CENG 383

Real-Time Systems

Lecture 4

Formal Methods - I

Asst. Prof. Tolga Ayav, Ph.D.

Department of Computer Engineering

İzmir Institute of Technology

Software Requirements Specification

Consider the following excerpt from the Software Requirements Specification for the nuclear monitoring system:

1.1 *If interrupt A arrives, then task B stops executing.*

1.2 *Task A begins executing upon arrival of interrupt A.*

1.3 *Either Task A is executing and Task B is not, or Task B is executing and Task A is not, or both are not executing.*

Formalization:

p: interrupt A arrives
q: task B is executing
r: task A is executing

1.1 $p \Rightarrow \neg q$

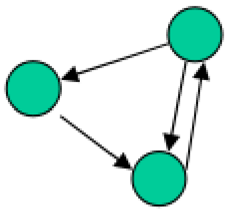
1.2 $p \Rightarrow r$

1.3 $(r \wedge \neg q) \vee (q \wedge \neg r) \vee (\neg q \wedge \neg r)$

Consistency Check of Requirements

	1	2	3	4	5	6	7	8
	p	q	r	$\neg q$	$\neg r$	$p \Rightarrow q$	$p \Rightarrow r$	$(r \wedge \neg q) \vee (q \wedge \neg r) \vee (\neg q \wedge \neg r)$
1	T	T	T	F	F	T	T	F
2	T	T	F	F	T	T	T	T
3	T	F	T	T	F	T	T	T
4	T	F	F	T	T	T	T	T
5	F	T	T	F	F	F	F	F
6	F	T	F	F	T	F	T	T
7	F	F	T	T	F	T	F	T
8	F	F	F	T	T	T	T	T

Finite State Machines



Finite state machines (FSMs) are powerful design elements used to implement algorithms in hardware.

An FSM is a 6-tuple, <Z, X, Y, δ, λ, z0>, where:

Z is a set of states {z0, z1, ... , zl},

X is a set of inputs {x0, x1, ..., xm},

Y is a set of outputs {y0, y1, ..., yn},

δ is a next-state function (i.e., transitions), mapping states and inputs to states, ($Z \times X \rightarrow Y$)

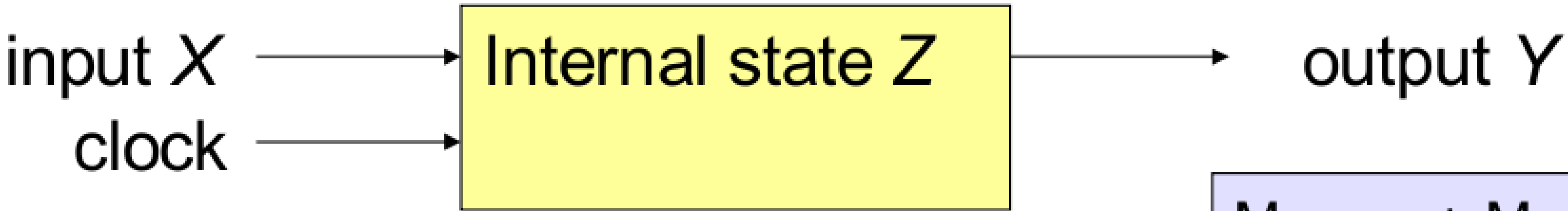
λ is an output function, mapping current states to outputs ($Z \rightarrow Y$),

and

z0 is an initial state.

Moore and Mealy FSMs

Classical automata:

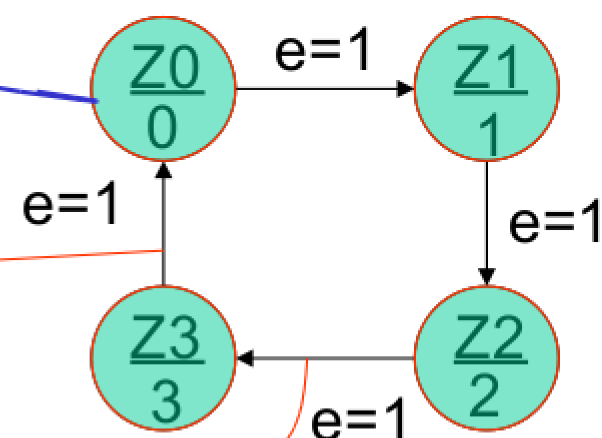


Next state Z^+ computed by function δ

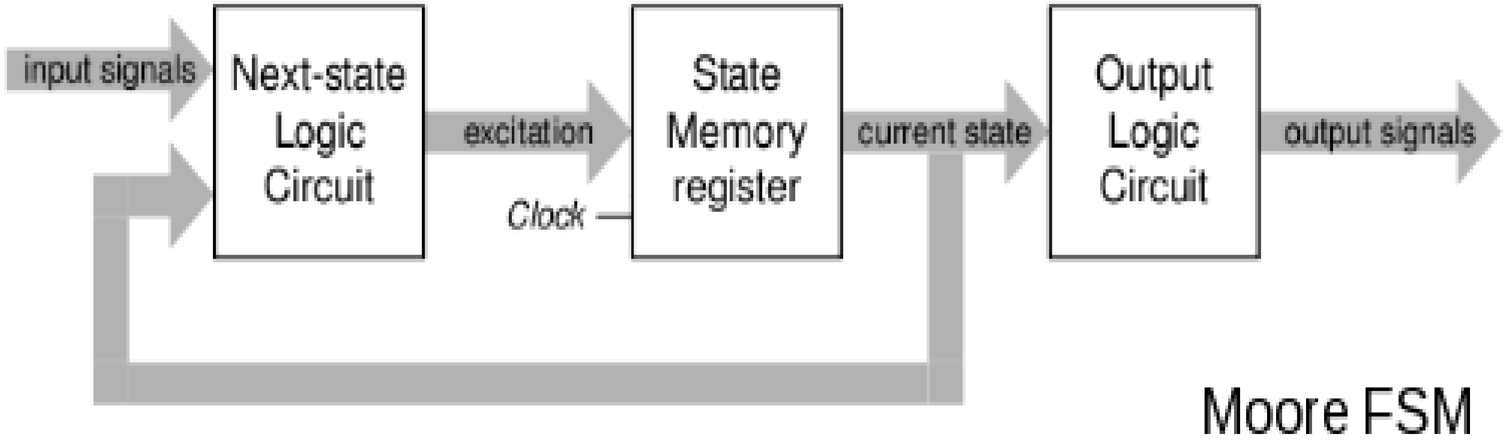
Output computed by function λ

Moore- + Mealy automata=finite state machines (FSMs)

- Moore-automata:
  $Y = \lambda(Z); \quad Z^+ = \delta(X, Z)$
- Mealy-automata
  $Y = \lambda(X, Z); \quad Z^+ = \delta(X, Z)$

