Safety and Verification

Lecture 19-21

Sayan Mitra
University of Illinois at Urbana-Champaign
mitras@Illinois.edu

Outline

- Why should you care about verification?
- Discrete models
  - CTL and CTL model checking
- Timed and hybrid models (cyber-physical systems)
  - Model checking timed and hybrid models
  - Simulation-driven verification

Building safe autonomous systems is going to be much harder than what we had imagined …

“Challenge is not so much building … but providing an assurance that these systems are safe” --- Dr. Sandeep Neema, DARPA program manager

Testing and verification will be central to this enterprise

Defense Advanced Research Projects Agency › News And Events

DARPA Assured Autonomy Seeks to Guarantee Safety of Learning-enabled Autonomous Systems

Program investigates ways to formalize and evolve functional and safety assurance for cyber-physical systems that learn

OUTREACH@DARPA.MIL
8/16/2017

How many miles must an autonomous car drive before we call it safe?

10 disengagements per 200 million miles?

0.07 fatalities per billion passenger miles (commercial flight)

Probability of fatal failure per hour of driving 10^{-9}

"30 billion miles of test driving is needed to achieve acceptable levels of assurance!"
[Koopman, CMU] [Shashua, CTO Mobileye]

Regulations and Audits

What fraction of the cost of developing a new aircraft is in SW?

DO178C

Primary document by which FAA & EASA approves software-based aerospace systems.

DAL establishes the rigor necessary to demonstrate compliance

Dev.Assurance Level (DAL)	Hazard Classification	Objectives
A	Catastrophic	71
B	Hazardous	69
C	Major	62
D	Minor	26
E	No Effect	0

Statement Coverage: Every statement of the source code must be covered by a test case

Condition Coverage: Every condition within a branch statement must be covered by a test case

"Special credits": For using formal methods based tools recently introduced

Crime records + Surveillance -> Predictions

2008: LAPD starts explorations on forecasting crime using data
2013: Better prediction of crime hotspots in Santa Cruz evaluation
2016: Used in 50+ police department

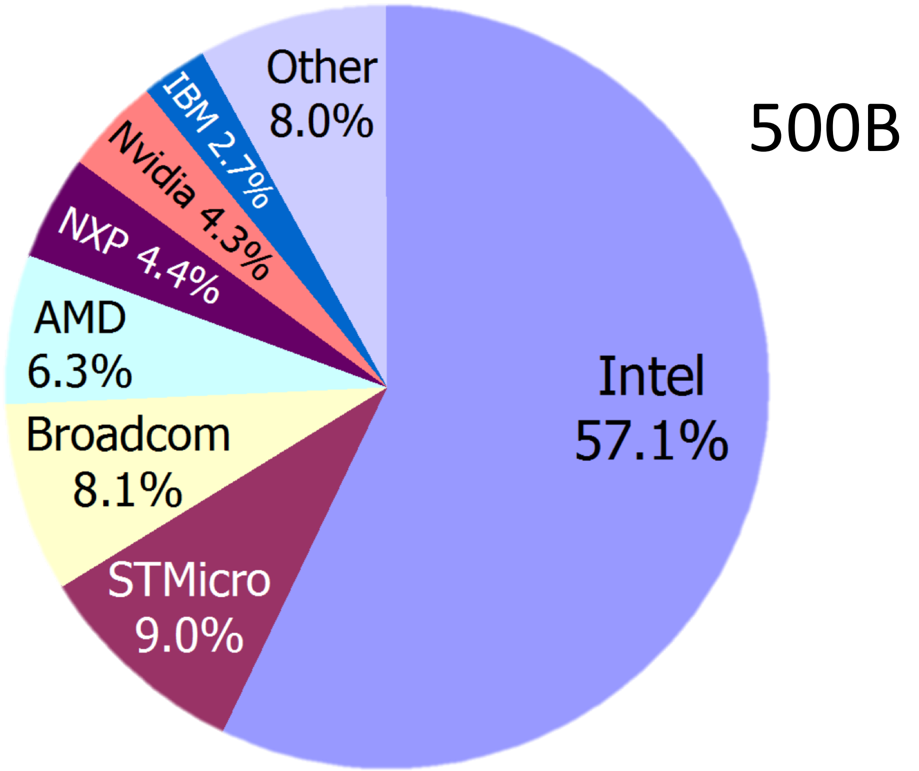
Zach Friend. "Predictive Policing: Using Technology to Reduce Crime". Federal Bureau of Investigation. Dec. 2013.

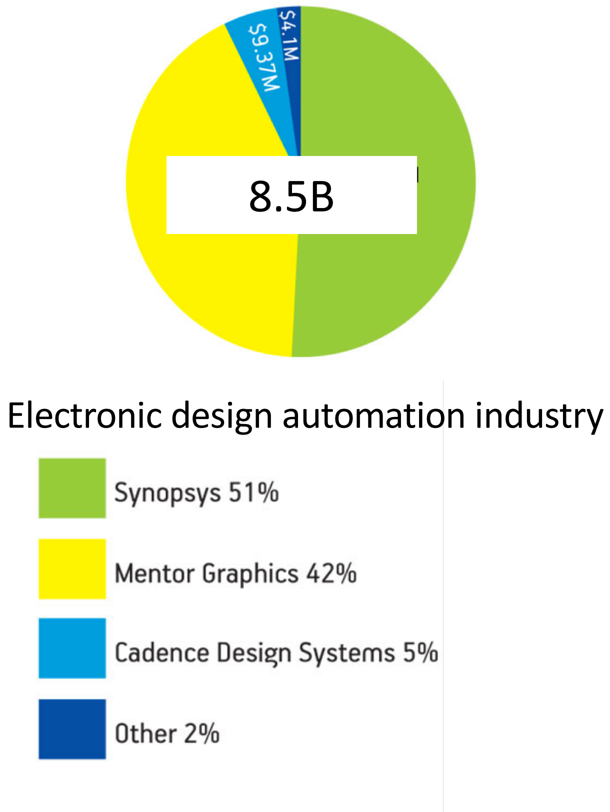
Is the algorithm fair?

Futureproof research area

Formal verification can provide:
standards, processes, tools, and trained individuals to ensure that cyber-physical systems meet the standards

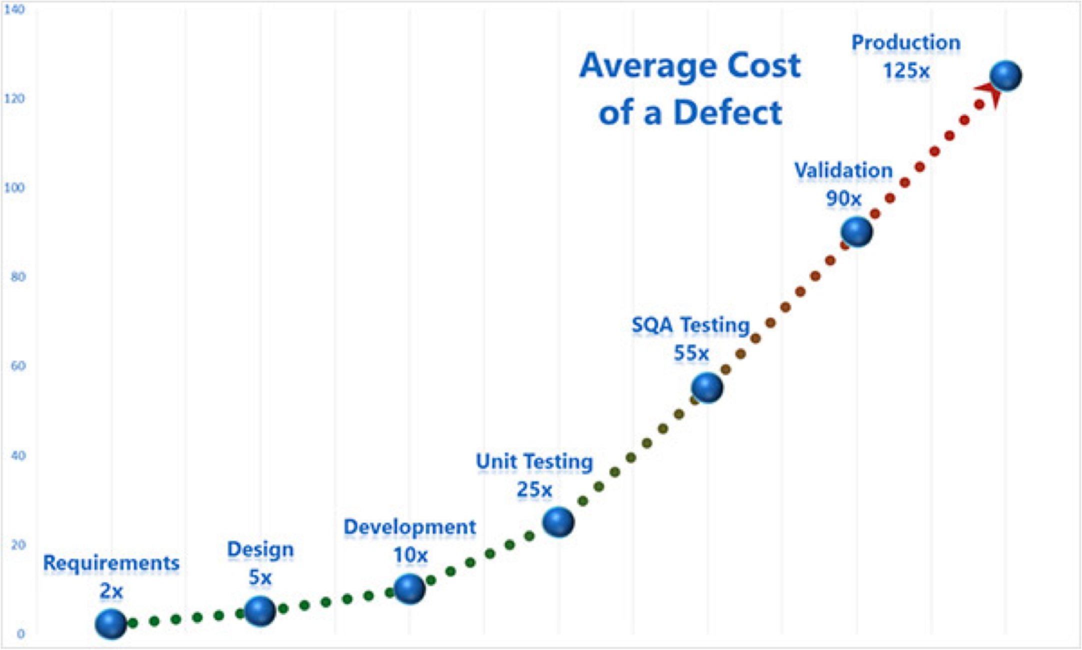
An earlier instance: microprocessor industry





Defects become more expensive with time

cost of bug



time

How to Cut Software-Related Medical Device Failures and Recalls, Lisa Weeks

Turing Award winners

(2018)
Bengio, Yoshua
Hinton, Geoffrey E
LeCun, Yann

(2017)
Hennessy, John L
Patterson, David

(2016)
Berners-Lee, Tim

(2015)
Diffie, Whitfield
Hellman, Martin

(2014)
Stonebraker, Michael

(2013)
Lamport, Leslie

(2012)
Goldwasser, Shafi
Micali, Silvio

(2011)
Pearl, Judea

(2010)
Valiant, Leslie Gabriel

(2009)
Thacker, Charles P. (Chuck) *

(2008)
Liskov, Barbara

(2007)
Clarke, Edmund Melson
Emerson, E. Allen
Sifakis, Joseph

(2006)
Allen, Frances ("Fran") Elizabeth

(2005)
Naur, Peter *

(2004)
Cerf, Vinton ("Vint") Gray
Kahn, Robert ("Bob") Elliot

(2003)
Kay, Alan

(2002)
Adleman, Leonard (Len) Max

(2000)
Yao, Andrew Chi-Chih

(1999)
Brooks, Frederick ("Fred")

(1998)
Gray, James ("Jim") Nicholas *

(1997)
Engelbart, Douglas *

(1996)
Pnueli, Amir *

(1995)
Blum, Manuel

(1994)
Feigenbaum, Edward A ("Ed")
Reddy, Dabbala Rajagopal ("Raj")

(1993)
Hartmanis, Juris
Stearns, Richard ("Dick") Edwin

(1992)
Lampson, Butler W

(1991)
Milner, Arthur John Robin Gorell ("Robin") *

(1990)
Corbato, Fernando J ("Corby")

(1989)
Kahan, William ("Velvel") Morton

(1988)
Sutherland, Ivan

(1987)
Cocke, John *

(1986)
Hopcroft, John E
Tarjan, Robert (Bob) Endre

(1985)
Karp, Richard ("Dick") Manning

(1984)
Wirth, Niklaus E

(1983)
Ritchie, Dennis M.*
Thompson, Kenneth Lane

(1982)
Cook, Stephen Arthur

(1981)
Codd, Edgar F. ("Ted") *

(1980)
Hoare, C. Antony ("Tony") R.

(1979)
Iverson, Kenneth E. ("Ken") *

(1978)
Floyd, Robert (Bob) W *

(1977)
Backus, John *

(1976)
Rabin, Michael O.
Scott, Dana Stewart

(1975)
Newell, Allen *
Simon, Herbert ("Herb") Alexander *

(1974)
Knuth, Donald ("Don") Ervin

(1973)
Bachman, Charles William *

(1972)
Dijkstra, Edsger Wybe *

(1971)
McCarthy, John *

(1970)
Wilkinson, James Hardy ("Jim") *

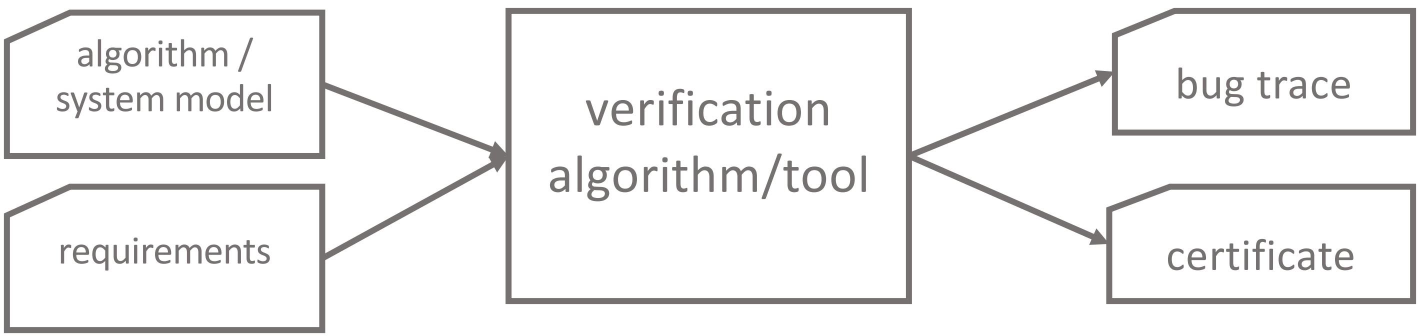
(1969)
Minsky, Marvin *

(1968)
Hamming, Richard W*

(1967)
Wilkes, Maurice V.*

(1966)
Perlis, Alan J *

Audit algorithms with Algorithms and find problems early



Relevant courses: Theory of computation, Program Verification, Formal System Development, Automated Deduction, Control theory, Embedded System Verification

Example requirements

Safety: “For all nominal behaviors of the car, the separation between the cars must be always > 1 m”

Efficiency: “For all nominal driver inputs, the air-fuel ratio must be in the range [1,4]”

Privacy: “Using GPS does not compromise user’s location”

Fairness: “Similar people are treated similarly”

Example modeling frameworks

Discrete transition systems, automata

Dynamical systems

Differential inclusions

Hybrid systems

Markov chains

Probabilistic automata, Markov decision processes (MDP)

Continuous time, continuous state MDPs

Stochastic Hybrid systems

Example verification approaches

- Theorem Proving (PVS, Isabelle, CoQ)
  - Automatic or Interactive
  - First Order vs Higher Order Logic
  - Decidable logics
  - Satisfiability Modulo Theory (SMT) solvers
- Model Checking
  - Explicit state or symbolic model checking
  - Abstraction Refinement
  - Symbolic executions
  - Probabilistic and statistical model checking
  - Data-driven verification
- Abstract Interpretation

Discrete Systems

Modeling Computation

Outline

- An Example: Token Ring
- Specification language (syntax)
- Automata (semantics)
- Invariants

