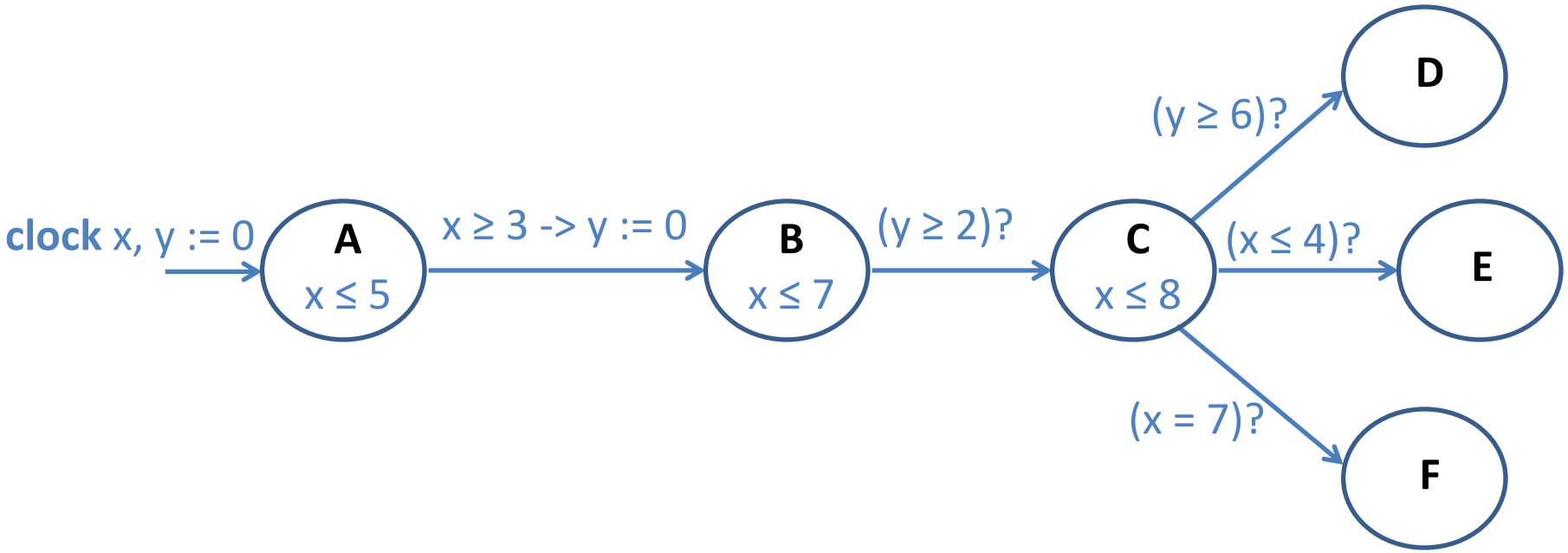
Timed Automata

Theorem (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



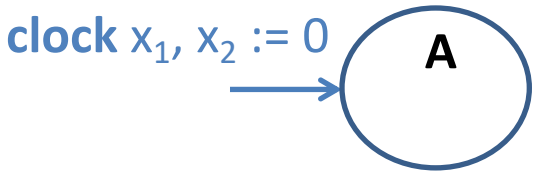
Which of the modes **D**, **E**, **F** are reachable ?

Requires propagation of the reachable combinations of x and y symbolically

Timing Analysis Example

Clock-zone R_0

$0 \leq x_1 \leq 0$



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

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

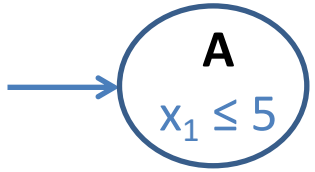
Timing Analysis Example

Clock-zone R_0

$0 \leq x_1 \leq 0$

Clock-zone R'_0

$0 \leq x_1 \leq \infty$



Starting from a state in $\mathbf{R_0}$, 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 ?

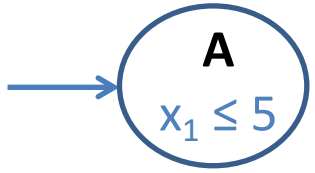
Step 1: Compute effect of timed transitions ignoring clock-invariants
Constraints on individual clocks: Change upper bound to ∞
Constraints on differences between clock values: unchanged (why?)