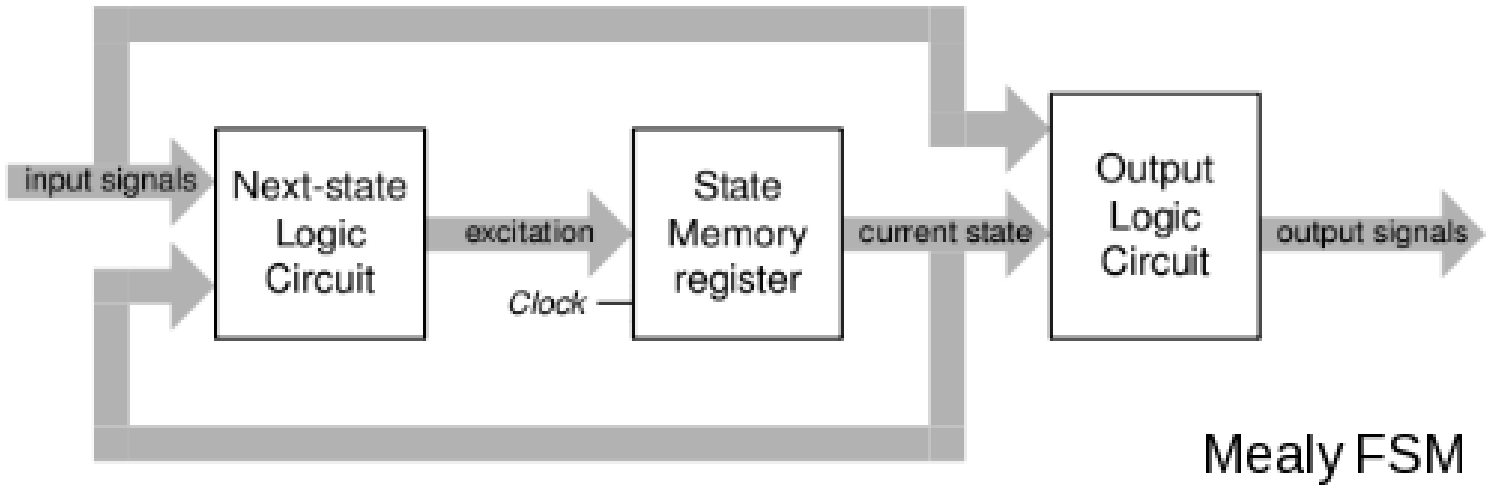


FSM Model: Moore



Outputs depends on states. $\lambda : (Z \rightarrow Y)$

FSM Model: Mealy

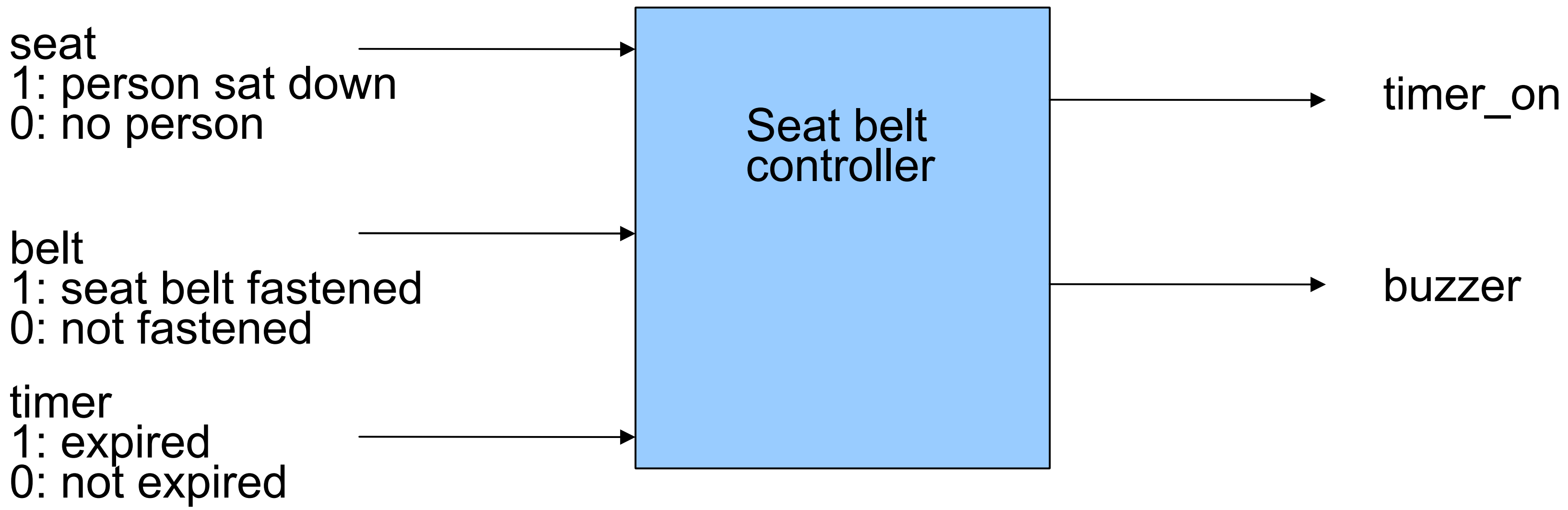


Outputs depends on states and inputs. $\lambda : (X \times Z \rightarrow Y)$

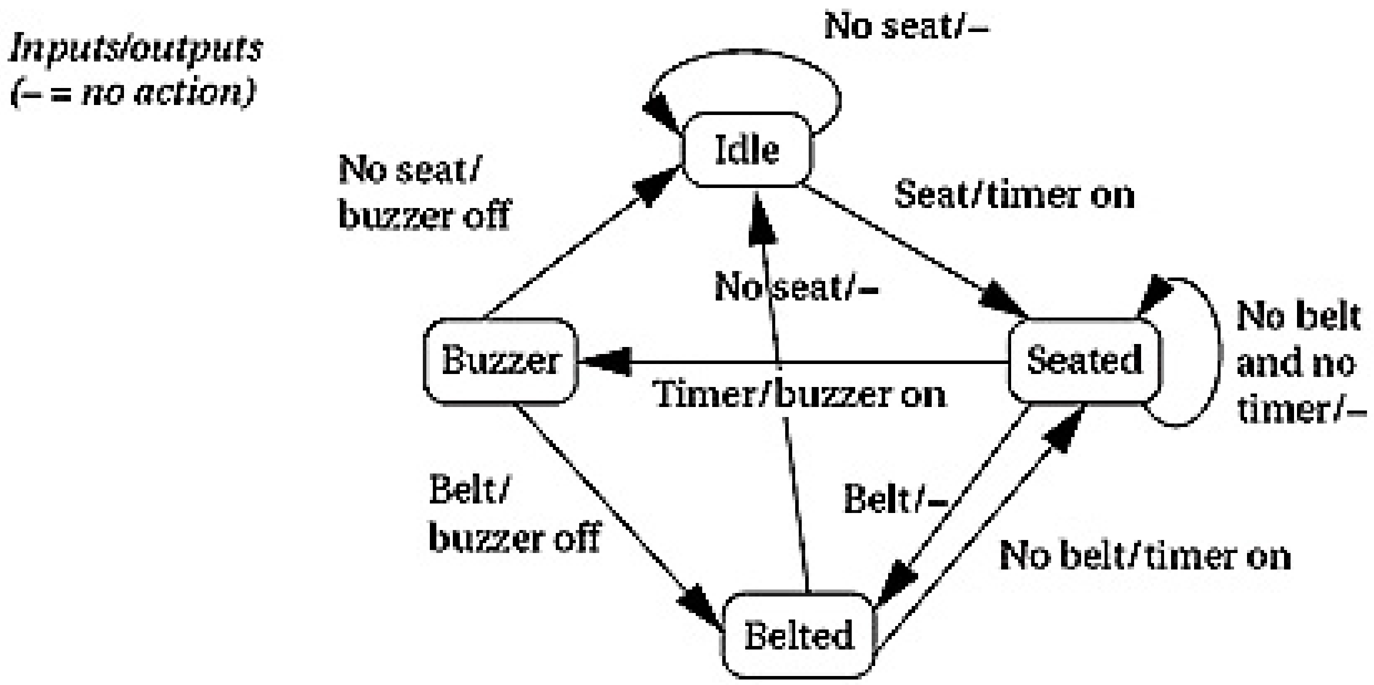
State Machine Example

Design a Simple Seat Belt Controller:

The controller's job is to turn on a buzzer if a person sits in a seat and does not fasten the seat belt within a fixed amount of time. This system has three inputs and one output.
The inputs are a sensor for the seat to know when a person has sat down, a seat belt sensor that tells when the belt is fastened, and a timer that goes off when the required time interval has elapsed. The output is the buzzer



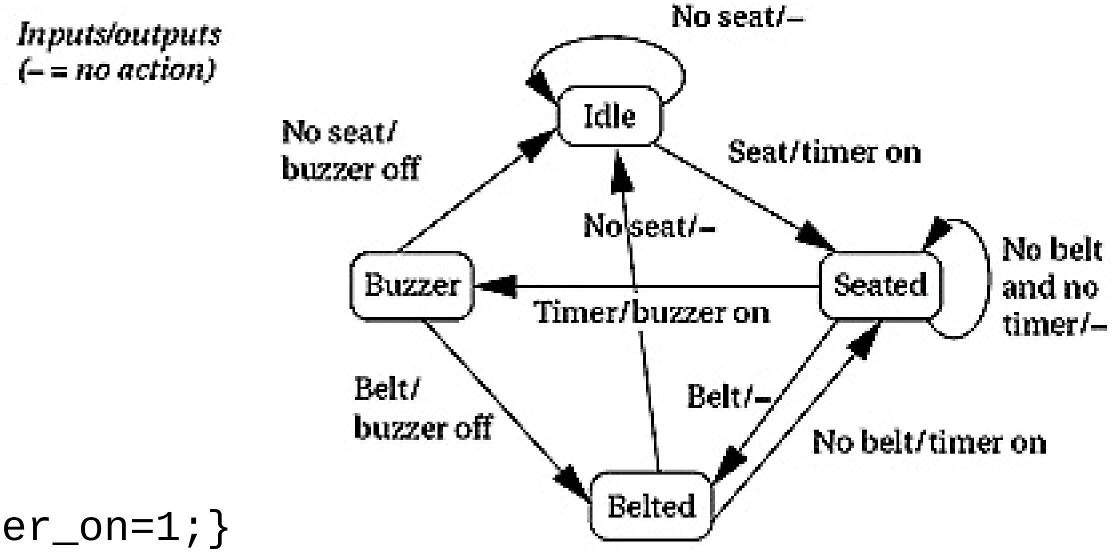
FSM for the Seat Belt Controller



C Code:

```c
#define IDLE 0
#define SEATED 1
#define BELTED 2
#define BUZZER 3

void FSM()
{
 switch(state)
 {
  case IDLE:
    if(seat) {state=SEATED; timer_on=1;}
  break;
  case SEATED:
    if(belt) state=BELTED;
    else if(timer) state=BUZZER;
  break;
  case BELTED:
    if(!seat) state=IDLE;
    else if(!belt) state=SEATED;
  break;
  case BUZZER:
    if(belt) state=BELTED;
    else if(!seat) state=IDLE;
  break;
 }
}
```



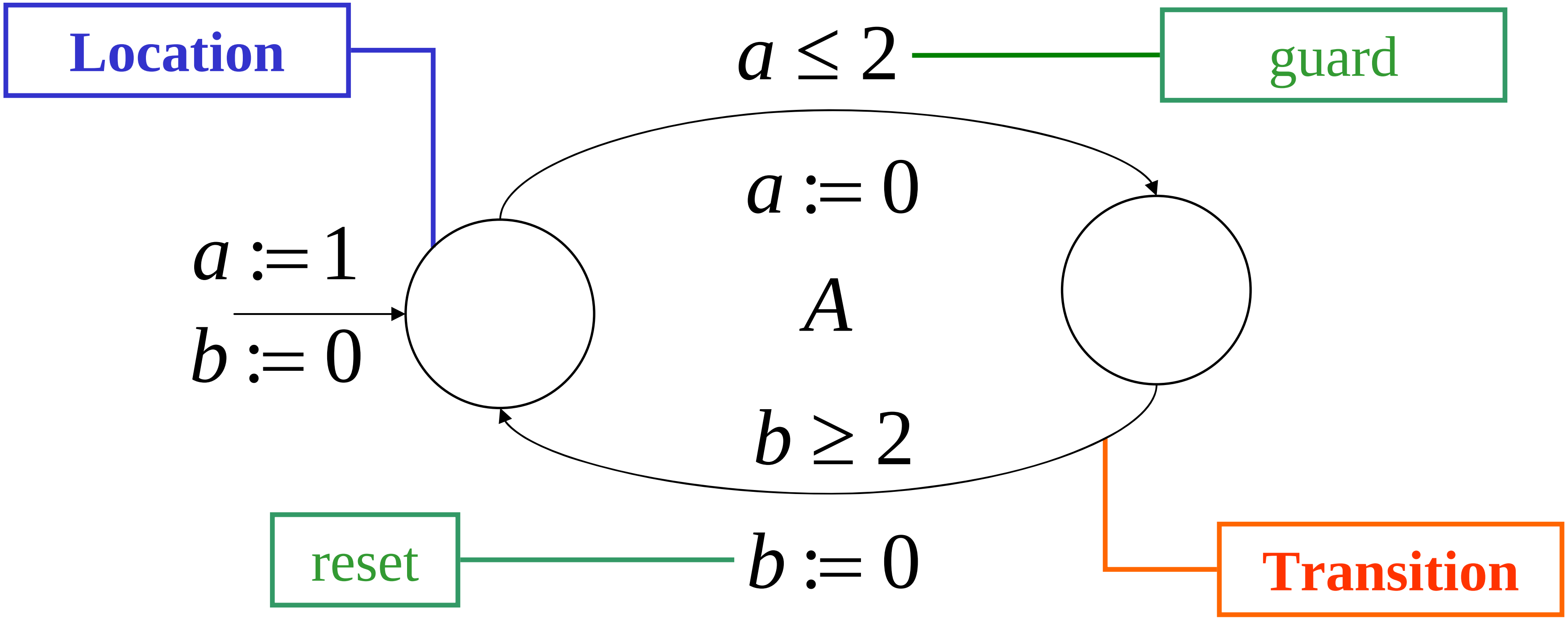
State diagram

```c
void main()
{

  while(1) FSM();

}
```

Timed Automata



Clocks: {a,b}

Formal Definitions-I

Timed Automata

Timed automata is a valuable tool for especially designing real-time systems. Here, we represent VHDL programs with timed automata. Let X be a finite set of real valued clock variables and V be a finite set of real valued data variables. A constraint C is of the form:

$$C ::= z \odot k \quad | \quad z - y \odot k$$

where $z, y \in X$ or $V, k \in \mathbb{N}$ and $\odot \in \{\leq, <, =, >, \geq\}$.