An example: Informal description

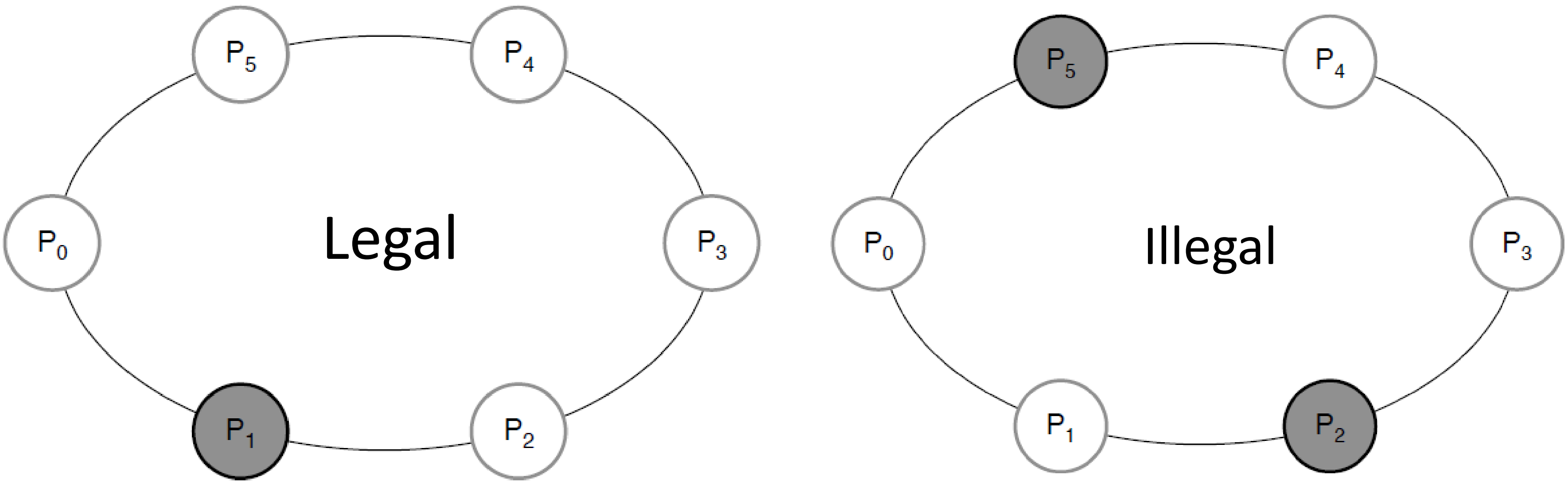
A token-based mutual exclusion algorithm on a ring network

- Collection of processes send and receive bits over a ring network so that only one of them has a "token"

Discrete

- Each process has variables that take only **discrete values**
- Time elapses in **discrete steps** (This is a modeling choice)

Token ring: Informal problem specification

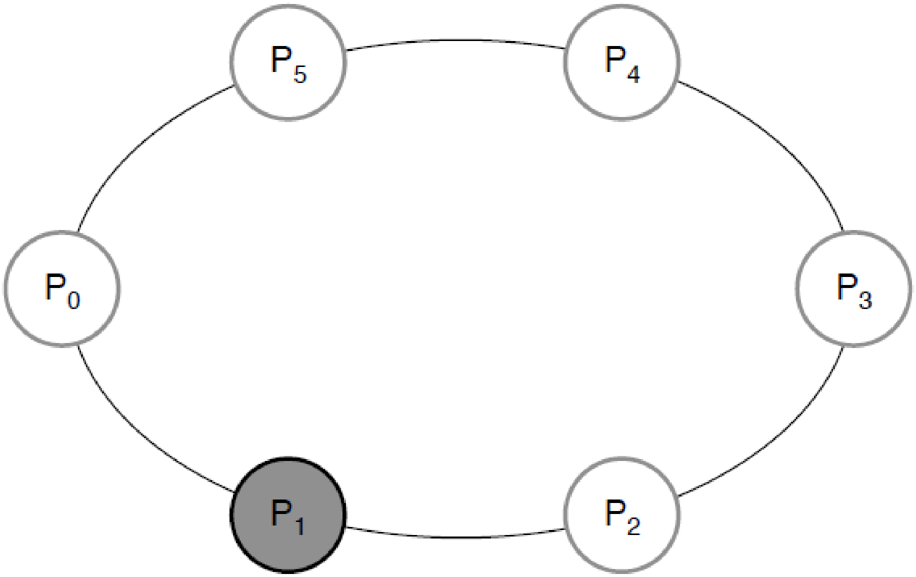


1. There is always at least one token
2. Legal configuration = exactly one “token” in the ring
3. Single token circulates in the ring
4. Even if multiple tokens somehow arise, e.g. with failures, if the algorithm continues to work correctly, then eventually there is a single token

Properties can be stated as Invariants

- Invariant (informal def.): A property of the system that always* holds
- Examples:
  - "Always at least one process has a token"
  - "Always exactly one process has the token"
  - "Always all processes have values at most k-1"
  - "Even if there are multiple tokens, eventually there is exactly one token" (not strictly an invariant)

Dijkstra's Algorithm [Dijkstra 1982]



n processes with indices 0, 1, …, n-1

state of process j is $x[j] \in \{0, 1, 2, k\text{-}1\}$, where $k > n$

p_0 **if** $x[0] = x[N-1]$ **then** $x[0] := x[0] + 1 \bmod k$

p_j $j > 0$, **if** $x[j] \neq x[j-1]$ **then** $x[j] := x[j-1]$

(p_i **has TOKEN** if and only if the conditional is true)

A Specification Language

auto DijkstraTR (n:natural,k:natural)

type indices: [0,…,n-1]

type values: [0,…,k-1]

actions

  internal step(i:indices)

variables

  x:[indices->values] **initially** $\forall \boldsymbol{i} \in$ indices, $x[i] = 0$

transitions

  internal step(i:indices)

  pre $i = 0 \wedge x[i] = x[n-1]$

  eff $x[i] := x[i] + 1 \bmod k$;

  internal step(i:indices)

  pre $i \neq 0 \wedge x[i] \neq x[i-1]$

  eff $x[i] := x[i-1]$;

trajectories

Discrete Transition System or Automaton

An automaton is a tuple $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$ where

1. X is a set of names of variables; each variable $x \in X$ is associated with a type, $type(x)$
   - A valuation for X maps each variable in X to its type
   - Set of all valuations: $val(X) = Q$ this is sometimes identified as the state space of the automaton
2. $\Theta \subseteq val(X)$ is the set of initial or start states
3. A is a set of names of actions or labels
4. $\mathcal{D} \subseteq val(X) \times A \times val(X)$ is the set of transitions
   - a transition is a triple (u, a, u')
   - We write $(u, a, u') \in \mathcal{D}$ in short as $u \overset{a}{\rightarrow} u'$

HIOA Specs to Automata: variables

variables s, v: Reals; a: Bools

$X = \{s, v, a\}$

Example valuations also called **states**:

- $u_1 = \langle s \mapsto 0, v \mapsto 5.5, a \mapsto 0\rangle$
- $u_2 = \langle s \mapsto 10, v \mapsto -2.5, a \mapsto 1\rangle$

$val(X) = \{\langle s \mapsto c_1, v \mapsto c_2, a \mapsto c_3\rangle |\ c_1, c_2 \in R, c_3 \in \{0,1\}\}$

type indices: [0,…,n-1]

variables x: [indices->values]

- Fix n = 6, k = 8
- x: [{0,…,5} -> {0,…,7}]
- Example valuations:
  - $u = \langle x \mapsto \langle 0 \mapsto 0, 1 \mapsto 0, 2 \mapsto 0, 3 \mapsto 0, 4 \mapsto 0, 5 \mapsto 0 \rangle\rangle$
  - $v = \langle x \mapsto \langle 0 \mapsto 7, 1 \mapsto 0, 2 \mapsto 0, 3 \mapsto 0, 4 \mapsto 0, 5 \mapsto 0 \rangle\rangle$
  - Notation: $\boldsymbol{u}.x, \boldsymbol{u}.x[4] = 0$

$val(x) = \left\{\langle x \mapsto \langle i \mapsto c_i\rangle_{\{i=0\ldots5\}}\rangle \middle| c_i \in \{0, \ldots, 7\}\right\}$

States and predicates

A predicate over a set of variables X is a formula involving the variables in X. For example:

- ϕ_1: $\mathrm{x}[1] = 0$
- ϕ_2: $\forall i \in indices, x[i] = 0$

A valuation u satisfies predicate $\boldsymbol{\phi}$ if substituting the values of the variables in **u** in ϕ makes it evaluate to **True.** We write u$\vDash$ $\boldsymbol{\phi}$

- $\boldsymbol{u} \vDash \phi_1, \boldsymbol{u} \vDash \phi_2, \boldsymbol{v} \vDash \boldsymbol{\phi_1}$ and $\boldsymbol{v} \nvDash \boldsymbol{\phi_2}$

$[[\boldsymbol{\phi}]] = \{u \in val(x) |\, u \vDash \phi\}$. Examples

- $[[\phi_1]] = \{\langle x \mapsto \langle 1 \mapsto 0, i \mapsto c_i \rangle_{\{i=0,2,\ldots,5\}} \rangle \big| c_i \in \{0, \ldots, 7\}\}$
- $[[\phi_2]] = \{\langle x \mapsto \langle 0 \mapsto 0, 1 \mapsto 0, 2 \mapsto 0, 3 \mapsto 0, 4 \mapsto 0, 5 \mapsto 0 \rangle\rangle\}$

Initial state and invariant assertions

- $\Theta \subseteq val(x)$ initial states
  - Often specified by a predicate
  - $\boldsymbol{\phi_0} =$ (**Initially** $\forall \boldsymbol{i} \in$ indices, $x[i] = 0$)
  - $\Theta = [[\phi_0]] = \langle x \mapsto \langle i \mapsto 0 \rangle_{i=0,\ldots,5} \rangle$

- Invariant properties
  - “At least one process has the token”.
  - $\boldsymbol{I_1} = (x[0] = x[5] \vee \exists\, i \in \{1, \ldots 5\}: x[i] \neq x[i-1])$
  - $[[I_1]] = \{\langle 0, \ldots, 0 \rangle, \langle 1,0, \ldots, 0 \rangle, \ldots, \langle k-1, \ldots, k-1 \rangle\} = val(x)$ *(?)*
  - “Exactly one process has the token”
  - $\boldsymbol{I_2} = (x[0] = x[5] \ \oplus x[1] \neq x[0] \oplus x[2] \neq x[1] \ldots)$

Actions

- **actions** defines the set of Actions
- Examples
  - **internal** step(i:indices)
  - $A = \{step[0], \ldots, step[5]\}$
  - **internal** brakeOn, brakeOff
  - $A = \{brakeOn, brakeOff\}$

Transitions

$\mathcal{D} \subseteq val(X) \times A \times val(X)$ is the set of **transitions**

internal step(i:indices)

pre i = 0 $\bigwedge$ x[i] = x[n-1]

eff x[i] := x[i] + 1 mod k;

internal step(i:indices)

pre i $\neq$ 0 $\bigwedge$ x[i] $\neq$x[i-1]

eff x[i] := x[i-1];

$(u, a, u') \in \mathcal{D}$ iff $u \vDash Pre_a$ and $(u, u') \in Eff_a$

$(u, step(i), u') \in \mathcal{D}$ *iff*

(a) $(i = 0 \wedge u.x[0] = u.x[5]$

$\wedge u'.x[0] = u.x[0] + 1 \; mod \; 6)$ $\vee$

(b)$(i \neq 0 \wedge u.x[i] \neq u.x[i-1]$

$\wedge u'.x[i] = u.x[i-1])$

Nondeterminism

- For an action $a \in A$, Pre(a) is the formula defining its **pre**condition, and Eff(a) is the relation defining the **eff**ect.

- States satisfying precondition are said to enable the action

- In general Eff(a) could be a relation, but for this example it is a function

- **Nondeterminism**
  - Multiple actions may be enabled from the same state
  - There may be multiple post-states from the same action

Executions, Reachability, & Invariants

An execution of $\mathcal{A}$ is an alternating (possibly infinite) sequence of states and actions

$\alpha = u_0 a_1 u_1 a_2 u_3$ …such that:

- $u_0 \in \Theta$
- $\forall\ i$ in the sequence, $u_i \xrightarrow{a_{i+1}} u_{i+1}$

A state u is reachable if there exists an execution that ends at u. The set of reachable states is denoted by $Reach_A$.

Invariants (Formal)

What does it mean for I to hold "always" for $\mathcal{A}$?

- I holds at all states along any execution $u_0 a_1 u_1 a_2 u_3$
- I holds in all reachable states of $\mathcal{A}$
- $\boldsymbol{Reach}_{\mathcal{A}} \subseteq [[\boldsymbol{I}]]$

Invariants capture most properties that you will encounter in practice

- safety: "aircraft **always** maintain separation"
- bounded reaction time: "within 15 seconds of press, light must turn to walk"

How to **verify** if I is an invariant?

- Does there exist reachable state u such that $u \nvDash I$?

Reachability Problem

- Given a directed graph $G = (V, E)$, and two sets of vertices $S, T \subseteq V$, T is reachable from S if there is a path from S to T.

- Reachability Problem (G, S, T) : decide if T is reachable from S in G.

Algorithm for deciding Reachability G,S, T

```
Set Marked := {}
Queue Q := S
Marked := Marked ∪ S
while Q is not empty
        t ← Q.dequeue()
        if t ∈ T return “yes”
        for each (t,u) ∈ E
                if u ∉ Marked then
                Marked := Marked ∪ {u}
                Q := enqueue(Q, u)
return “no”
```

Verifying Invariants by solving Reachability

Given $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$ and a candidate invariant I, how to check that I is indeed an invariant of $\mathcal{A}$?

Define a graph $\mathrm{G} = \langle V, E \rangle$ where

$$V = val(X)$$

$$E = \{(u, u') | \exists\, a \in A, u \xrightarrow{a} u'\}$$

Claim. $[[I]]^c$ is not reachable from Θ in G iff I is an invariant of $\mathcal{A}$.

Summary so far

- Well-formed specifications define automata
- Invariants: Properties that hold at all reachable states. $\boldsymbol{Reach_{\mathcal{A}}} \subseteq [[\boldsymbol{I}]]$
- BFS to verify invariants automatically for (finite) automata

Temporal Logic and Model Checking

Verification thus far

Given an automaton $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$ and a set of unsafe states $U \subseteq val(X)$ we can check whether $Reach_{\mathcal{A}}(\Theta) \cap U = \emptyset$?

Thus, far we looked at verification of invariant properties through reachability analysis

What about more general types of properties, e.g.,

- “Eventually the light turns red and prior to that the orange light blinks”
- “After failures, eventually there is just one token in the system”

How to express and verify such properties?

Introduction to temporal logics

Temporal logics give a formal language for representing, and reasoning about, propositions qualified in terms of time, or their validity in a sequence

Amir Pnueli received the ACM Turing Award (1996) for seminal work introducing temporal logic into computer science and for outstanding contributions to program and systems verification.