Timing Analysis Example

Clock-zone R'_0
$0 \leq x_1 \leq \infty$

Clock-zone R''_0
$0 \leq x_1 \leq 5$

Clock-zone R_1
$0 \leq x_1 \leq 5$



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

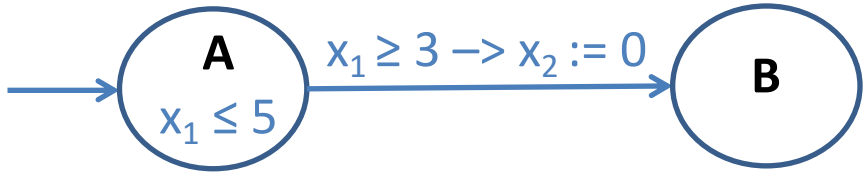
Timing Analysis Example

Clock-zone R_1

$0 \leq x_1 \leq 5$ $\quad$ $3 \leq x_1 \leq 5$ $\quad$ $3 \leq x_1 \leq 5$

Clock-zone R_2

$3 \leq x_1 \leq 5$



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 $\mathbf{R_1}$

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

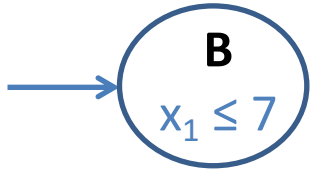
Timing Analysis Example

Clock-zone R_2
$3 \leq x_1 \leq 5$

$3 \leq x_1 \leq \infty$

$3 \leq x_1 \leq 7$

Clock-zone R_3
$3 \leq x_1 \leq 7$



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 ∞

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

$3 \leq x_1 \leq 7$

$3 \leq x_1 - x_0 \leq 7$

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

$x_1 - x_0 \leq 7$

$x_0 - x_1 \leq -3$

$x_2 - x_0 \leq \infty$

$x_0 - x_2 \leq 0$

$x_1 - x_2 \leq 5$

$x_2 - x_1 \leq -3$

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 $x_1, x_2, \ldots, x_m$
- 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 $(m+1) \times (m+1)$ matrix representing the constraints:

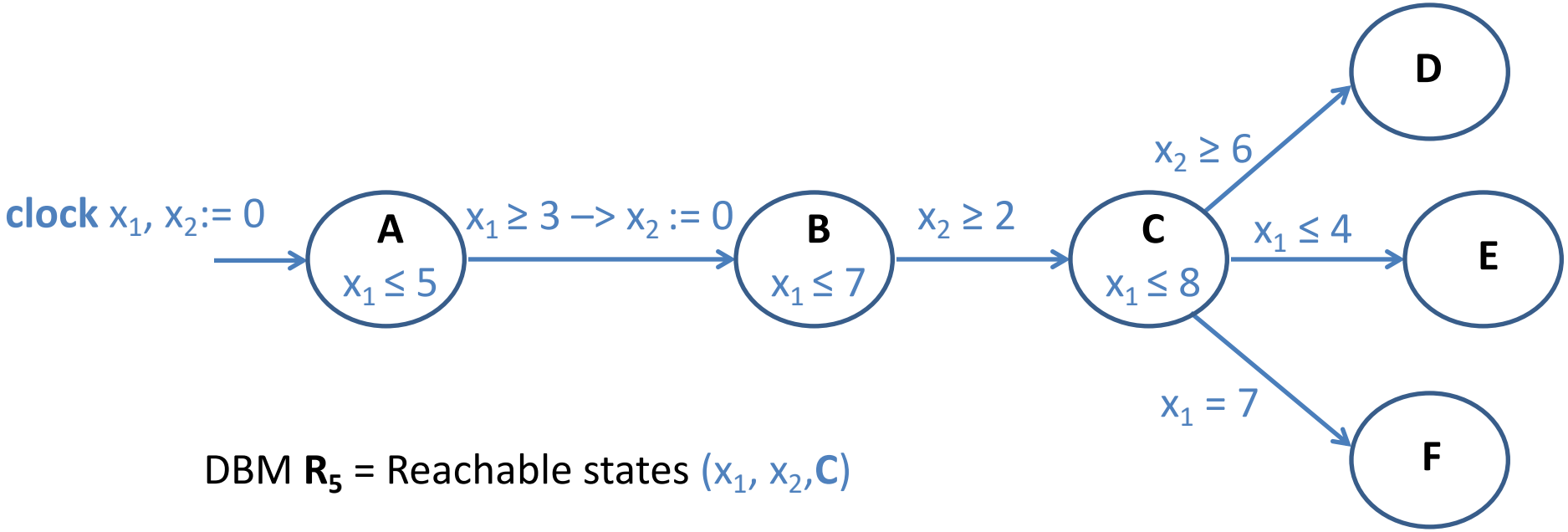
$$\text{for } 0 \leq i \leq m, \text{ for } 0 \leq j \leq m, \; x_i - x_j \leq R[i,j]$$

- Diagonal entries should be zero: $x_i - x_j \leq 0$
- There is a one-to-one correspondence between DBMs and clock-zones
- **Entries of DBM:** Integers plus a special symbol ∞ (to represent absence of a bound)

Timing Analysis Example

Clock-zone R_5

$$5 \leq x_1 \leq 8$$



DBM $\mathbf{R_5}$ = Reachable states (x_1, x_2,**C**)

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

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

Intersection of $\mathbf{R_5}$ 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