Definition 1 *(Timed Automaton). A timed automaton is a tuple* $(Q, q_0, X, \Sigma, \delta, I)$ *where:*

- Q *is a finite set of locations.*
- $q_0 \in Q$ *is the initial location.*
- X *is a finite set of clock variables.*
- Σ *is the set of denoting actions.*
- $\delta \subseteq Q \times 2^C \times \Sigma \times 2^X \times Q$ *is the set of transitions.*
- $I : Q \rightarrow 2^C$ *assigns invariants to locations.*

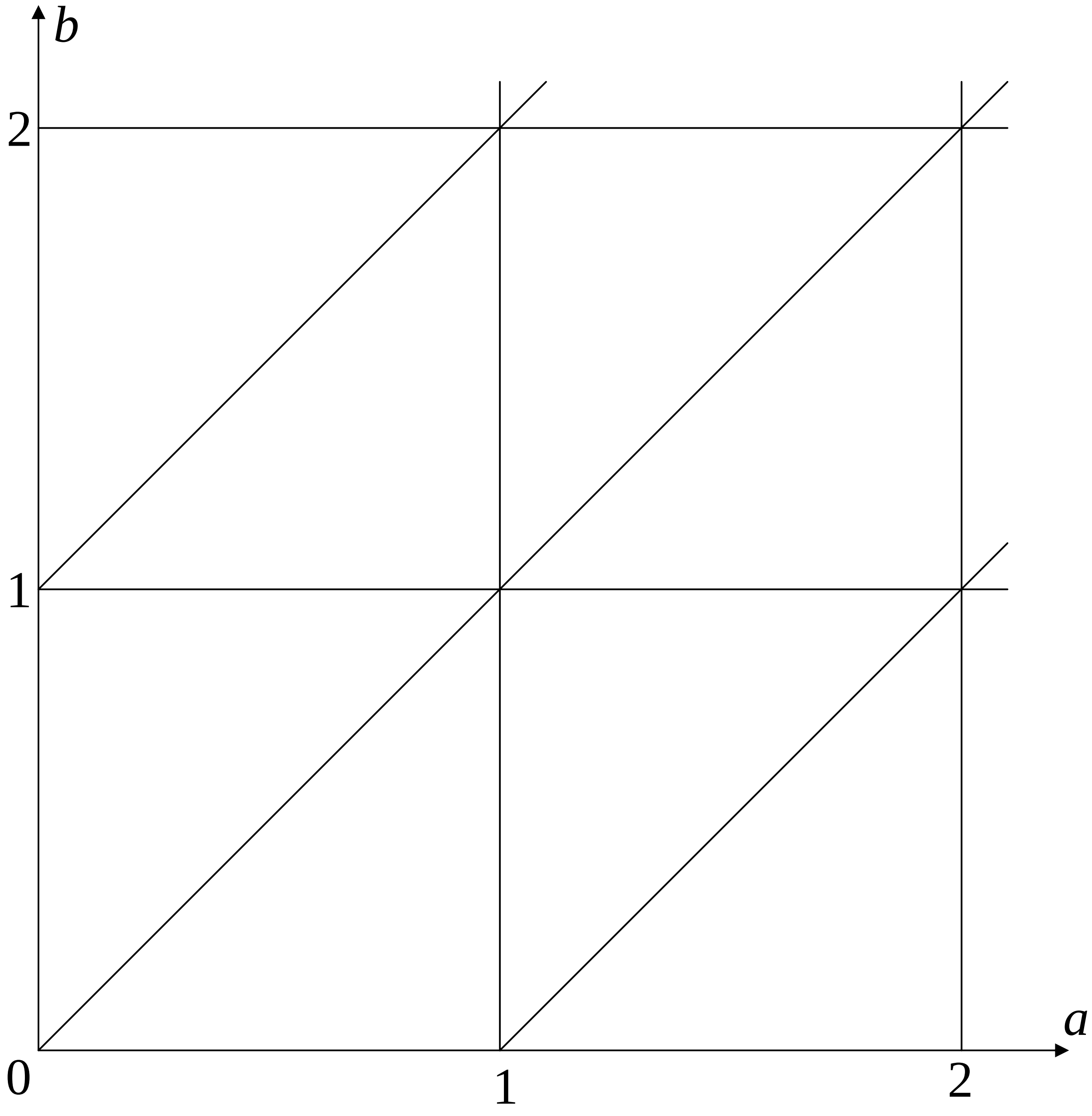
Formal Definitions-II

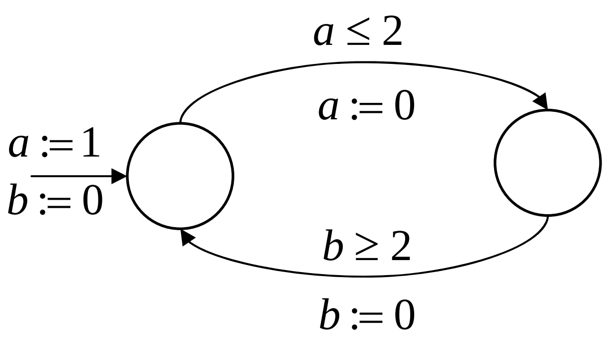
Definition 2 *(Semantics of Timed Automaton). Let* $(Q, q_0, X, \Sigma, \delta, I)$ *be a timed automaton. The semantics is given by a transition system* $\langle S, s_0, \rightarrow \rangle$ *where* $S \subseteq L \times \mathbb{R}^X$ *is the set of states,* $s_0 = (q_0, u_0)$ *is the initial state and* $\rightarrow \subseteq S \times \{\mathbb{R}_{\geq 0} \cup \Sigma\} \times S$ *is the transition relation such that:*

- $(q, u) \xrightarrow{d} (q, u+d)$ *if* $\forall d' : 0 \leq d' \leq d \Rightarrow u+d' \in I(q)$, *and*
- $(q, u) \xrightarrow{a} (q', u')$ *if* $\exists (q, g, a, r, q') \in \delta : u \in g$, $u' = [r \mapsto 0]u$ *and* $u' \in I(q)$.

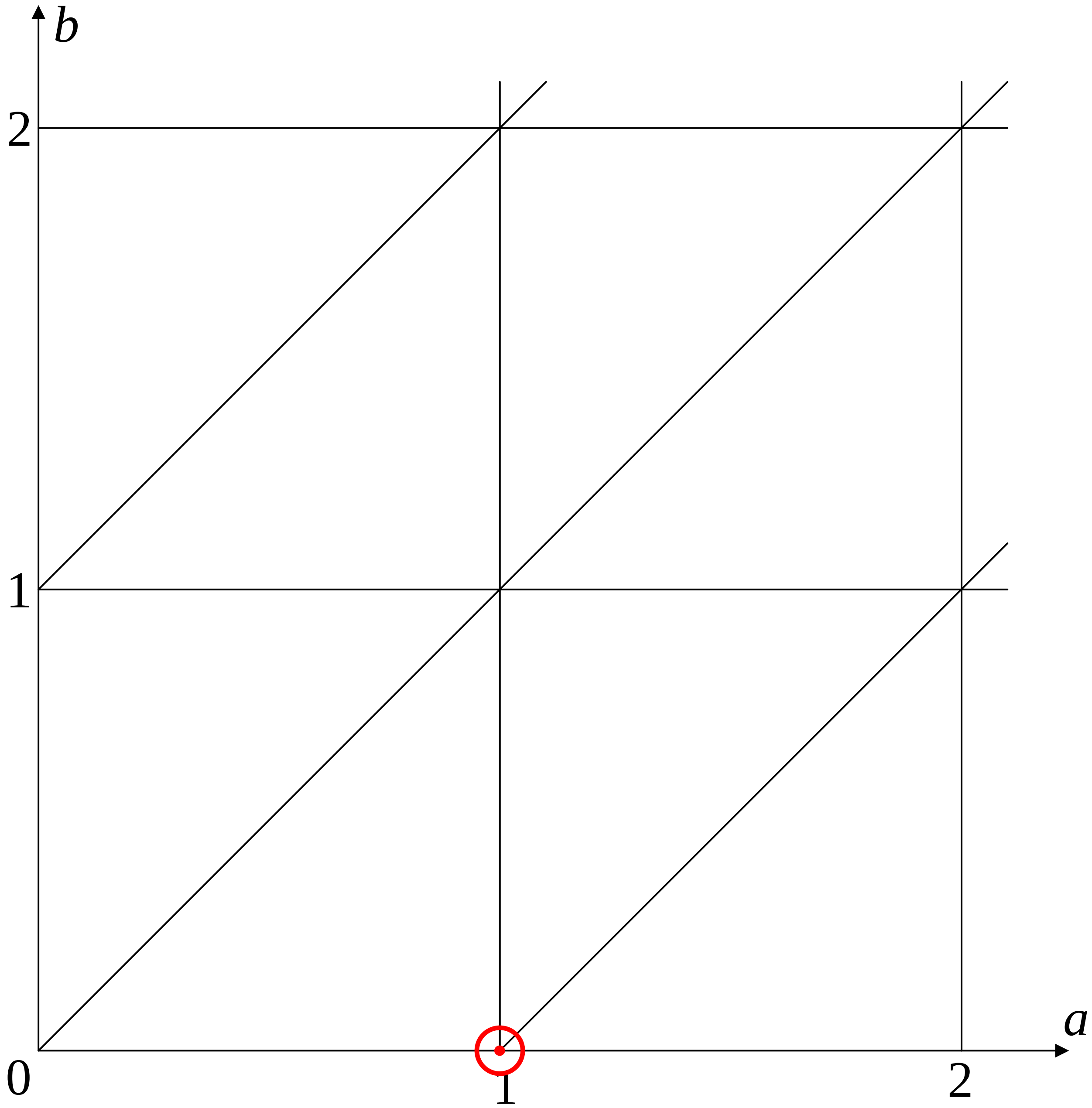
where for $d \in \mathbb{R}_{\geq 0}$, $u + d$ *maps each clock* x *in* X *to the value* $u(x) + d$, *and* $[r \mapsto 0]u$ *denotes the clock valuation which maps each clock in* r *to* 0 *and agrees with* u *over* $X \setminus r$.