Large follow-up literature, e.g., different temporal logics MTL, MITL, PCTL, ACTL, STL

Setup

We have a set of atomic propositions (AP)

These are the properties that hold in each state, e.g., “light is green”, “has 2 tokens”

We have a labeling function that assigns to each state, a set of propositions that hold at that state

$L: Q \to 2^{AP}$

Notations (this lecture)

$\mathcal{A} = \langle Q, Q_0, T, L \rangle, T \subseteq Q \times Q, L: Q \rightarrow 2^{AP}$

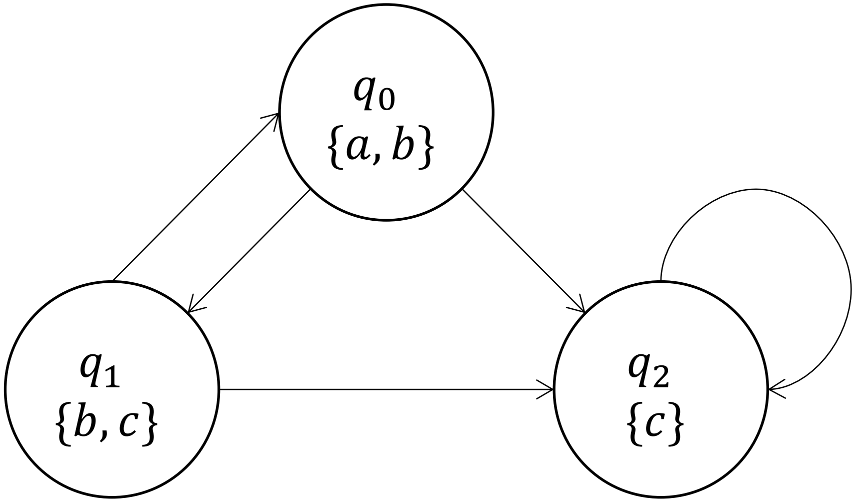
Executions $\alpha = q_0\ q_1 \ldots q_k = \alpha.lstate$

$\alpha[i] = q_i$

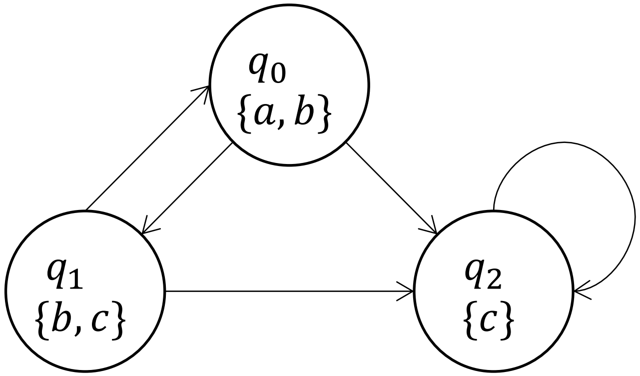
$Exec_{\mathcal{A}}$ set of all executions

$AP = \{a, b, c\}$

$L(q_0) = \{a, b\}$

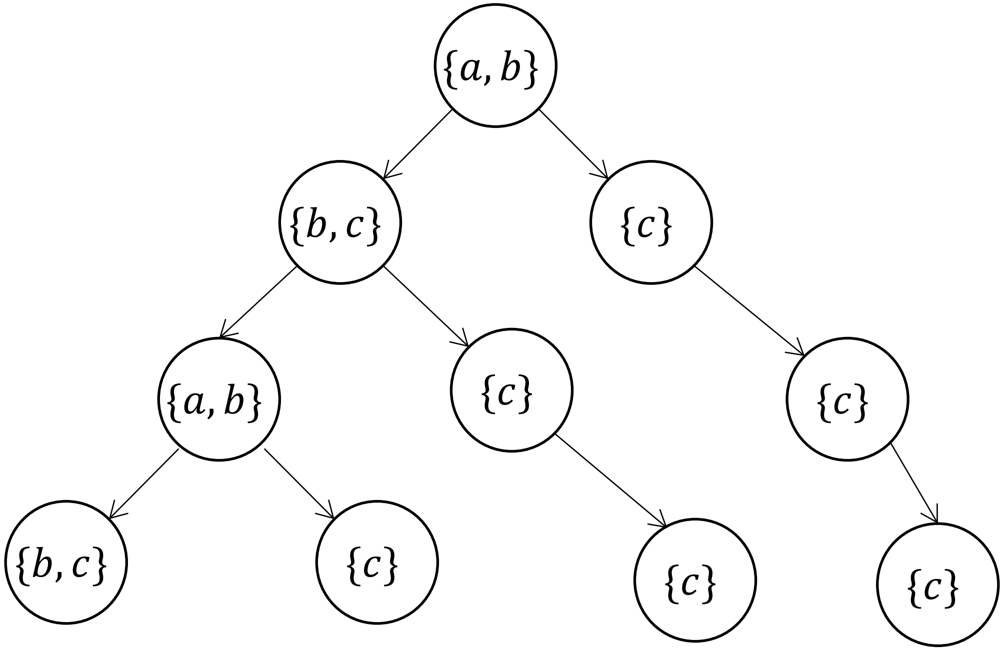


Computational tree logic (CTL)



Unfolding the automaton

We get a tree



A CTL formula allows us to specify subsets of paths in this tree

CTL quantifiers

Path quantifiers

- E: Exists some path
- A: All paths

Temporal operators

- X: Next state
- U: Until
- F: Eventually
- G: Globally (Always)

CTL syntax

CTL syntax

$$State\ Formula\ (SF) ::= true \mid p \mid \neg f_1 \mid f_1 \wedge f_2 \mid E\ \phi \mid A\ \phi$$

$$Path\ Formula\ (PF) ::= X f_1 \mid f_1 U\ f_2 \mid G f_1 \mid F\ f_1$$

where $p \in AP, f_1, f_2 \in SF, \phi \in PF$

Depth of formula: number of production rules used

Examples (depth)

E$X\ a$; AXE$X\ a$; AXEXa U b; AG AF green; AF AG single token

Depth 3, 5, …

Non-examples

AX$X\ a$; path and state operators must alternate in CTL

CTL semantics

Given automaton $\mathcal{A} = \langle Q, Q_0, T, L \rangle, q \in Q$ and a CTL formula $\phi, q \vDash \phi$ denotes that q satisfies ϕ; $\alpha \vDash \phi$ denotes that path (execution) α satisfies ϕ. The relation $\vDash$ is defined inductively as:

$$
\begin{array}{ll}
\mathcal{A}, q \vDash p & \Leftrightarrow p \in L(q) \text{ for } p \in AP \\
\mathcal{A}, q \vDash \neg f_1 & \Leftrightarrow \mathcal{A}, q \nvDash f_1 \\
\mathcal{A}, q \vDash f_1 \wedge f_2 & \Leftrightarrow \mathcal{A}, q \vDash f_1 \wedge \mathcal{A}, q \vDash f_2 \\
\mathcal{A}, q \vDash E\phi & \Leftrightarrow \exists\, \alpha, \alpha.fstate = q, \mathcal{A}, \alpha \vDash \phi \\
\mathcal{A}, q \vDash A\phi & \Leftrightarrow \forall\, \alpha, \alpha.fstate = q, \mathcal{A}, \alpha \vDash \phi \\
\mathcal{A}, \alpha \vDash Xf & \Leftrightarrow \mathcal{A}, \alpha[1] \vDash f \\
\mathcal{A}, \alpha \vDash f_1\, U\, f_2 & \Leftrightarrow \exists i \geq 0, \mathcal{A}, \alpha[i] \vDash f_2 \textit{ and } \forall j < i\ \alpha[j] \vDash f_1 \\
\mathcal{A}, \alpha \vDash F\, f_1 & \Leftrightarrow \exists i \geq 0, \mathcal{A}, \alpha[i] \vDash f_1 \\
\mathcal{A}, \alpha \vDash G\, f_1 & \Leftrightarrow \forall i \geq 0, \mathcal{A}, \alpha[i] \vDash f_1
\end{array}
$$

Automaton satisfies property: $\mathcal{A} \vDash f$ iff $\forall q \in Q_0, \mathcal{A}, q \vDash f$

Back to CTL: Universal CTL operators

X, U, G can be used to derive other operators

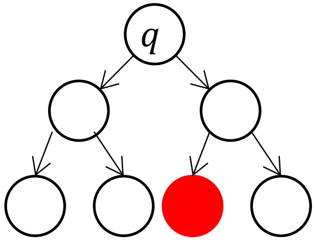
$true\ U\ f \equiv F\ f$

$Gf \equiv \neg F(\neg f)$

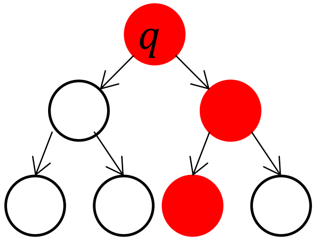
All ten combinations can be expressed using EX, EU, EG

AXf	AGf	AFf	AUf	ARf
$\neg EX(\neg f)$	$\neg EF(\neg f)$	$\neg EG(\neg f)$		
EX	EG	EF	EU	ER
EX	EG	$E(true\ U\ f)$	EU	

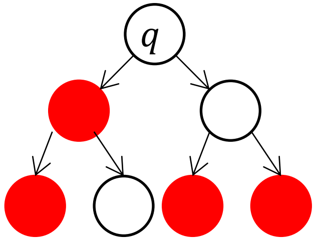
Visualizing semantics



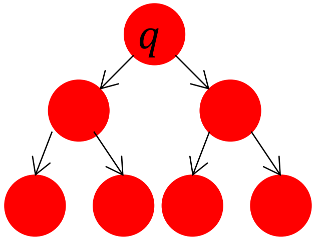
$q \vDash EF\ red$



$q \vDash EG\ red$



$q \vDash AF\ red$



$q \vDash AG\ red$

Exercise

- How are CTL properties related to Lyapunov stability?
- Asymptotic stability?

Algorithm for deciding $\mathcal{A} \models f$

Algorithm works by structural induction on the depth of the formula

Explicit state model checking

Compute the subset $Q' \subseteq Q$ such that $\forall q \in Q'\ we\ have\ \mathcal{A}, \mathrm{q} \models f$

If $Q_0 \subseteq Q'$ $\cancel{Q' \subseteq Q_0}$ then we can conclude $\mathcal{A} \models f$

Induction on depth of formula

Algorithm computes a function $label: Q \rightarrow CTL(AP)$ that labels each state with a CTL formula

- Initially, $label(q) = L(q)$ for each $q \in \mathrm{Q}$
- At i^{th} iteration $label(q)$ contains all sub-formulas of f of depth $(i-1)$ that q satisfies

At termination $f \in label(q) \Leftrightarrow \mathcal{A}, q \vDash f$

Structural induction on formula

Six cases to consider based on structure of f

$f = p,$	for some $p \in AP, \forall q, label(q) \coloneqq label(q) \cup f$
$f = \neg f_1$	if $f_1 \notin label(q)$ then $label(q) \coloneqq label(q) \cup f$
$f = f_1 \wedge f_2$	if $f_1, f_2 \in label(q)$ then $label(q) \coloneqq label(q) \cup f$
$f = EXf_1$	if $\exists q' \in$ Q such that $(q, q') \in T$ and $f_1 \in label(q')$ then $label(q) \coloneqq label(q) \cup f$
$f = E[f_1 U f_2]$	$CheckEU(f_1, f_2, Q, T, L)$ [next slide]
$f = EGf_1$	$CheckEG(f_1, Q, T, L)$ [next slide]

$CheckEU(f_1, f_2, Q, T, L)$

Let $S = \{q \in Q \mid f_2 \in label(q)\}$
for each $q \in S$
  $label(q) := label(q) \cup \{E[f_1 U f_2]\}$
while $S \neq \emptyset$
  for each $q' \in S$
    $S := S \setminus \{q'\}$
    for each $q \in T^{-1}(q')$
      if $f_1 \in label(q)$ then
        $label(q) := label(q) \cup \{E[f_1 U f_2]\}$
        $S := S \cup \{q\}$

Proposition. For any state $label(q) \ni E[f_1 U f_2]$ iff $q \vDash E[f_1 U f_2]$.

Proposition. Finite Q therefore terminates and in $O(|Q| + |T|)$ steps.

$CheckEG(f_1, Q, T, L)$

From $\mathcal{A}$ we construct a new automaton $\mathcal{A}' = \langle Q', T', L' \rangle$ such that

$Q' = \{q \in Q \mid f_1 \in label(q)\}$

$T' = \{\langle q_1, q_2 \rangle \in T \mid q_1 \in Q'\} = T \mid Q'$ // T restricted to Q'

$L': Q' \rightarrow 2^{AP}\ \forall\, q' \in Q', L'(q') := L(q')$ // L restricted to Q'

Claim. $\mathcal{A}$, q $\vDash EG f_1$ iff in $\mathcal{A}'$

(1) $q \in Q'$

(2) $\exists \alpha \in Execs_{\mathcal{A}}$, with $\alpha.fstate = q$ and $\alpha.lstate$ is in a nontrivial strongly connected component (SCC) C of the graph $\langle Q', T' \rangle$

Claim. $\mathcal{A}, \mathrm{q} \vDash EGf_1$ iff

(1) $q \in Q'$ and

(2) $\exists \alpha \in Execs_{\mathcal{A}}$, with $\alpha.fstate = q$ and $\alpha.lstate$ is in a nontrivial SCC C of the graph $\langle Q', T' \rangle$

Proof. Suppose $\mathcal{A}, \mathrm{q} \vDash EGf_1$

Consider any execution α with $\alpha.fstate = q$. Obviously, $q \vDash f_1$ and so, $q \in Q'$.

Since Q is finite α can be written as $\alpha = \alpha_0\alpha_1$ where α_0 is finite and every state in α_1 repeats infinitely many times.

Let C be the states in α_1. $C \in Q'$.

Consider any two q_1 and q_2 states in C, we observe that $q_1 \rightleftarrows q_2$, and therefore C is a SCC.

Consider (1) and (2). We will construct a path $\alpha = \alpha_0\alpha_1$ such that $\alpha_0.fstate = q$ and $\alpha_0 \in Q'$and $\alpha_1$visits some states infinitely often.

$CheckEG(f_1, Q, T, L)$

Let $Q' = \{q \in Q \mid f_1 \in label(q)\}$

Let $\mathbb{C}$ be the set of nontrivial SCCs of $\langle Q', T' \rangle$

$\boldsymbol{T} = \cup_{C \in \mathbb{C}} \{q \mid q \in C\}$

for each $q \in \boldsymbol{T}$

  $label(q) := label(q) \cup \{EG f_1\}$

while $\boldsymbol{T} \neq \emptyset$

  for each $q' \in \boldsymbol{T}$

    $\boldsymbol{T} := \boldsymbol{T} \setminus \{q'\}$

    for each $q' \in Q'$ such that $(q', q) \in T'$

      if $\mathrm{EG} f_1 \notin label(q')$ then

        $label(q') := label(q') \cup \{EG f_1\}$

        $\boldsymbol{T} := \boldsymbol{T} \cup \{q\}$

Proposition. Finite Q therefore terminates and in $O(|Q| + |T|)$ steps.

Proposition. For any state $label(q) \ni EG f_1$ iff $q \vDash EG f_1$.