Reachability Example

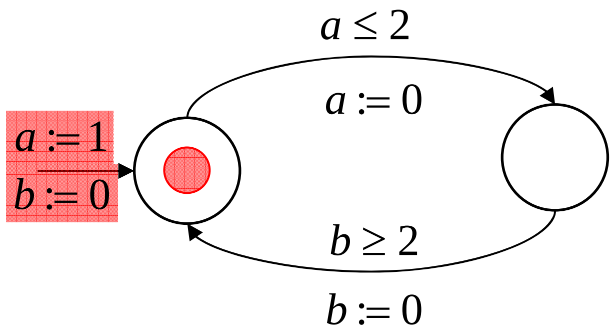




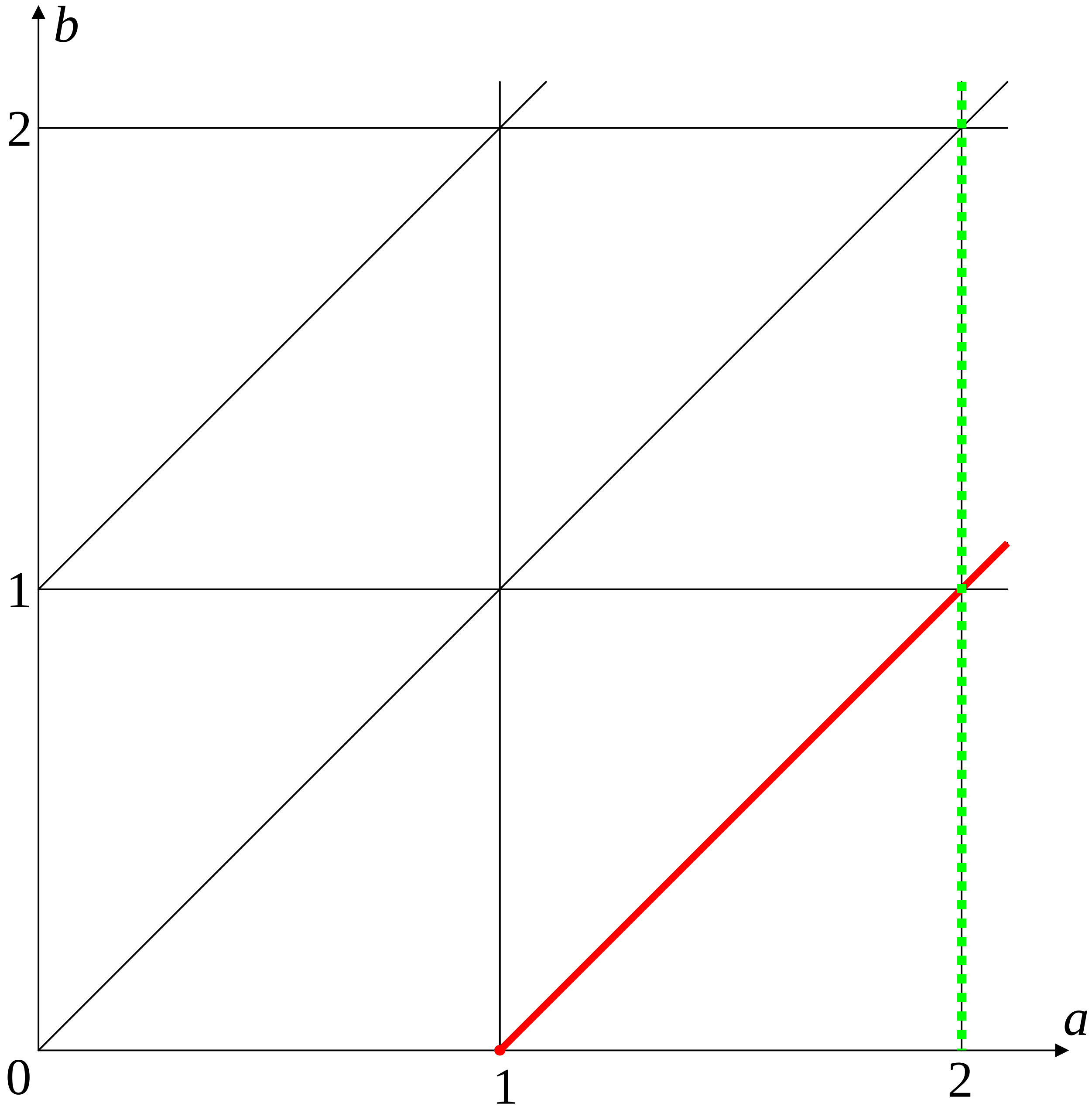
Example



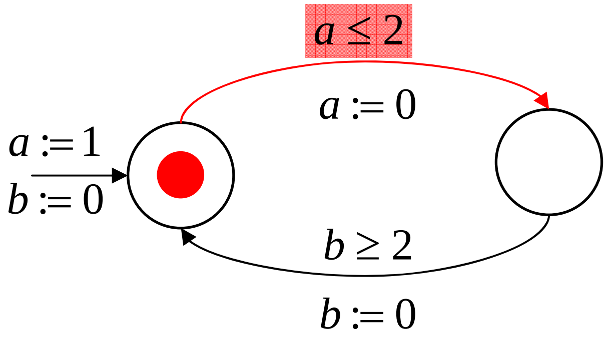
Classical Semantics



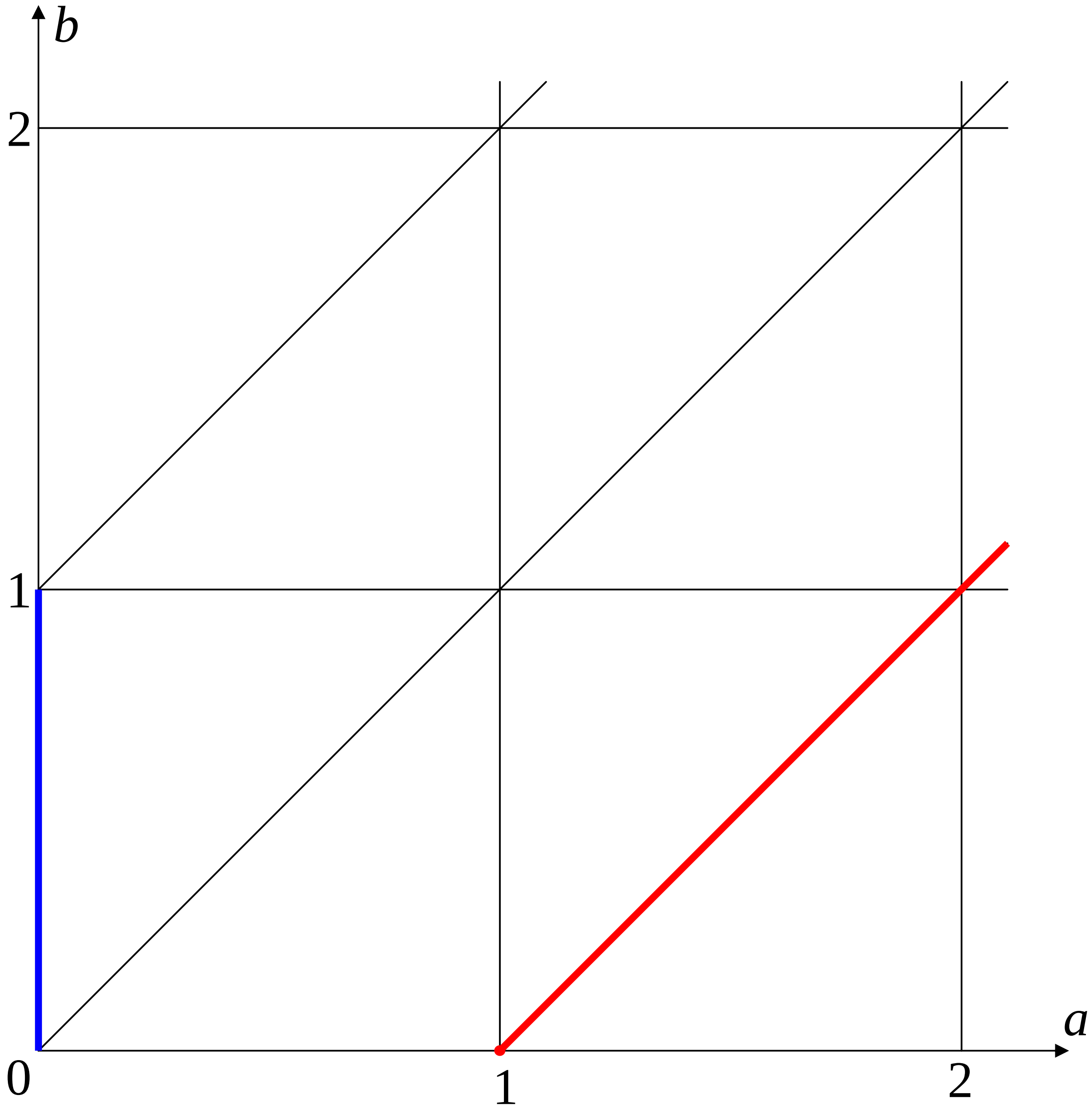
Example



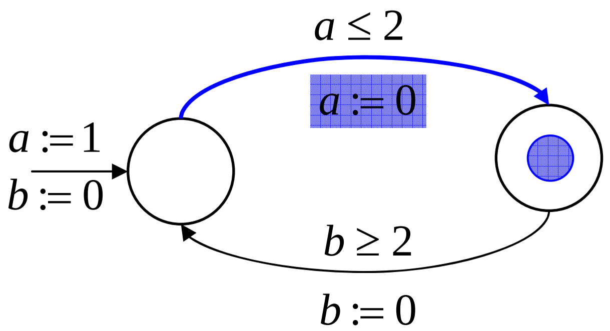
Classical Semantics



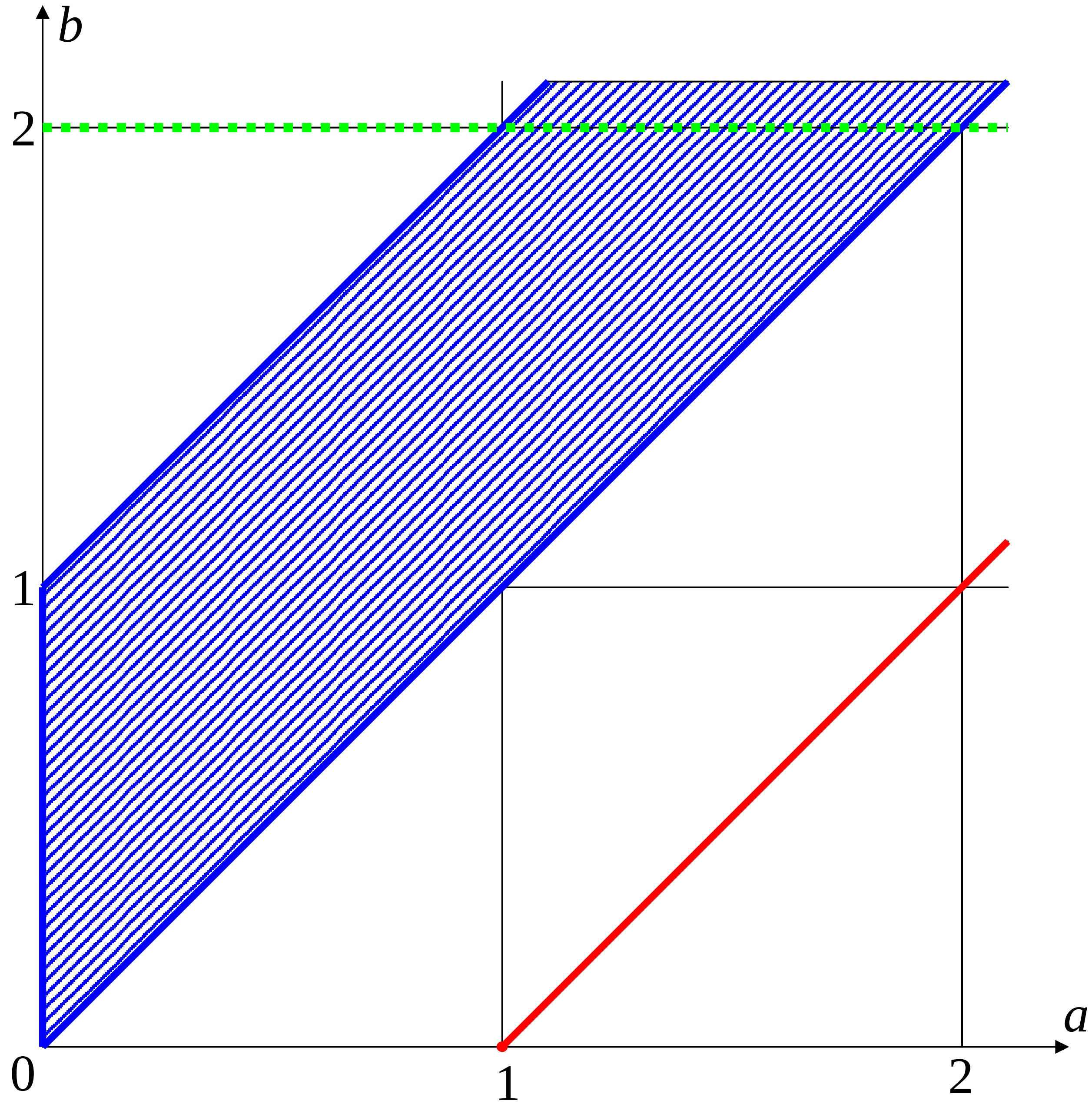
Example



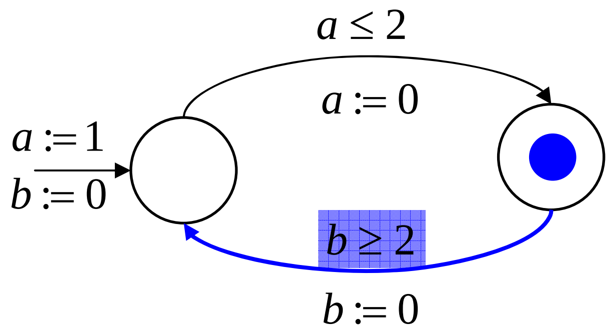
Classical Semantics