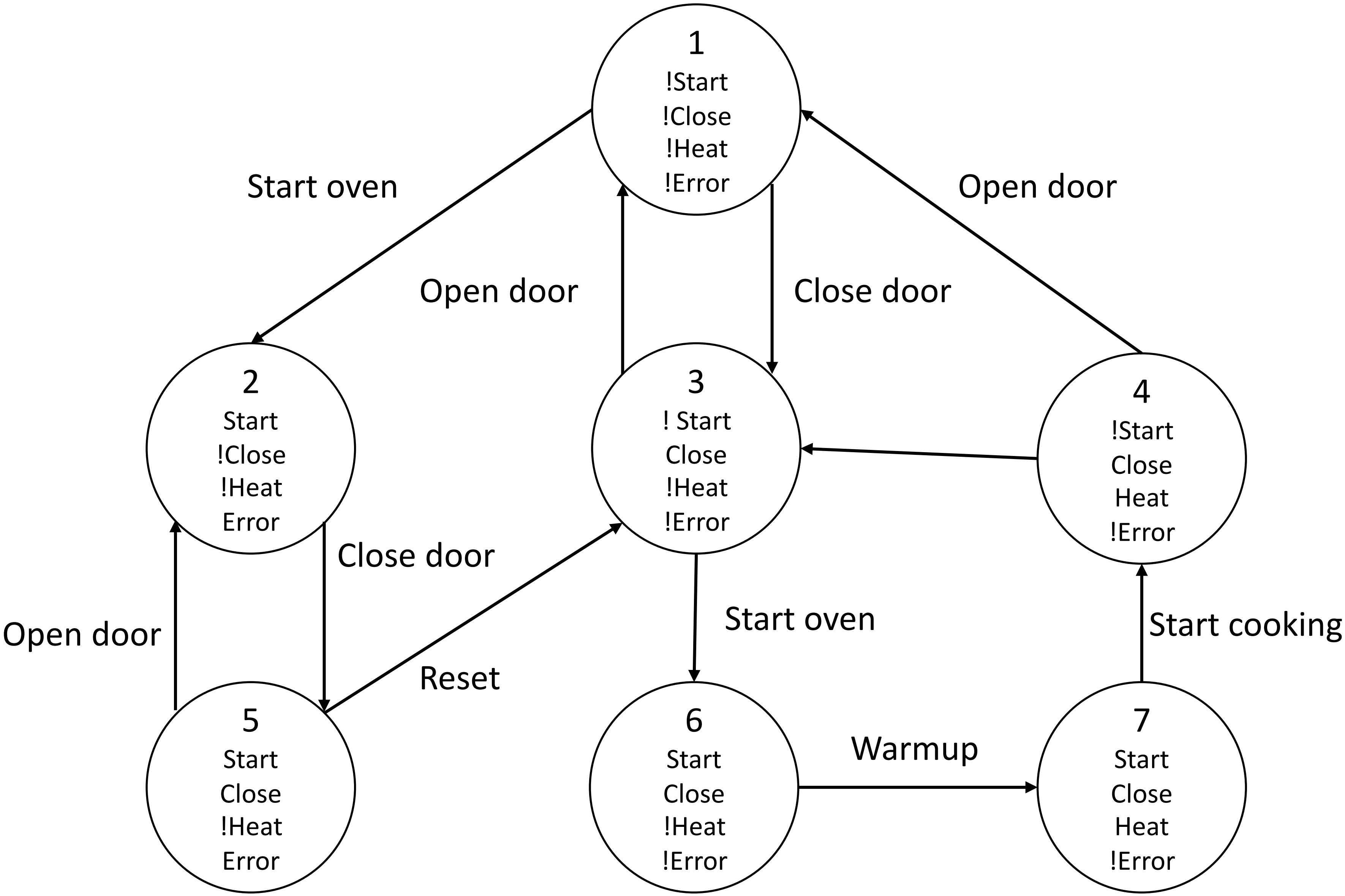
Putting it all together

Explicit model checking algorithm input $\mathcal{A} \vDash f$?

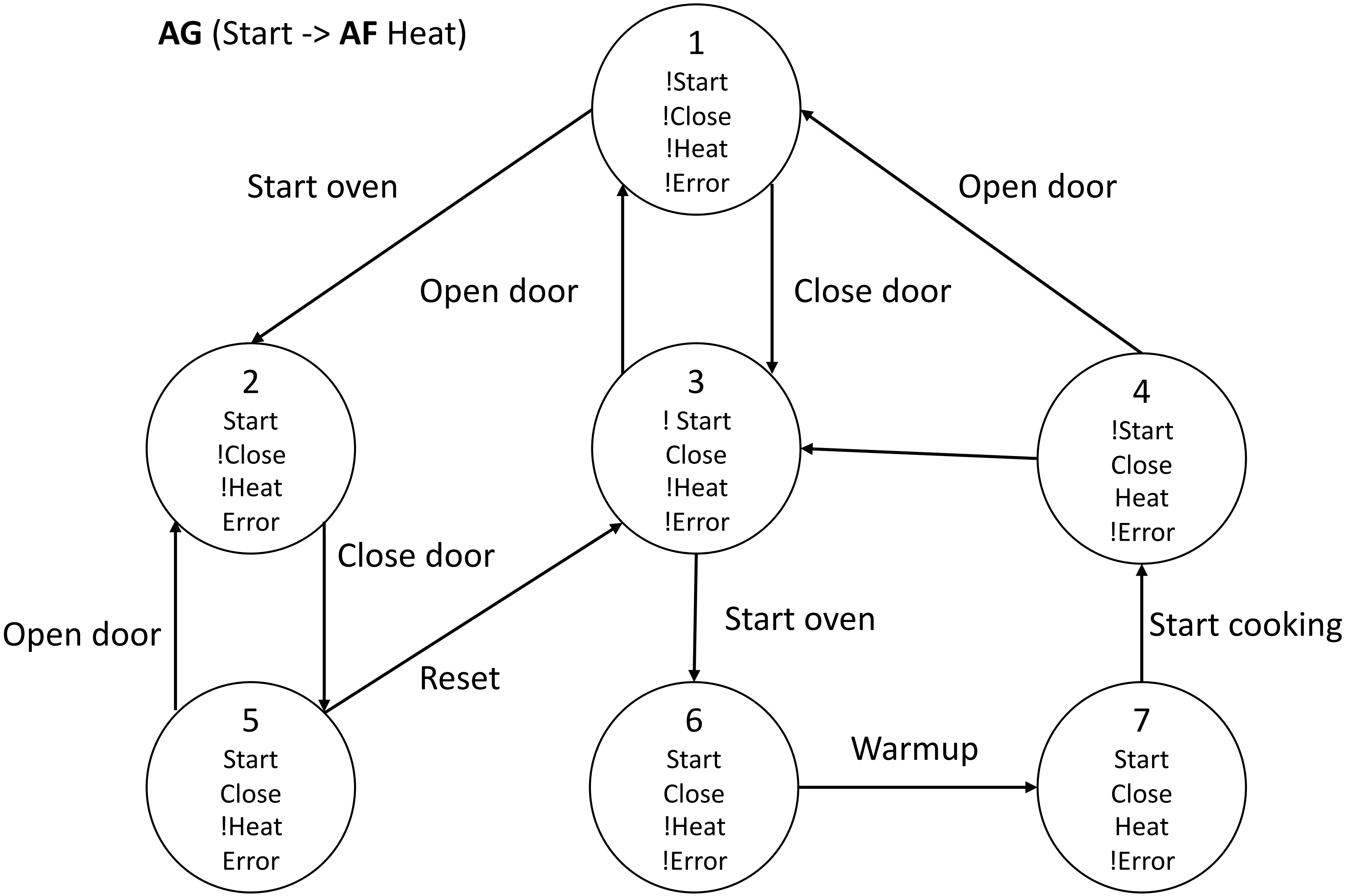
Structural induction over CTL formula

$f = p,$	for some $p \in AP, \forall q, label(q) := label(q) \cup \{p\}$
$f = \neg f_1$	if $f_1 \notin label(q)$ then $label(q) := label(q) \cup f$
$f = f_1 \wedge f_2$	if $f_1, f_2 \in label(q)$ then $label(q) := label(q) \cup f$
$f = EX f_1$	if $\exists q' \in$ Q such that $(q, q') \in T$ and $f_1 \in label(q')$ then $label(q) := label(q) \cup f$
$f = E[f_1 U f_2]$	$CheckEU(f_1, f_2, Q, T, L)$
$f = EG f_1$	$CheckEG(f_1, Q, T, L)$

Proposition. Overall complexity of CTL model checkign $O(|f|(|Q| + |T|))$ steps.

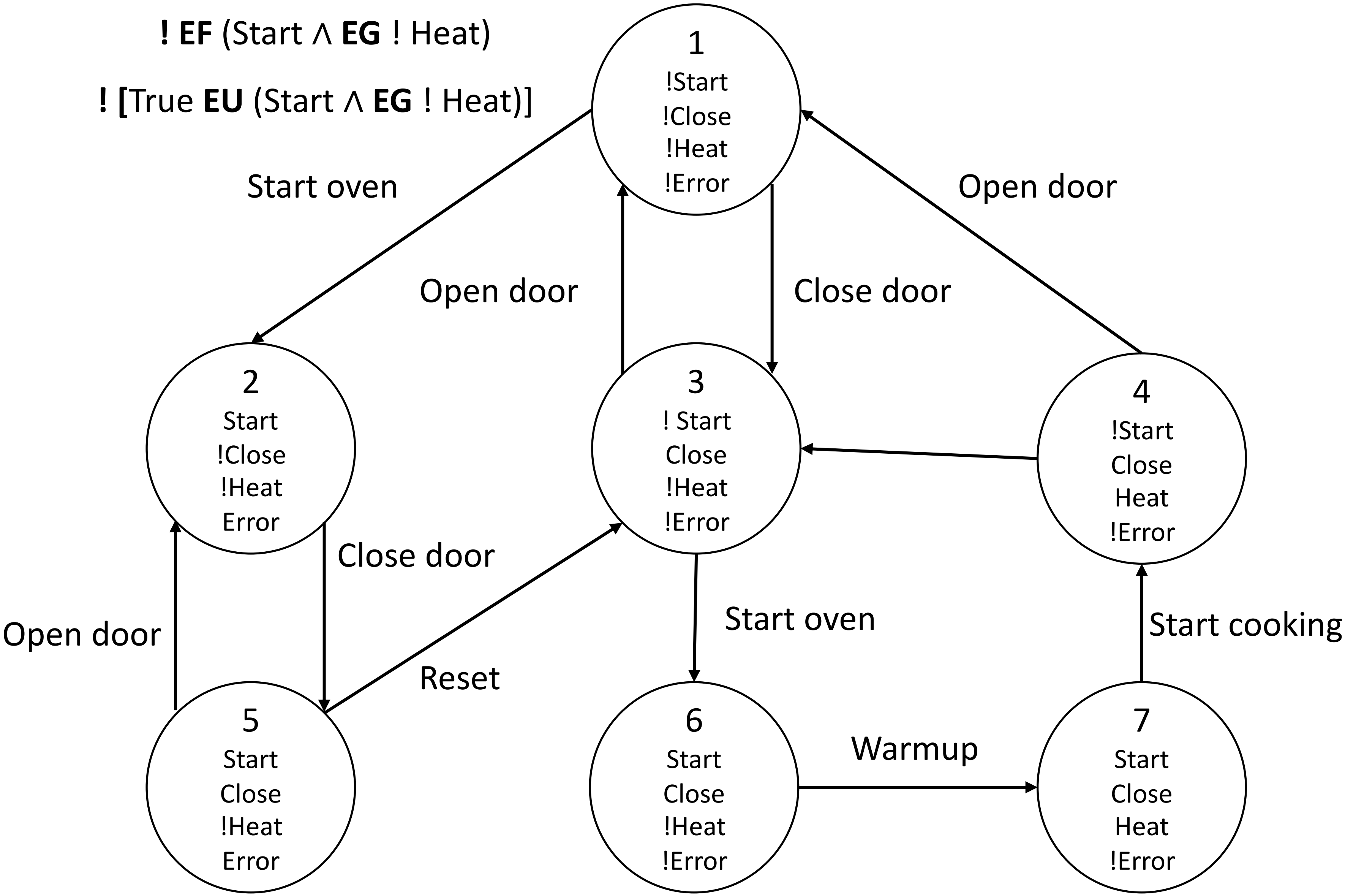

1
!Start
!Close
!Heat
!Error
Start oven
Open door
Open door
Close door
2
Start
!Close
!Heat
Error
3
! Start
Close
!Heat
!Error
4
!Start
Close
Heat
!Error
Close door
Open door
Reset
Start oven
Start cooking
5
Start
Close
!Heat
Error
6
Start
Close
!Heat
!Error
Warmup
7
Start
Close
Heat
!Error

AG (Start -> **AF** Heat)



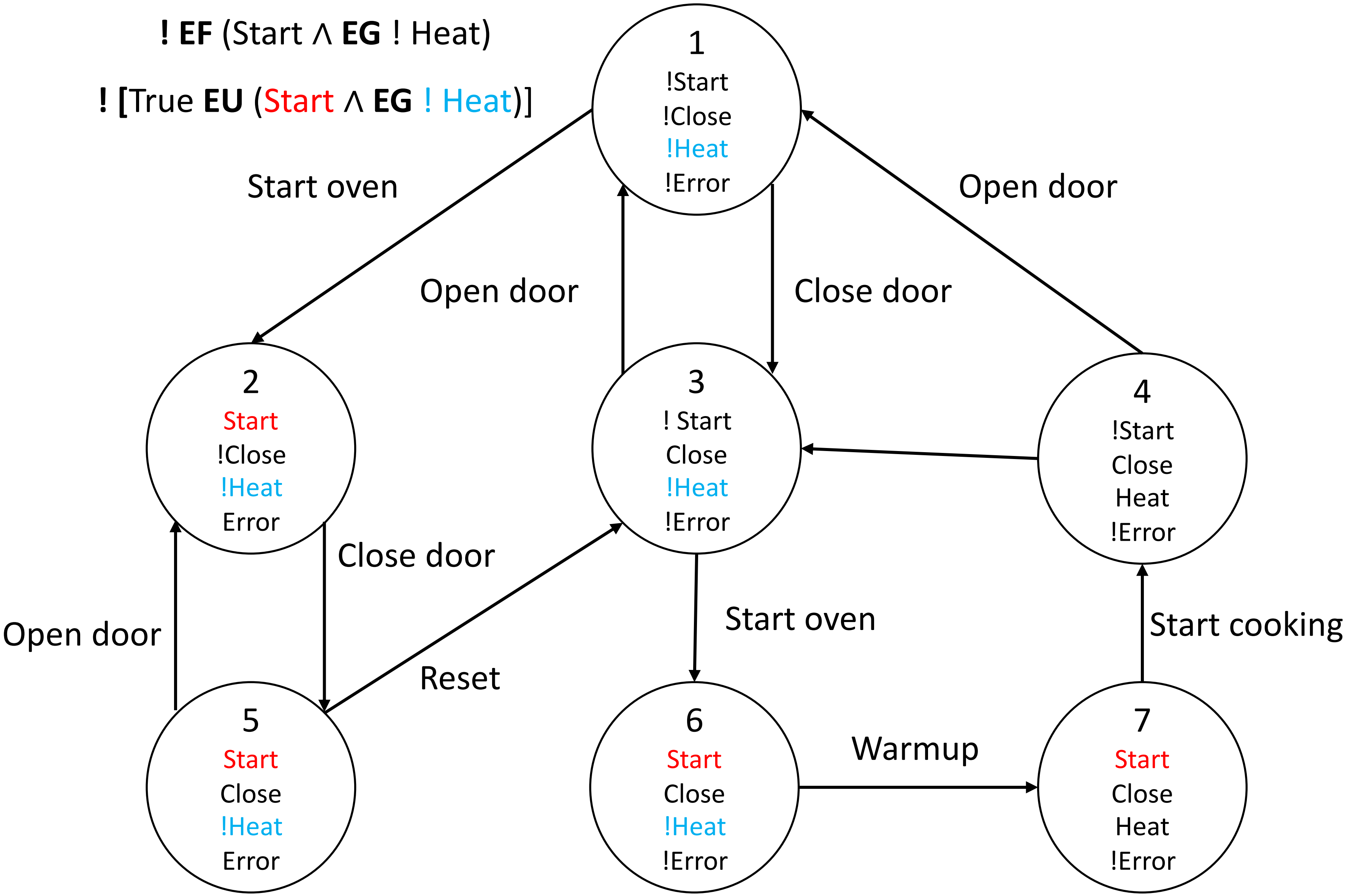
! EF (Start ∧ **EG** ! Heat)