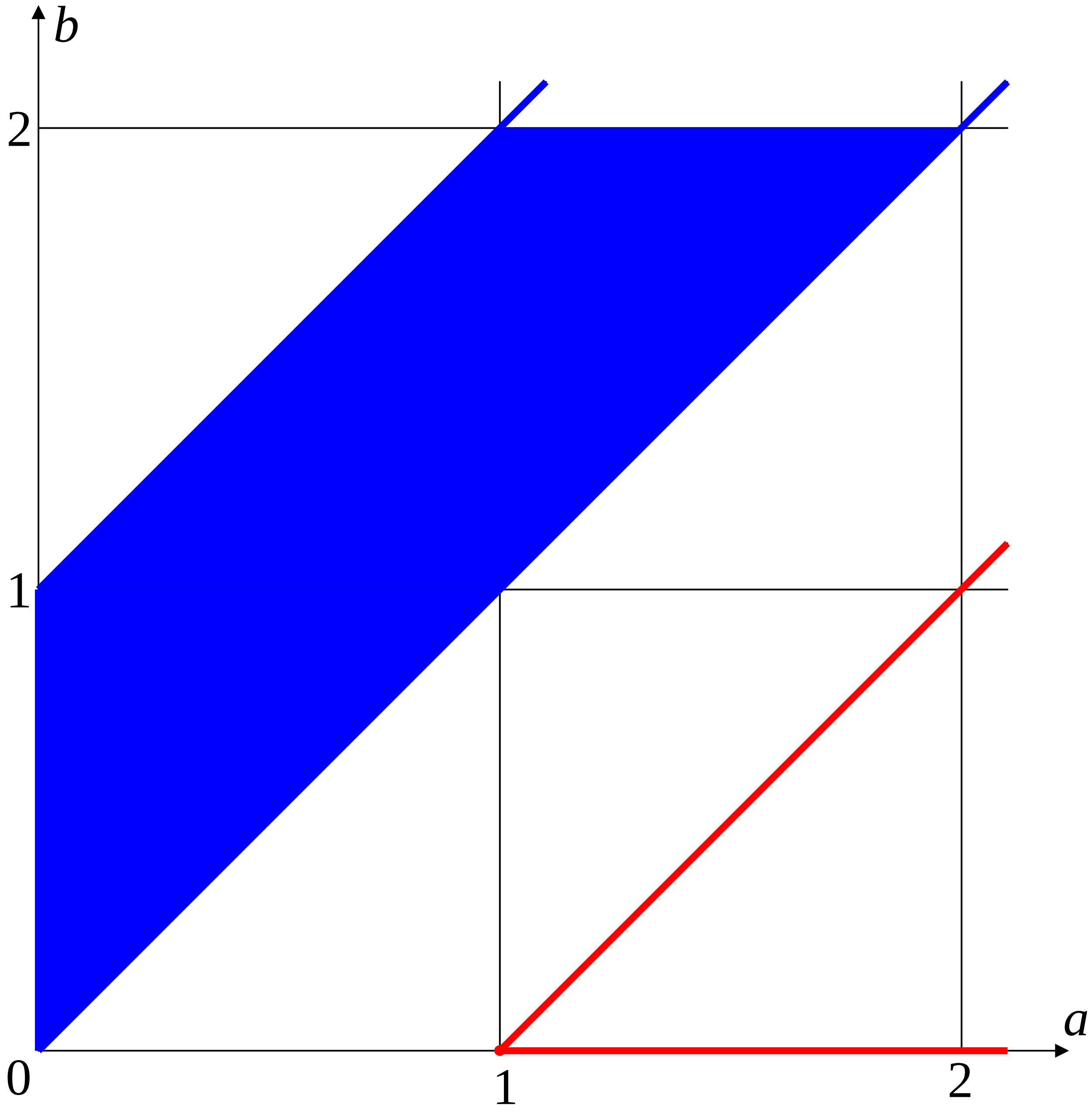
Example



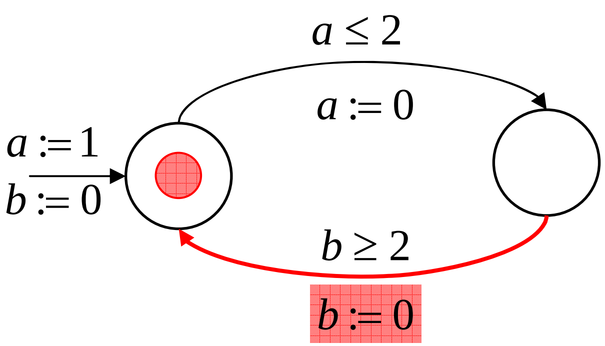
Classical Semantics



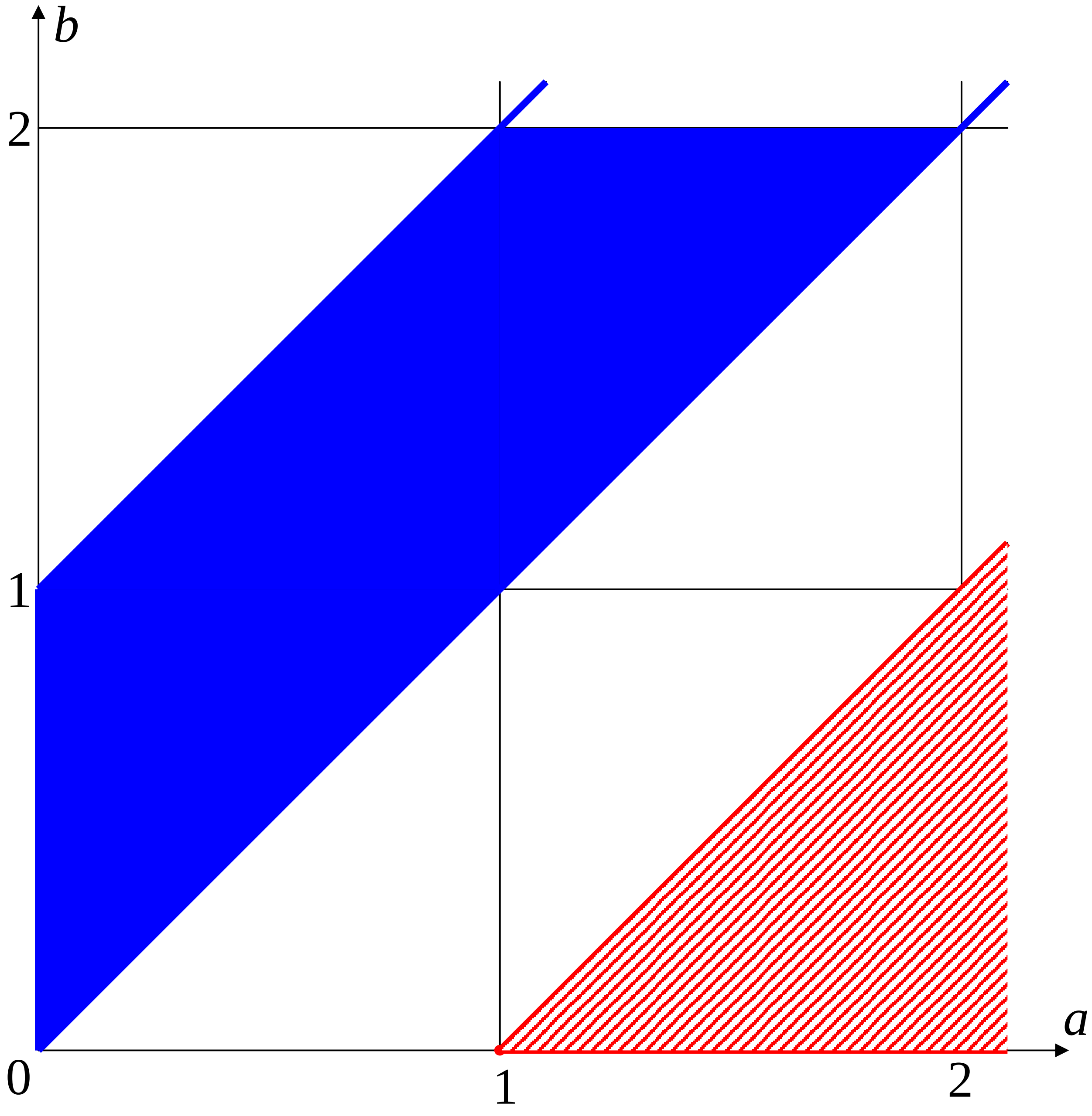
Example



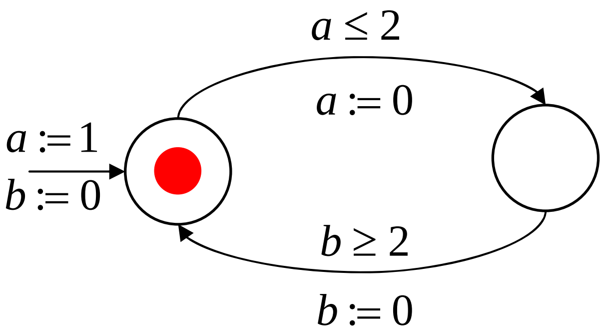
Classical Semantics



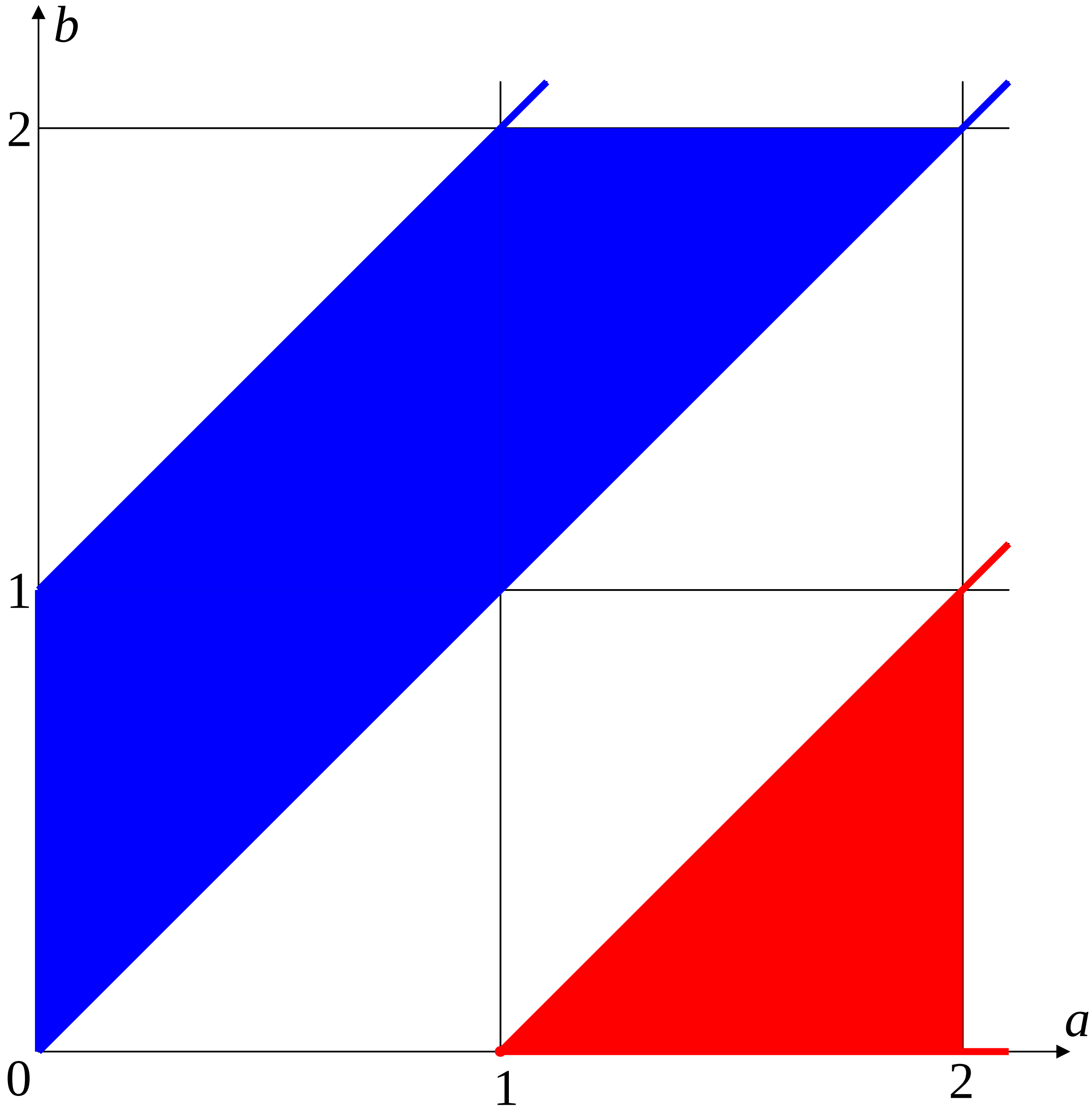
Example



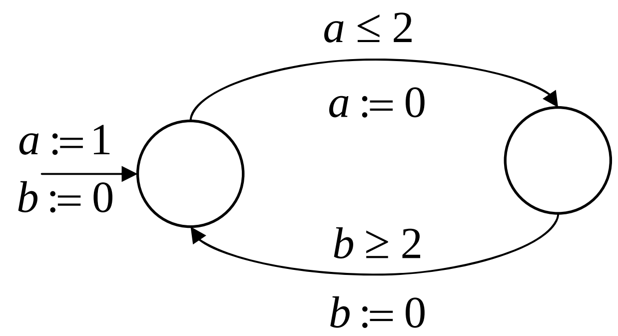
Classical Semantics



Example

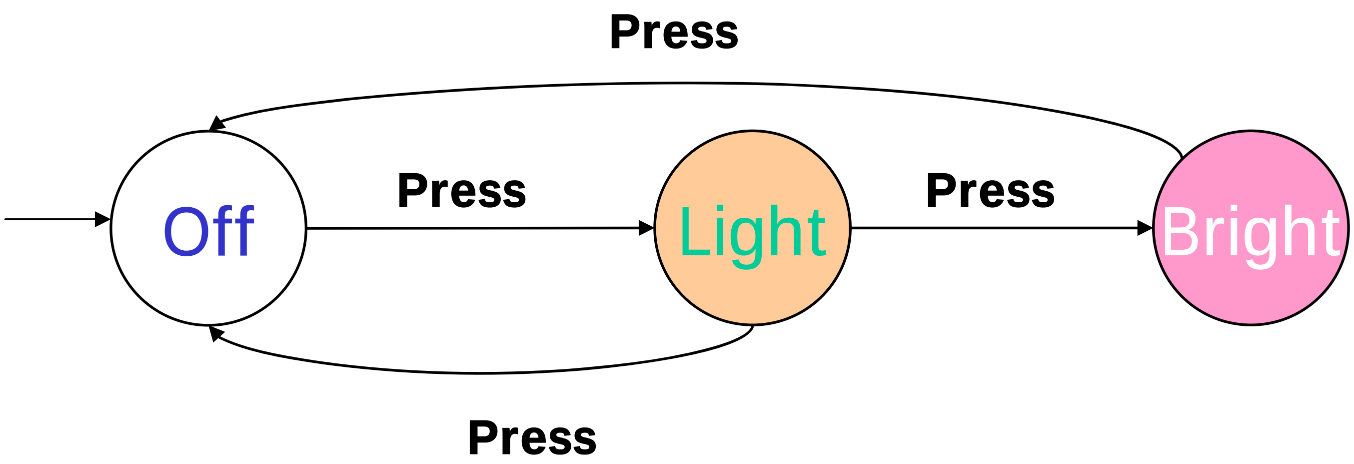


Classical Semantics



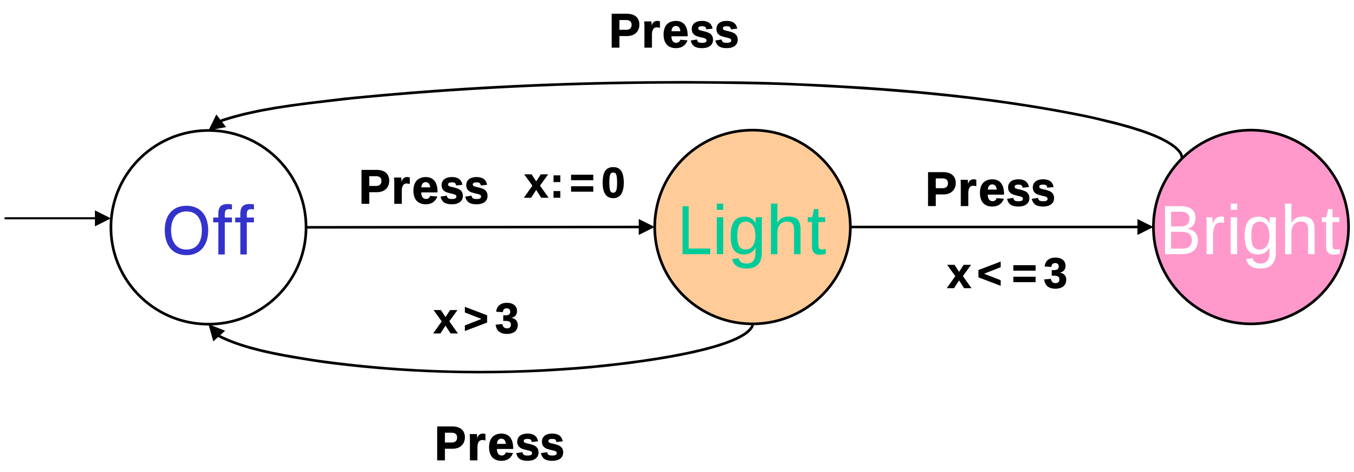
$$\boxed{\mathrm{Reach}([A])}$$

Simple Light Control



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

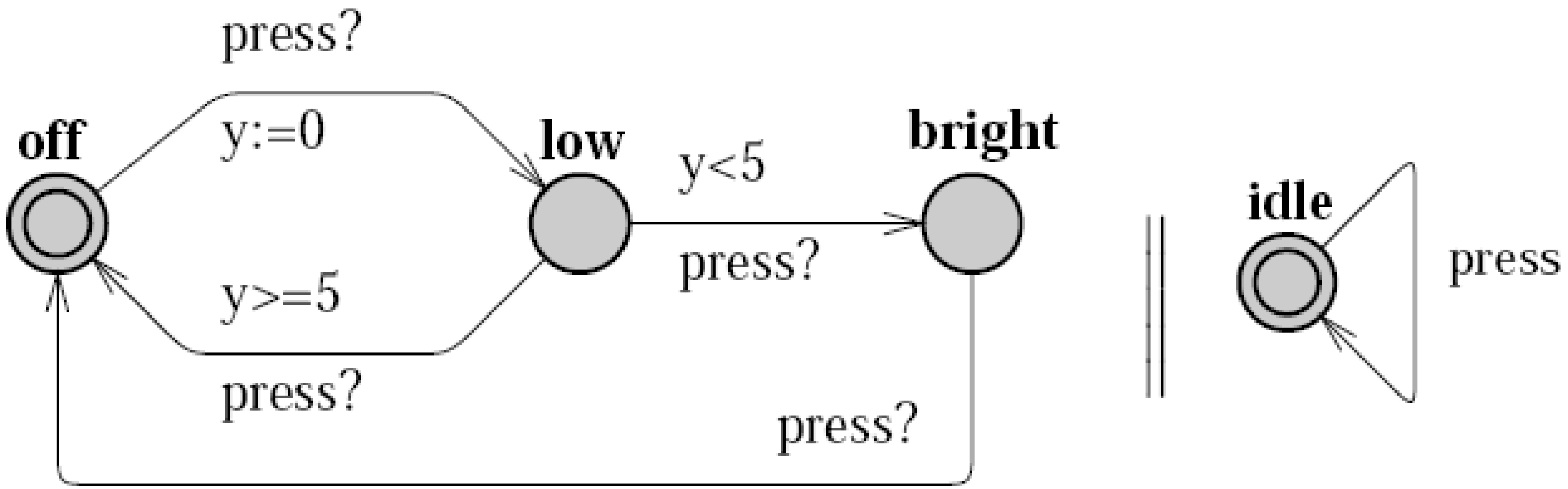
Simple Light Control



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

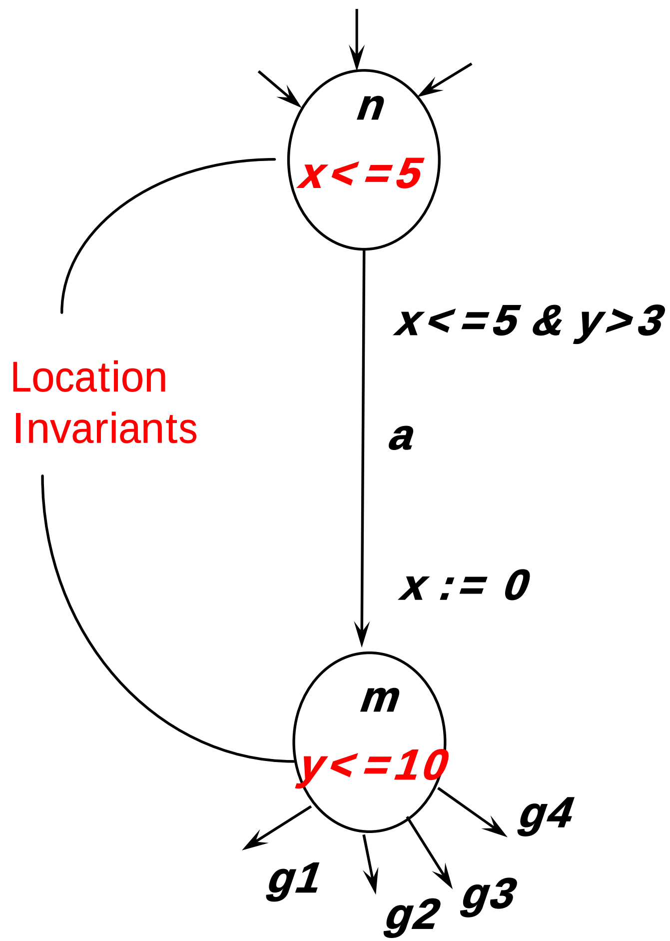
Adding continuous variables to state machines

Network of Timed Automata



(a) Lamp. (b) User.