! [True **EU** (Start ∧ **EG** ! Heat)]



! EF (Start ∧ **EG** ! Heat)

! [True **EU** (Start ∧ **EG** ! Heat)]

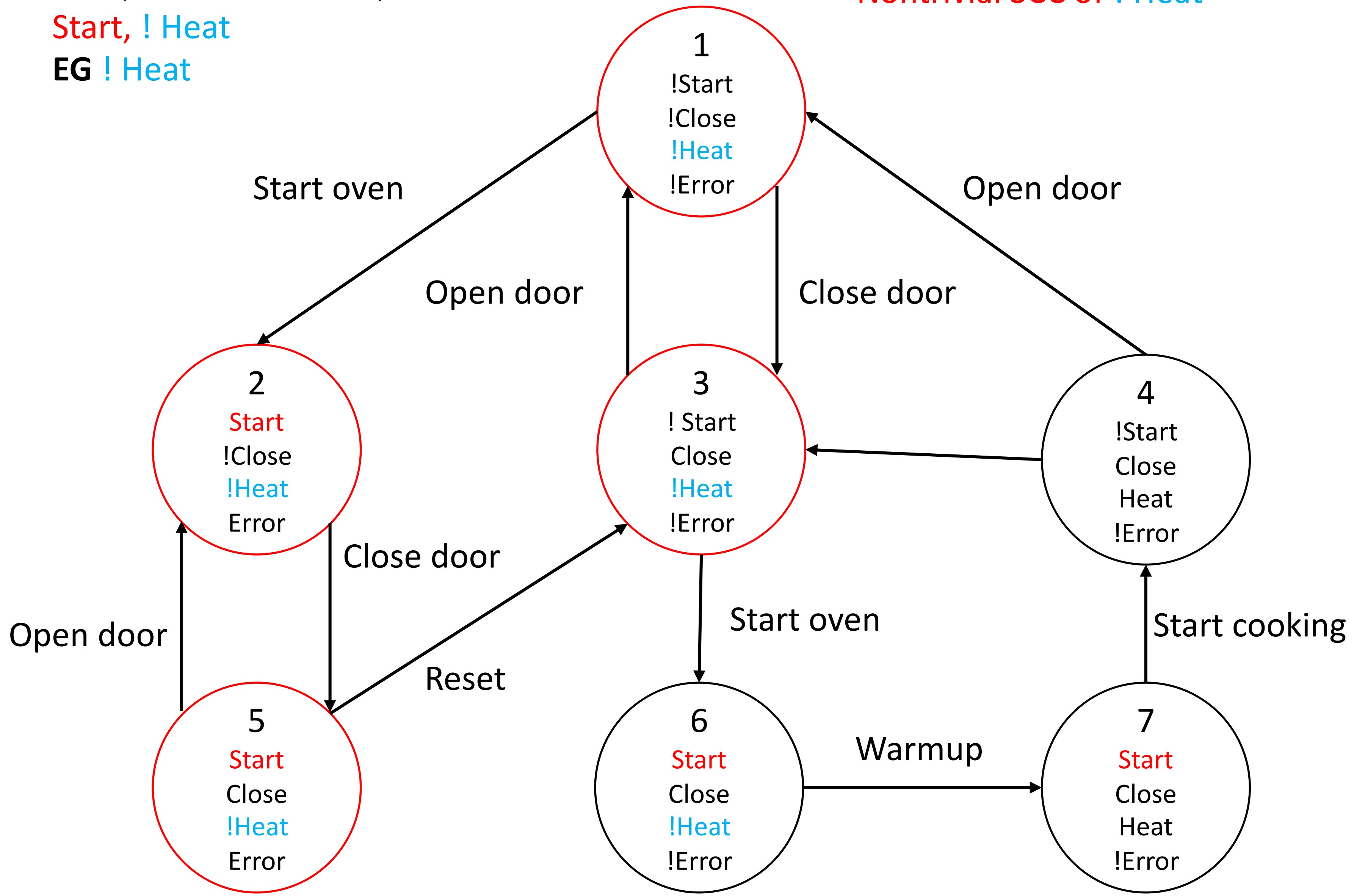


! EF (Start ∧ **EG** ! Heat)

Start, ! Heat

EG ! Heat

Nontrivial SCC of ! Heat



! **EF** (Start ∧ **EG** ! Heat)

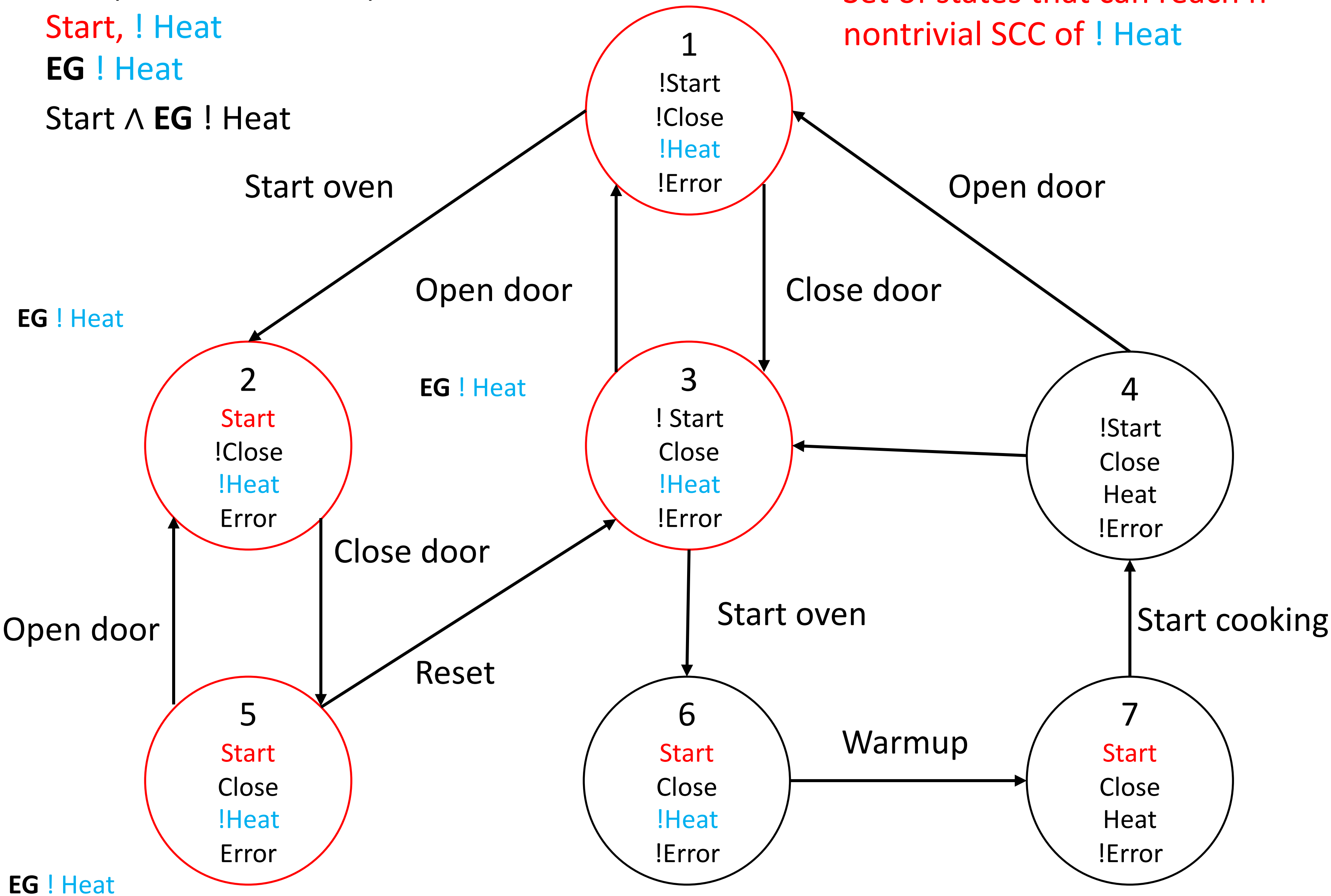
EG ! Heat

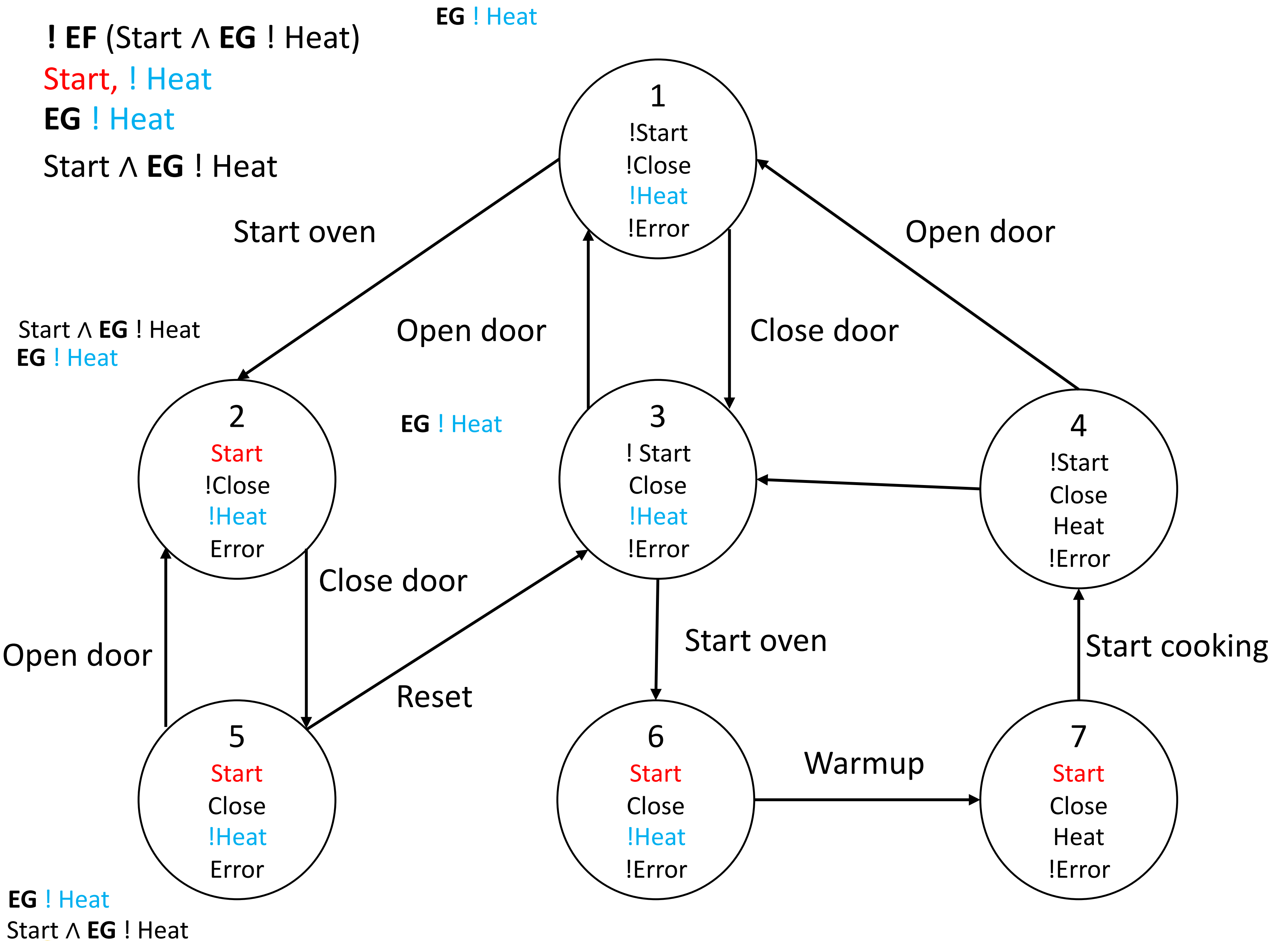
Start, ! Heat

EG ! Heat

Start ∧ **EG** ! Heat

Set of states that can reach n nontrivial SCC of ! Heat





! EF (Start ∧ EG ! Heat)
Start, ! Heat
EG ! Heat
Start ∧ EG ! Heat
EG ! Heat
1
!Start
!Close
!Heat
!Error
Start oven
Open door
Close door
Open door
Start ∧ EG ! Heat
EG ! Heat
2
Start
!Close
!Heat
Error
EG ! Heat
3
! Start
Close
!Heat
!Error
4
!Start
Close
Heat
!Error
Close door
Start oven
Start cooking
Open door
Reset
5
Start
Close
!Heat
Error
6
Start
Close
!Heat
!Error
Warmup
7
Start
Close
Heat
!Error
EG ! Heat
Start ∧ EG ! Heat