Adding Invariants



Clocks: ***x, y***

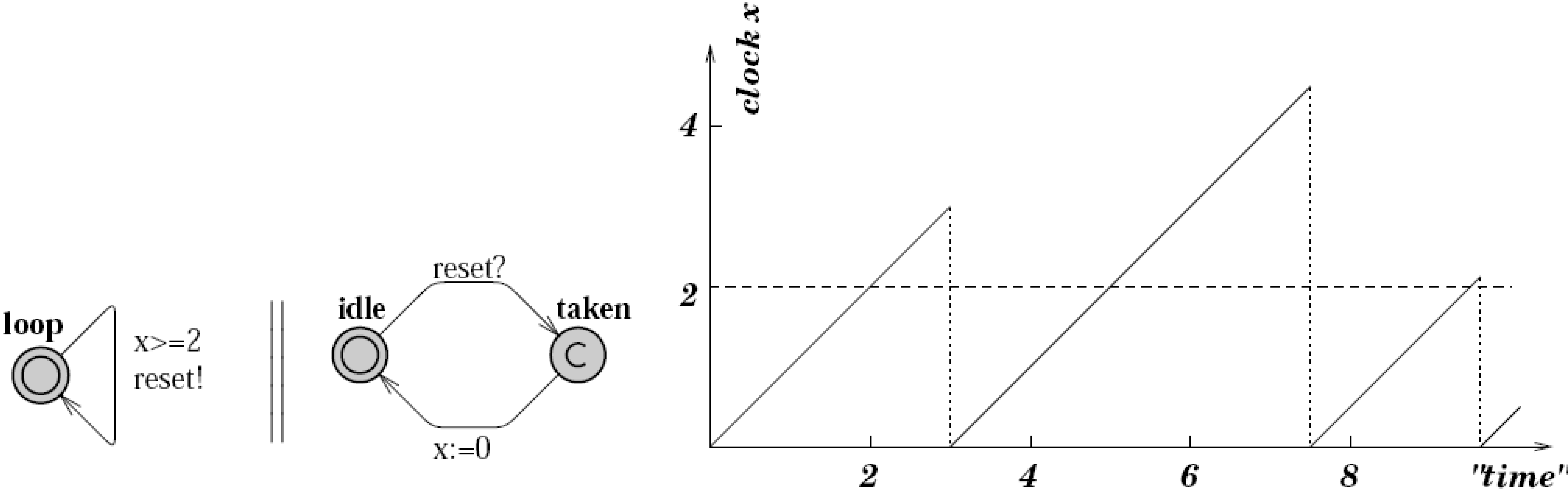
Transitions

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

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

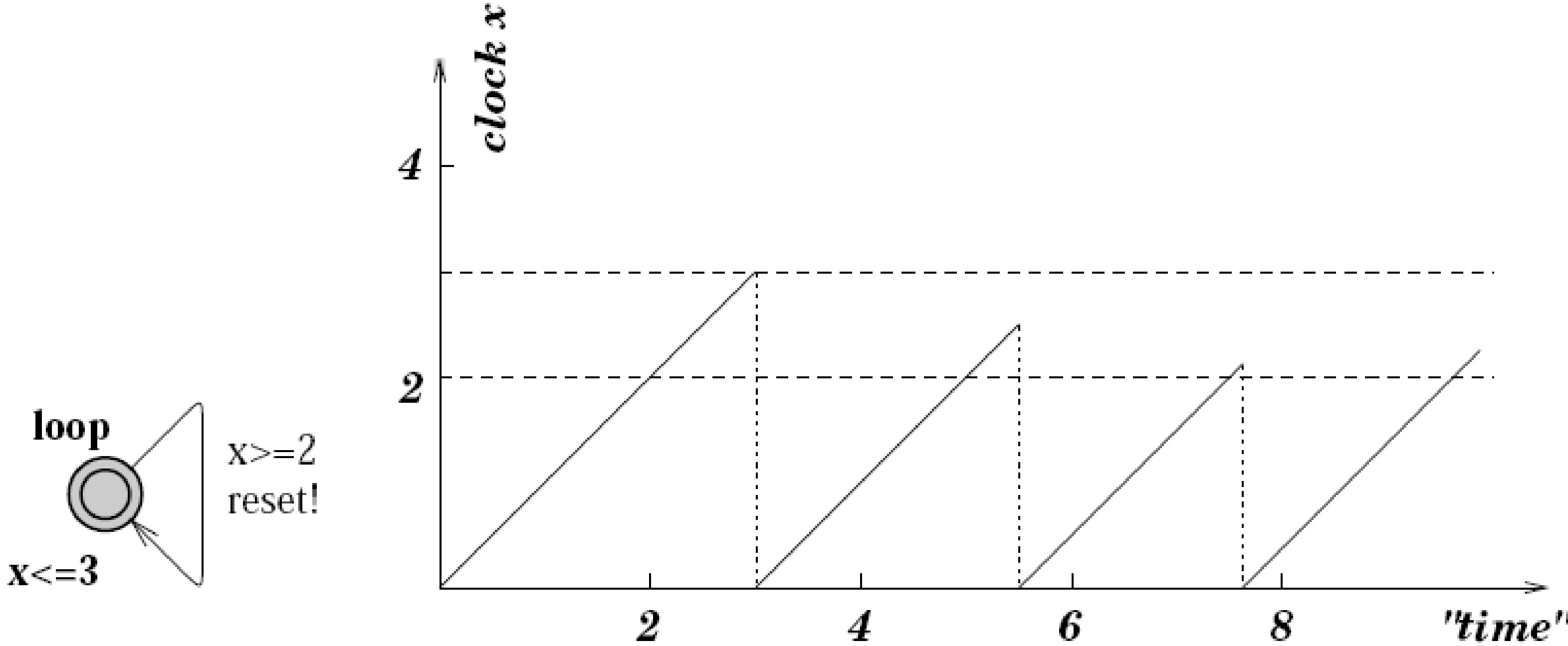
Invariants ensure progress!!

Timed Automata Examples and Timing Behaviours



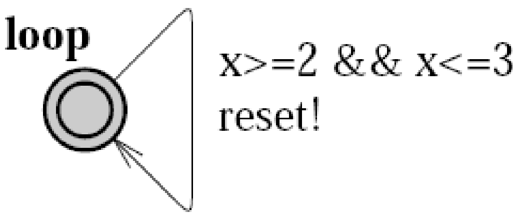
(a) Test. (b) Observer. (c) Behaviour: one possible run.