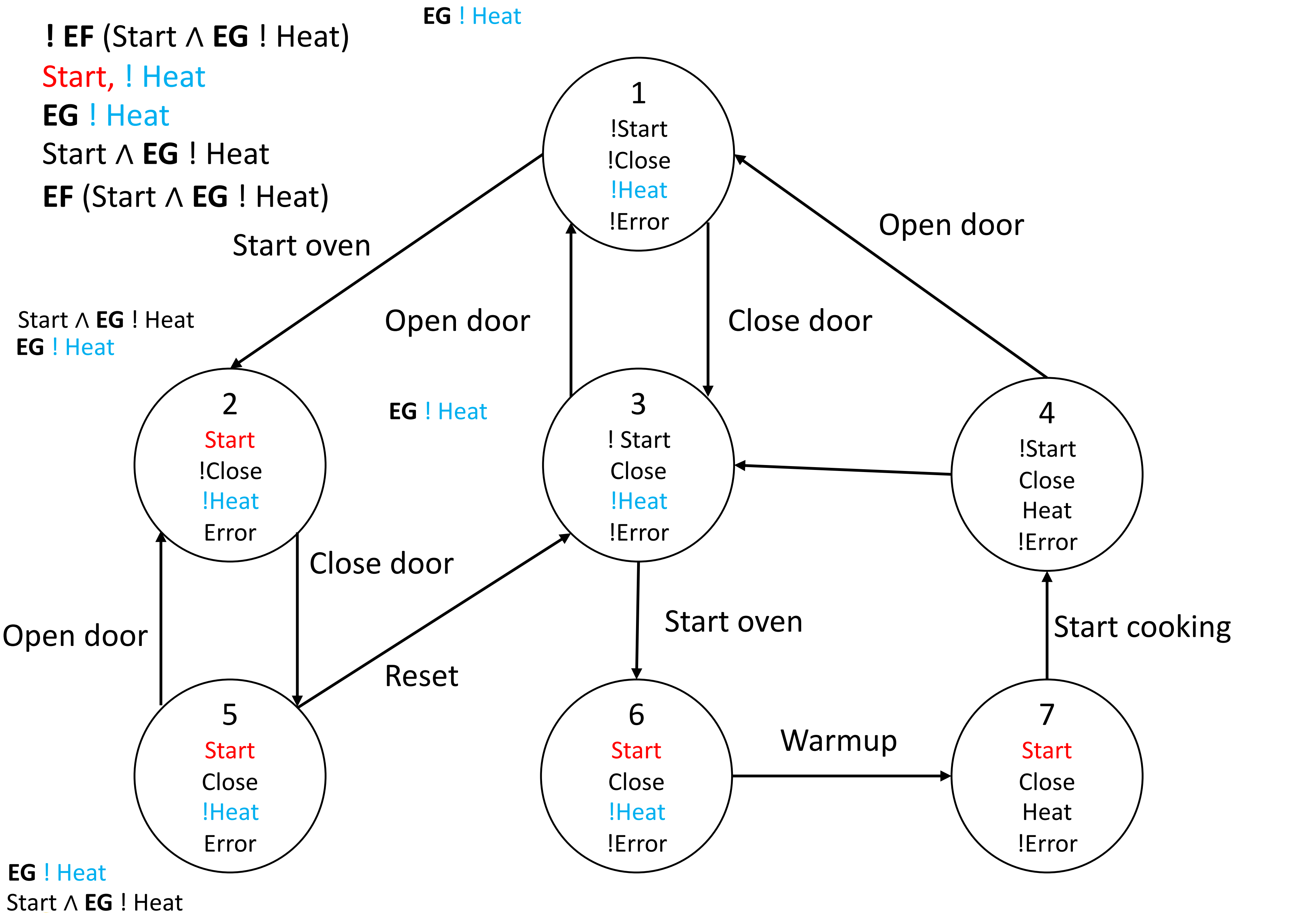
! EF (Start ∧ **EG** ! Heat)

Start, ! Heat

EG ! Heat

Start ∧ **EG** ! Heat

EF (Start ∧ **EG** ! Heat)



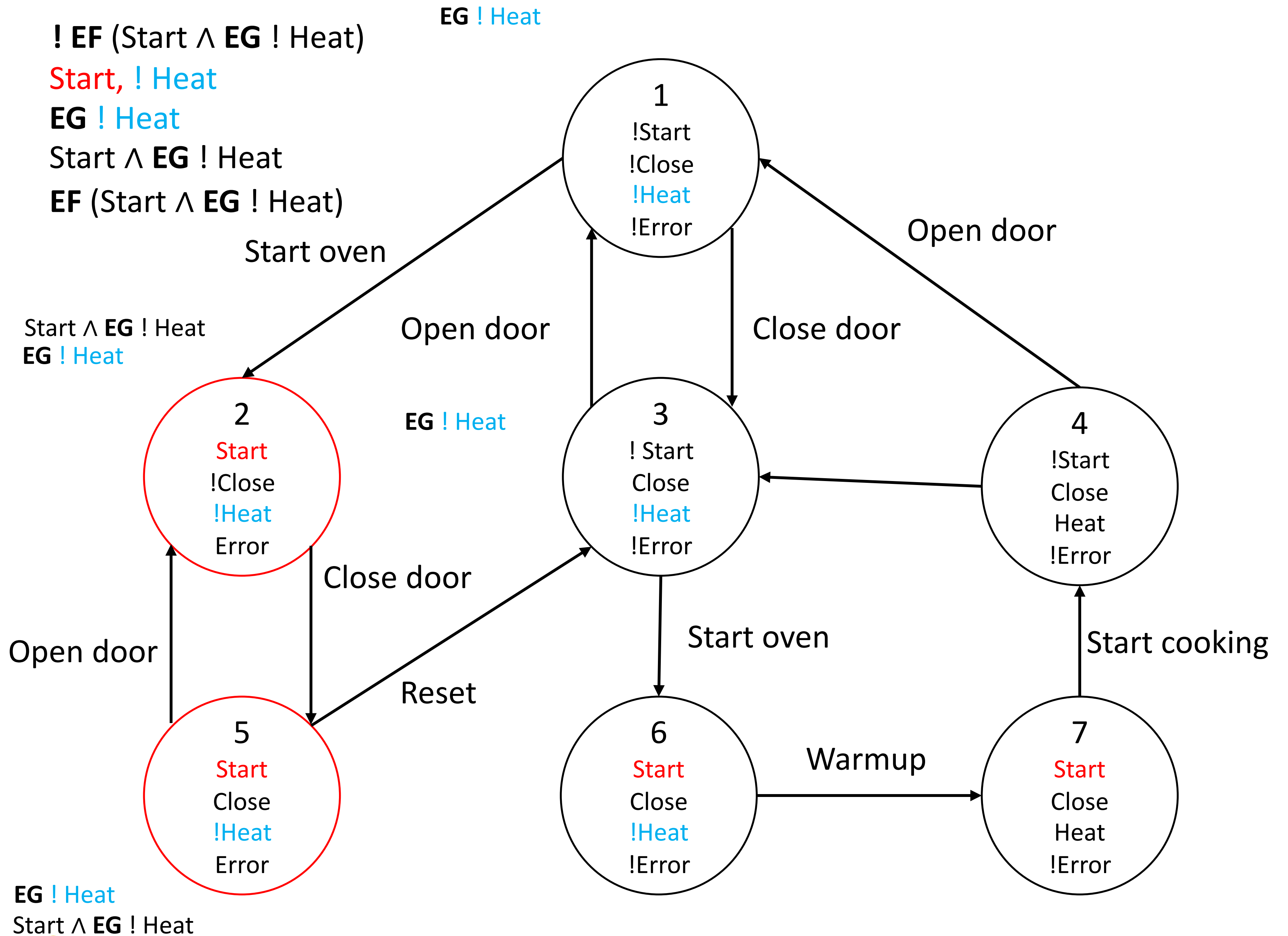
! **EF** (Start ∧ **EG** ! Heat)

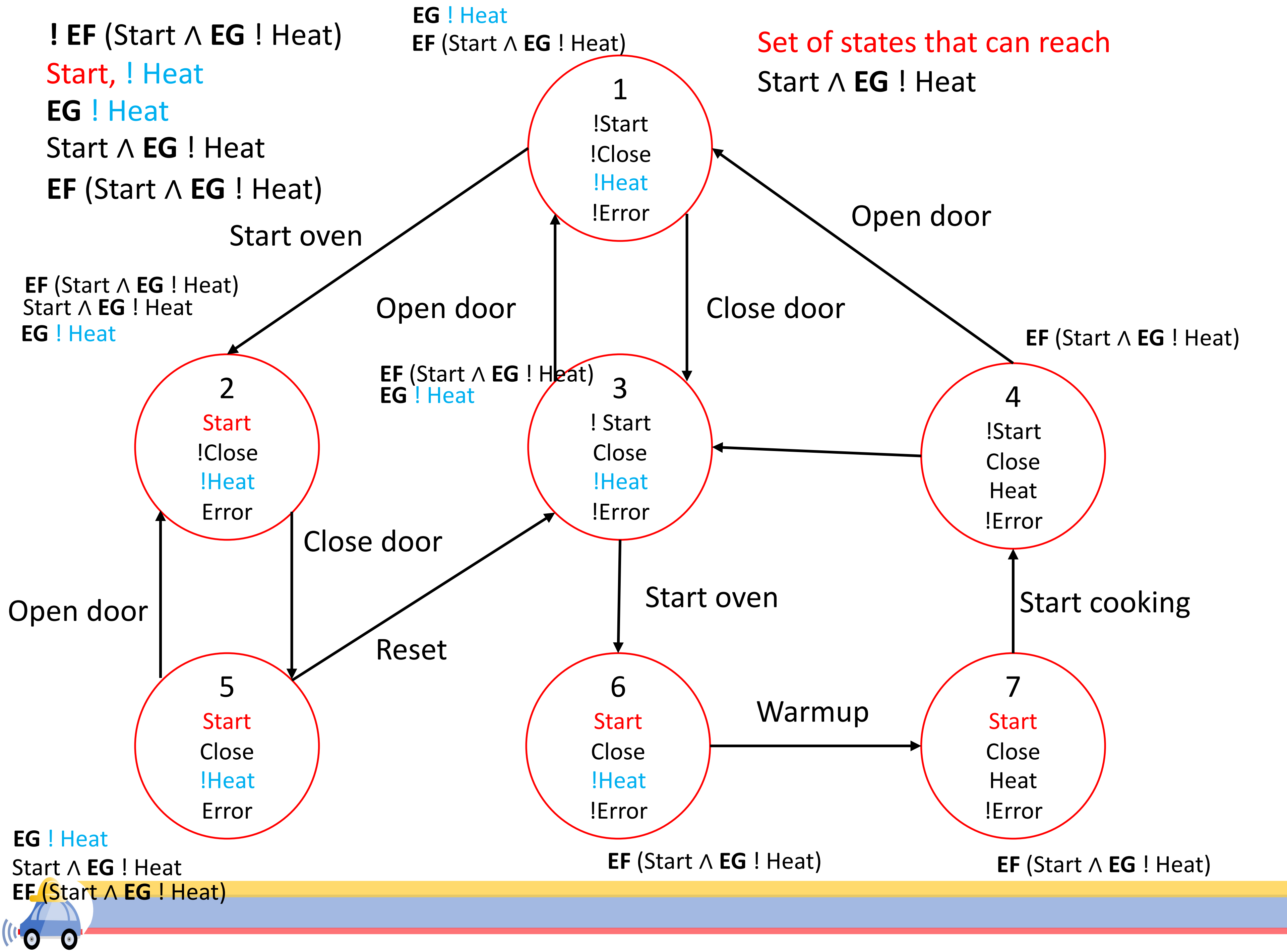
Start, ! Heat

EG ! Heat

Start ∧ **EG** ! Heat

EF (Start ∧ **EG** ! Heat)




! EF (Start ∧ EG ! Heat)
Start, ! Heat
EG ! Heat
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)
EG ! Heat
EF (Start ∧ EG ! Heat)
Set of states that can reach
Start ∧ EG ! Heat
1
!Start
!Close
!Heat
!Error
Start oven
Open door
Close door
Open door
EF (Start ∧ EG ! Heat)
Start ∧ EG ! Heat
EG ! Heat
EF (Start ∧ EG ! Heat)
EG ! Heat
EF (Start ∧ EG ! Heat)
2
Start
!Close
!Heat
Error
3
! Start
Close
!Heat
!Error
4
!Start
Close
Heat
!Error
Close door
Start oven
Start cooking
Open door
Reset
5
Start
Close
!Heat
Error
6
Start
Close
!Heat
!Error
Warmup
7
Start
Close
Heat
!Error
EG ! Heat
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)
EF (Start ∧ EG ! Heat)
EF (Start ∧ EG ! Heat)

! EF (Start ∧ EG ! Heat)

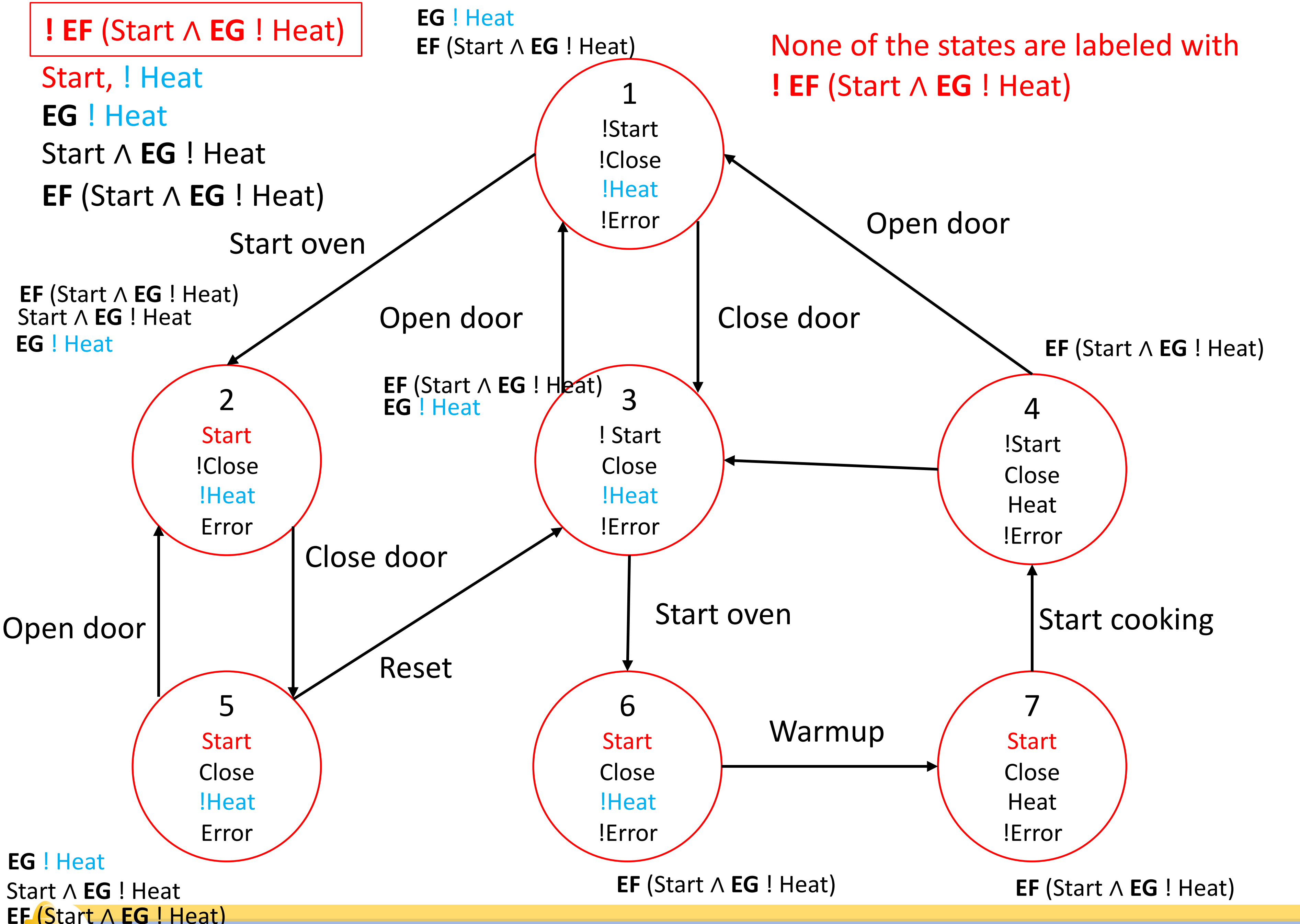
Start, ! Heat

EG ! Heat

Start ∧ **EG** ! Heat

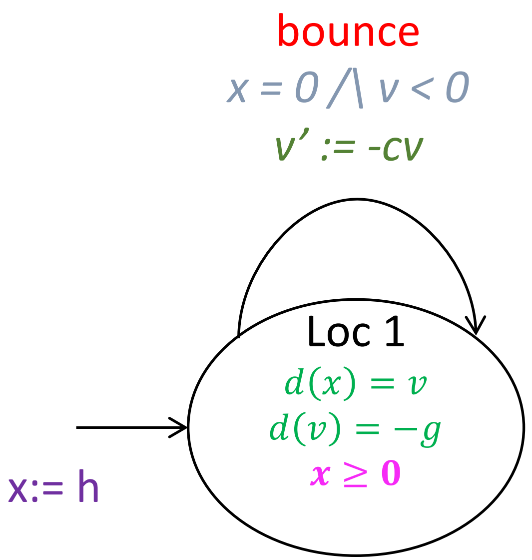
EF (Start ∧ **EG** ! Heat)

None of the states are labeled with **! EF (Start ∧ EG ! Heat)**



Timed and hybrid models

Bouncing Ball



Automaton Bouncingball(c,h,g)

variables: analog x: Reals := h, v: Reals := 0

states: True

actions: external bounce

transitions:

bounce

pre $x = 0 \wedge v < 0$

eff v := -cv

trajectories:

evolve d(x) = v; d(v) = -g

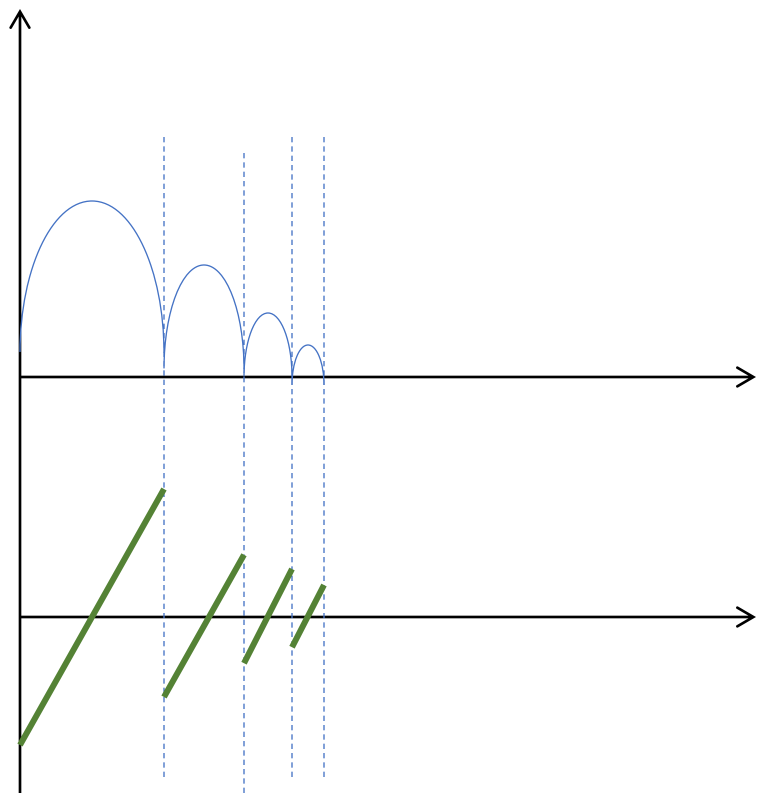
invariant $x \geq 0$

Graphical Representation used in many articles

TIOA Specification Language (close to PHAVer & UPPAAL's language)

Semantics: Executions and Traces

- An execution fragment of $\mathcal{A}$ is an (possibly infinite) alternating (A, X)-sequence $\alpha = \tau_0\ a_1\ \tau_1 a_2 \tau_2\ \ldots$ where
  - $\forall$ i $\tau_i.lstate \xrightarrow{a_{i+1}} \tau_{i+1}.fstate$
- If τ_0.fstate $\in \Theta$ then its an execution
- **Execs$_{\mathcal{A}}$** set of all executions

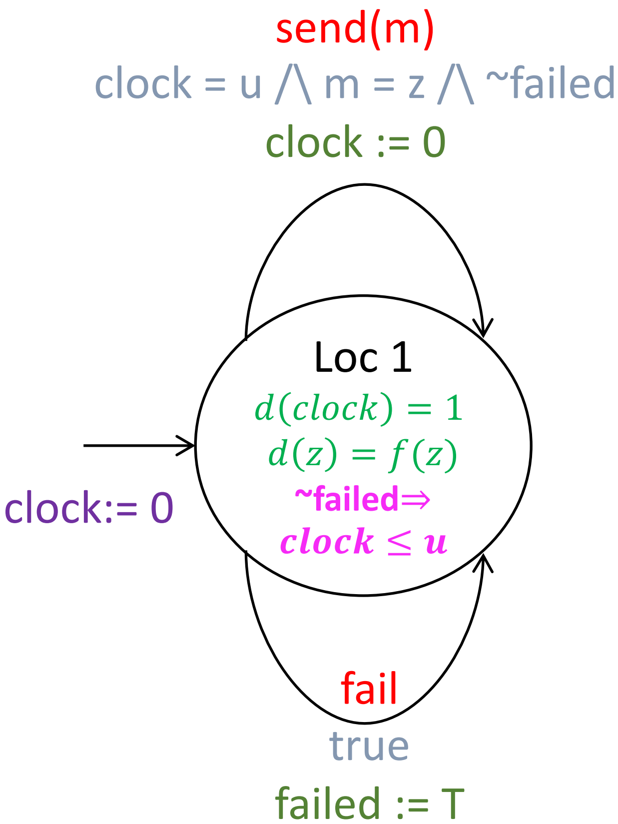


- The trace of an execution: external part of the execution. Alternating sequence of **external** actions and trajectories of the **empty set** of variables

Special kinds of executions

- Infinite: Infinite sequence of transitions and trajectories
- Closed: Finite with final trajectory with closed domain
- Admissable: Infinite duration
  - May or may not be infinite
- Zeno: Infinite but not admissable
  - Infinite number of transitions in finite time

Another Example: Periodically Sending Process



Automaton PeriodicSend(u)

variables: analog

clock: Reals := 0, z:Reals, failed:Boolean := F

actions: external send(m:Reals), fail

transitions:

send(m)

pre clock = u /\ m = z /\ ~failed

eff clock := 0

fail

pre true

eff failed := T

trajectories:

evolve d(clock) = 1, d(z) = f(z)

stop when ~failed /\ clock=u

Special Classes of Hybrid Automata

- Timed Automata ←
- Rectangular Initialized HA
- Rectangular HA
- Linear HA
- Nonlinear HA

Clocks and Clock Constraints
[Alur and Dill 1991]

- A clock variable x is a continuous (analog) variable of type real such that along any trajectory τ of x, for all t ∈ $\tau.dom, \tau(t)[x = t$.

- That is, $\dot{x} = 1$

- For a set X of clock variables, the set Φ(X) of integral clock constraints are expressions defined by the syntax:

  g ::= x $\leq q \mid x \geq q \mid \neg\, g \;\mid g_1 \wedge \;\; g_2$

  where $x \in X \; and \; q \in \; \mathbb{Z}$

- Examples: x = 10; x ∈ [2, 5); true are valid clock constraints
- Semantics of clock constraints $[g]$

Integral Timed Automata [Alur and Dill 1991]

Definition. A integral timed automaton is a HIOA $\boldsymbol{\mathcal{A}}$ = $\langle V, Q, \Theta, A, \mathcal{D}, \mathcal{T} \rangle$ where

- V = X ∪ $\{l\}$, where X is a set of n clocks and l is a discrete state variable of finite type Ł
- A is a finite set of actions
- $\mathcal{D}$ is a set of transitions such that
  - The guards are described by clock constrings $\Phi(X)$
  - $\langle x, l \rangle - a \rightarrow \langle x', l' \rangle$ implies either $x' = x$ or $x = 0$
- $\mathcal{T}$ set of clock trajectories for the clock variables in X

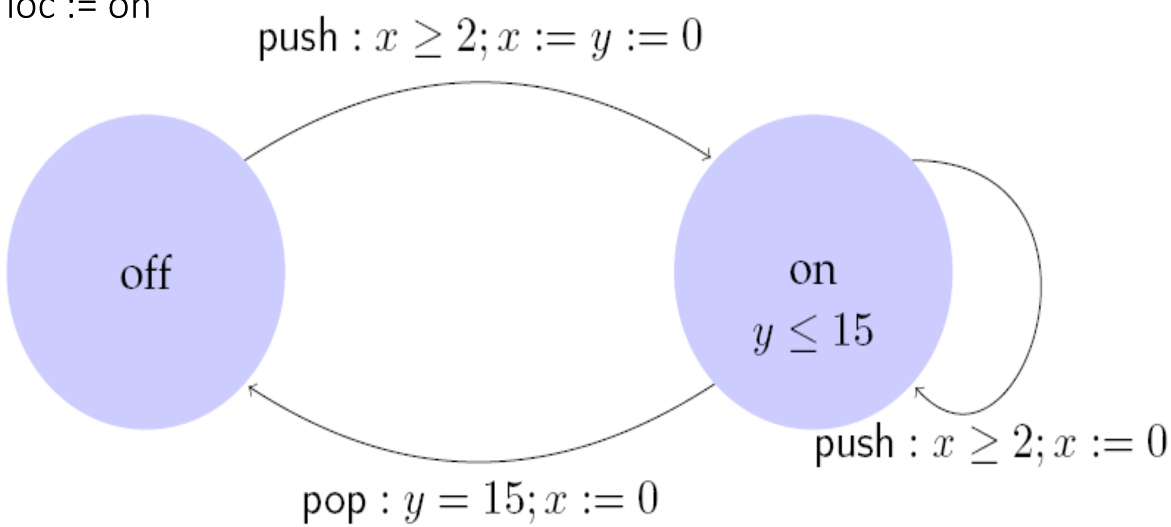
Example: Light switch

automaton Switch
variables
internal x, y:Real := 0, loc: {on,off} := off

transitions
internal push
pre x $\geq$ 2
eff if loc = off **then** y := 0 **fi;** x := 0; loc := on
internal pop
pre y = 15 $\bigwedge$ loc = off
eff x := 0

trajectories
invariant loc = on $\bigvee$ loc = off
stop when y = 15 $\bigwedge$ loc = off
evolve d(x) = 1; d(y) = 1

Description
Switch can be turned on whenever at least 2 time units have elapsed since the last turn off. Switches off automatically 15 time units after the last on.



Control State (Location) Reachability Problem

- Given an ITA, check if a particular location is reachable from the initial states
- Is this problem easier or harder than general reachability?
- Is this problem is decidable?
- Key idea:
  - Construct a Finite State Machine that is a time-abstract bisimilar to the ITA
  - Check reachability of FSM

Key idea: put states that behave identically in the same equivalence class

When two states $\mathbf{x}_1$ and $\mathbf{x}_2$ in Q behave identically?

- $\mathbf{x}_1.loc = \mathbf{x}_2.loc$ and
- $\mathbf{x}_1$ and $\mathbf{x}_2$ satisfy the same set of clock constraints
  - For each clock y int($\mathbf{x}_1.y$) = int($\mathbf{x}_2.y$) or int($\mathbf{x}_1.y$) $\geq c_{\mathcal{A}y}$ and int($\mathbf{x}_2.y$) $\geq c_{\mathcal{A}y}$. ($c_{\mathcal{A}y}$ is the maxium clock guard of y)
  - For each clock y with $\mathbf{x}_1.y \leq c_{\mathcal{A}y}$, frac($\mathbf{x}_1.y$) $= 0$ iff frac($\mathbf{x}_2.y$) $= 0$
  - For any two clocks y and z with $\mathbf{x}_1.y \leq c_{\mathcal{A}y}$ and $\mathbf{x}_1.z \leq c_{\mathcal{A}z}$, frac($\mathbf{x}_1.y$) $\leq$ frac($\mathbf{x}_1.z$) iff frac($\mathbf{x}_2.y$) $\leq$ frac($\mathbf{x}_2.z$)

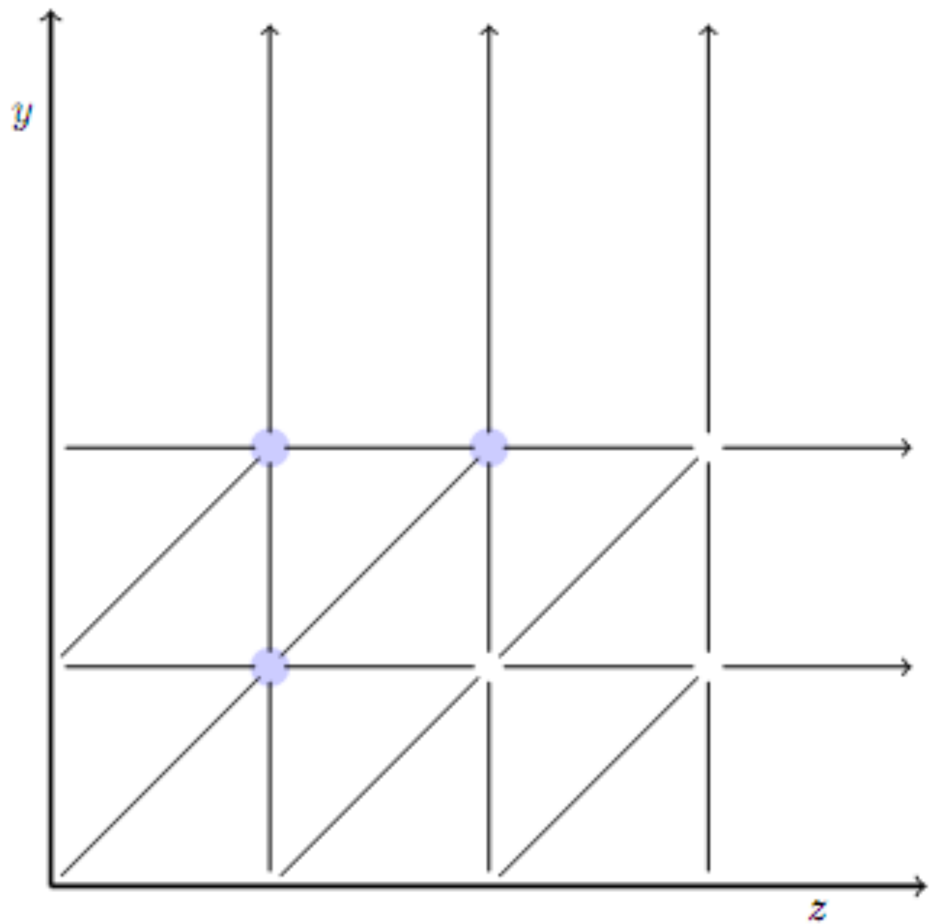
Lemma. This is a equivalence relation on Q

The partition of Q induced by this relation is are called clock regions

What do the clock regions look like?