– `A[] Obs.taken imply x>=2` : all resets off `x` will happen when `x` is above 2. This query means that for all reachable states, being in the location `Obs.taken` implies that `x>=2`.
– `E<> Obs.idle and x>3` : this property requires, that it is possible to reacha state where `Obs` is in the location `idle` and `x` is bigger than 3. Essentially we check that we delay at least 3 time units between resets. The result would have been the same for larger values like 30000, since there are no invariants in this model.

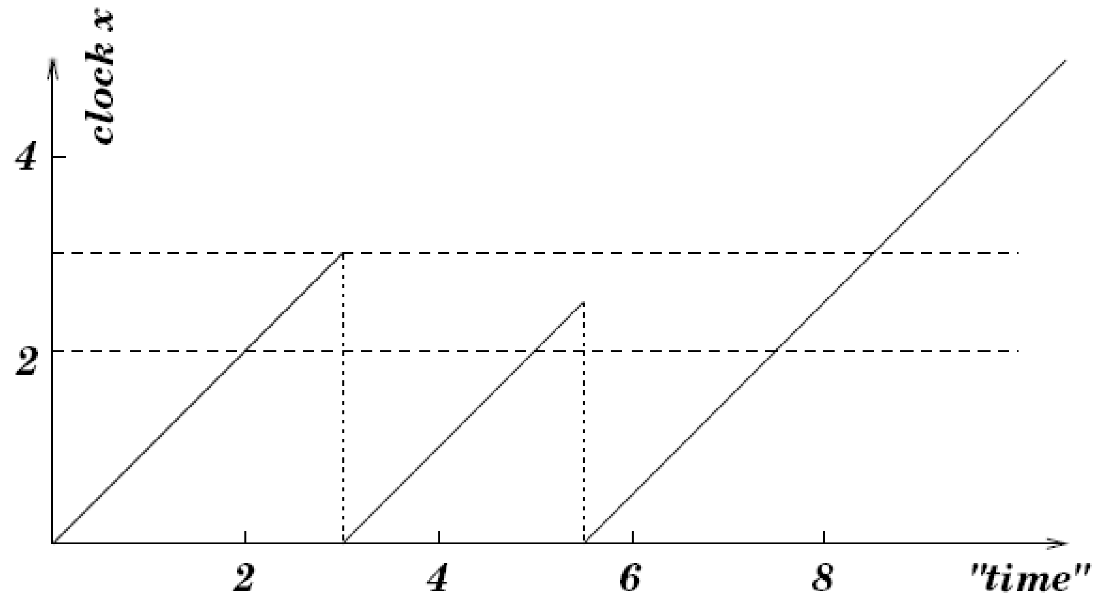


(a) Test. (b) Updated behaviour with an invariant.

- `A[] Obs.taken imply x>=2` : all resets off `x` will happen when `x` is above 2. This query means that for all reachable states, being in the location `Obs.taken` implies that `x>=2`.
- `E<> Obs.idle and x>3` : this property requires, that it is possible to reacha state where `Obs` is in the location `idle` and `x` is bigger than 3. Essentially we check that we delay at least 3 time units between resets. The result would have been the same for larger values like 30000, since there are no invariants in this model.



(a) Test.



(b) Updated behaviour with a guard and no invariant.

Property **A[] not deadlock** will not satisfy!

CTL (Computation Tree Logic)

SAFETY

$E\Diamond\psi$ (Possibly). There exists a path that property ψ eventually holds.

$A\Box\psi$ (Invariantly). Property ψ always holds.

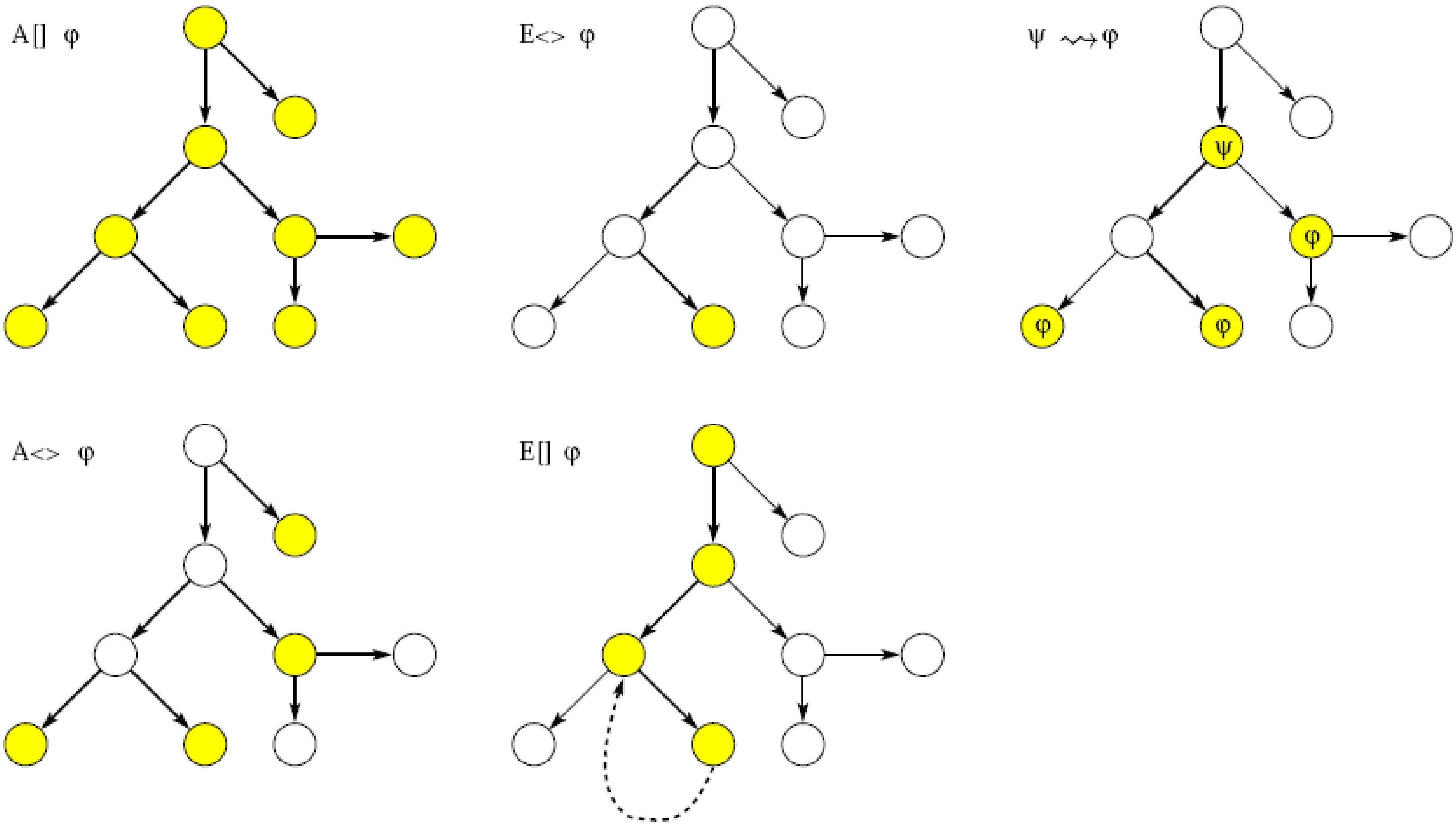
$E\Box\psi$ (Potentially always). There exists a path along which property ψ always holds.

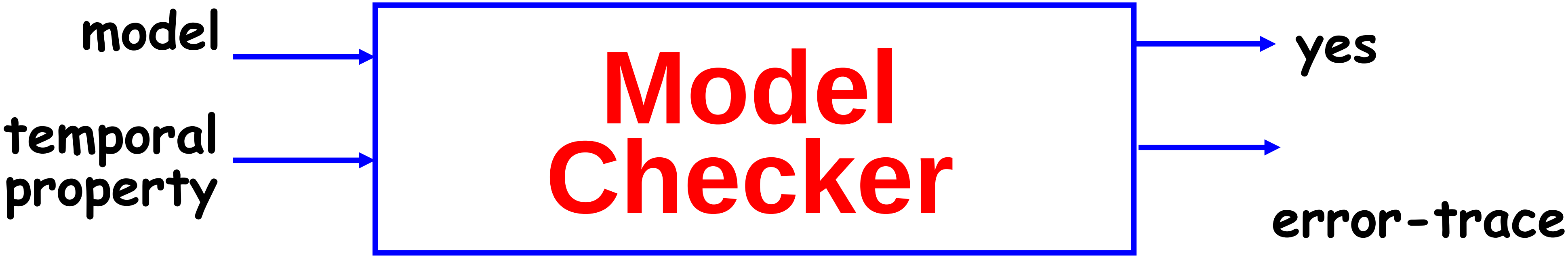
$A\Diamond\psi$ (Eventually). Property ψ eventually holds.

$\psi \leadsto \varphi$ (Leads-to). Whenever property ψ holds, property φ eventually holds.

LIVENESS

Path Formulae





Advantages

Automated formal verification, Effective debugging tool

Moderate industrial success

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

Commercial model checkers: FormalCheck by Cadence

Obstacles

Scalability is still a problem (about 500 state vars)

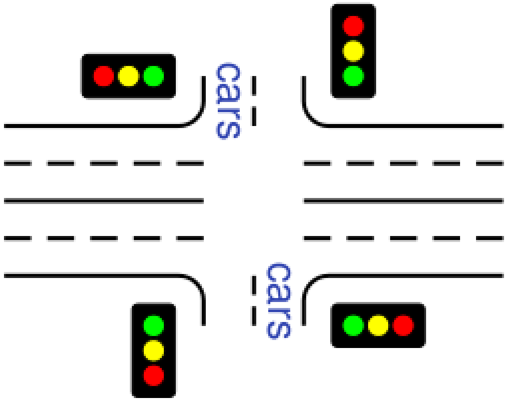
Effective use requires great expertise

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

UPPAAL: www.uppaal.com

- developed jointly by Uppsala university and Aalorg university
- UPPsala + AALborg = UPPAAL
  - SWEDEN + DENMARK = SWEDEN
  - SWEDEN + DENMARK = DENMARK

Assignment: Traffic Light Controller



Traffic light controller controls a traffic light at the intersection of a busy highway and a farm road. Normally, the highway light is green but if a sensor detects a car on the farm cars road, the highway light turns yellow then red. The farm road light then turns green until there are no cars or after a long timeout. Then, the farm road light turns yellow then red, and the highway light returns to green. The inputs to the machine are the car sensor, a short timeout signal, and a long timeout signal. The outputs are a timer start signal and the colors of the highway and farm road lights.

Create a Timed Automata to model this intersection.
Draw your timed automata in UPPAAL.