Example of
Two Clocks

X = {y,z}
$c_{\mathcal{A}y}$ = 2
$c_{\mathcal{A}z}$ = 3



Complexity

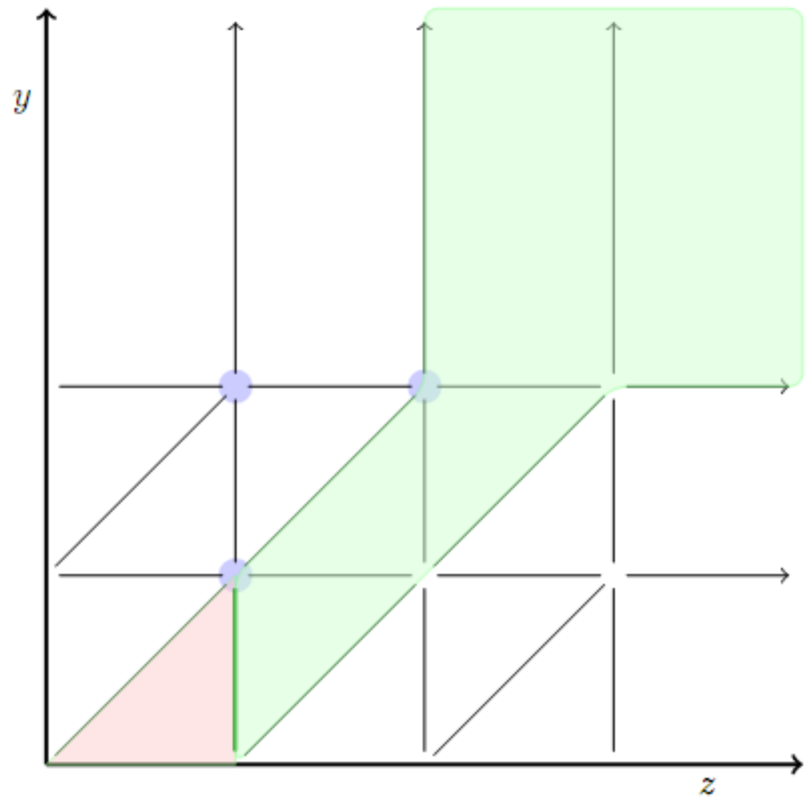
- **Lemma**. The number of clock regions is bounded by $|X|!\, 2^{|X|} \prod_{z \in X} (2c_{\mathcal{A}z} + 2)$.

Region Automaton

- ITA (clock constants) defines the clock regions

- Now we add the "appropriate transitions" between the regions to create a finite automaton which gives a time abstract bisimulation of the ITA with respect to control state reachability
  - Time successors: Consider two clock regions $\gamma \text{ and } \gamma'$, we say that γ' is a time successor of γ if there exits a trajectory of ITA starting from γ that ends in γ'
  - Discrete transitions: Same as the ITA

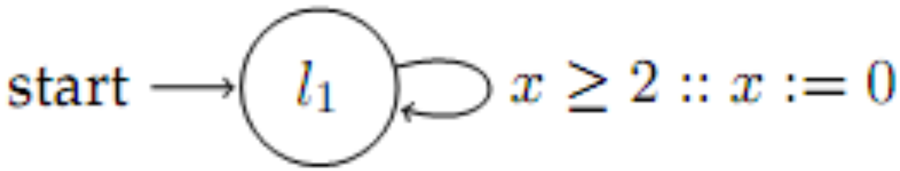
Time Successors



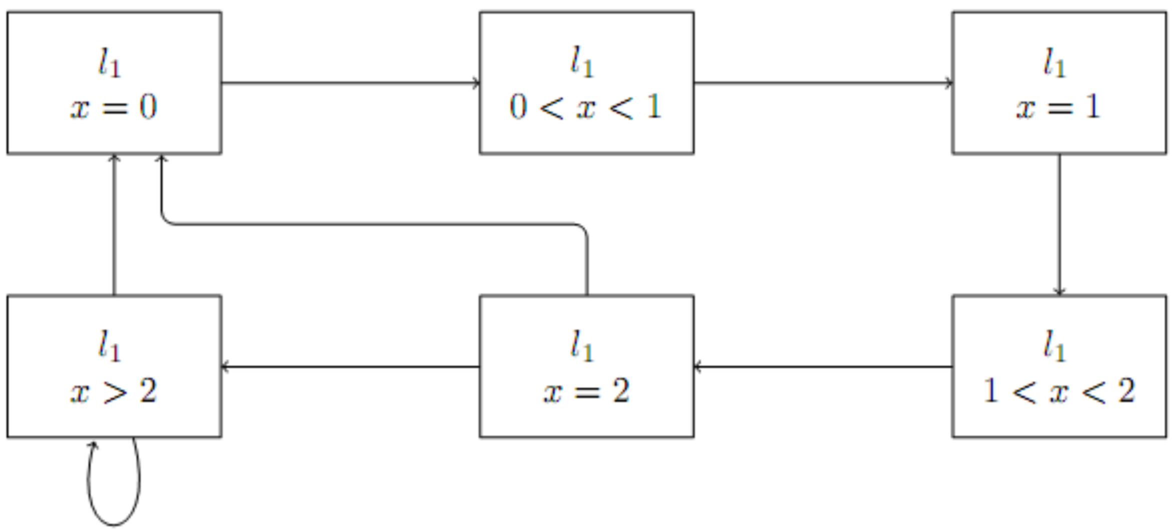
The clock regions in blue are time successors of the clock region in red.

Example 1: Region Automata

ITA

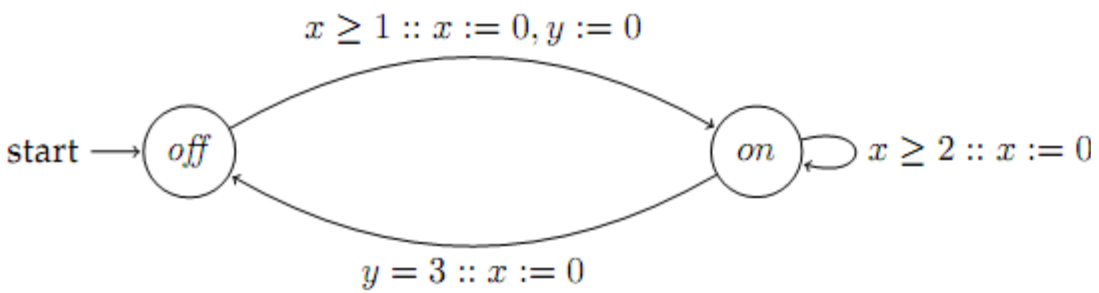


Corresponding FA

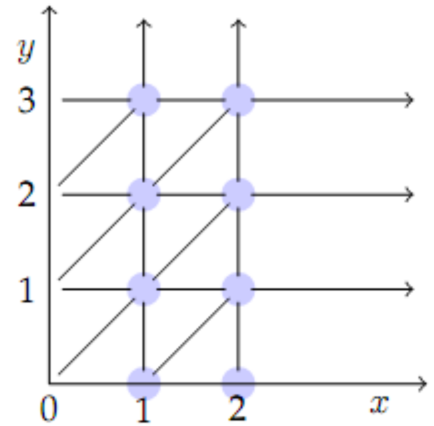


Example 2

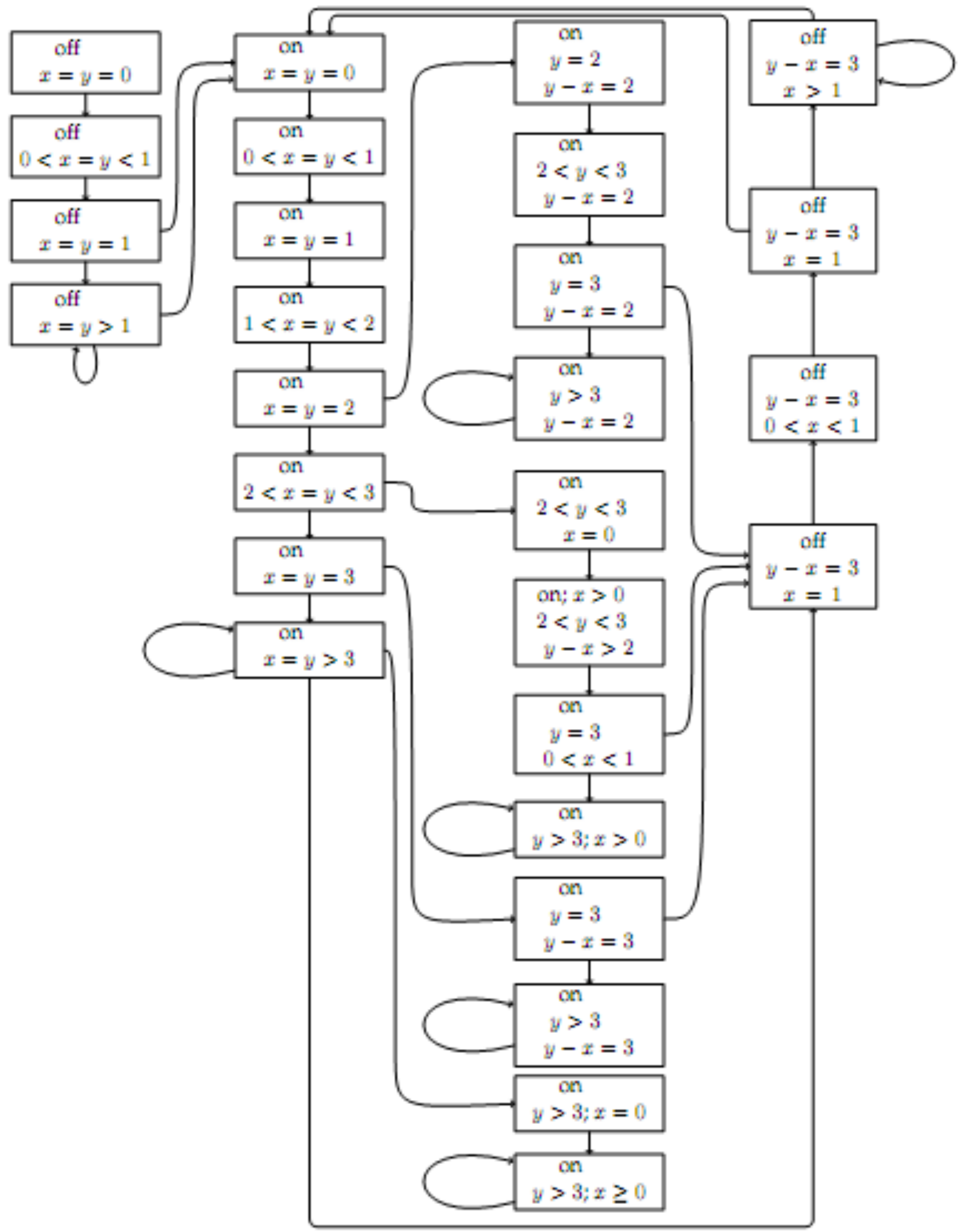
ITA



Clock Regions



Corresponding FA



$|X|!\, 2^{|X|} \prod_{z \in X} (2c_{\mathcal{A}z} + 2)$

Drastically increasing with the number of clocks

Clocks and Rational Clock Constraints

- A clock variable x is a continuous (analog) variable of type real such that along any trajectory τ of x, for all t ∈ $\tau.dom, (\tau \downarrow x)(t) = t$.
- For a set X of clock variables, the set Φ(X) of integral clock constraints are expressions defined by the syntax:

  g ::= x ≤ $q \mid x \geq q \mid \neg\, g \ \mid g_1 \wedge \ g_2$

  where $x \in X \text{ and } q \in \ \mathbb{Q}$
- Examples: x = 10.125; x ∈ [2.99, 5); true are valid rational clock constraints
- Semantics of clock constraints $[g]$

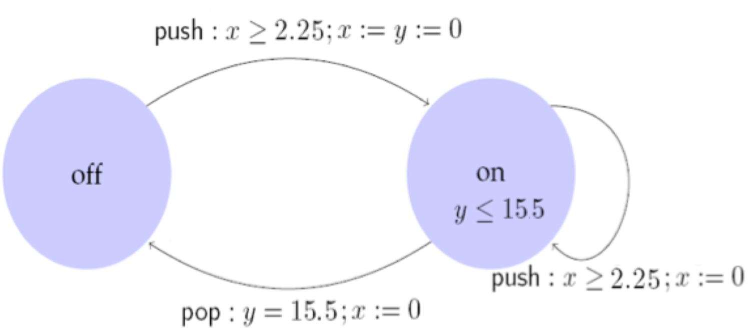
Step 1. Rational Timed Automata

- **Definition.** A rational timed automaton is a HA $\boldsymbol{\mathcal{A}} = \langle V, Q, \Theta, A, \mathcal{D}, \mathcal{T} \rangle$ where
  - V = X ∪ $\{loc\}$, where X is a set of n clocks and l is a discrete state variable of finite type Ł
  - A is a finite set
  - $\mathcal{D}$ is a set of transitions such that
    - The guards are described by rational clock constraings $\Phi(X)$
    - $\langle x, l \rangle - a \rightarrow \langle x', l' \rangle$ implies either $x' = x$ or $x = 0$
  - $\mathcal{T}$ set of clock trajectories for the clock variables in X

Example: Rational Light switch

Switch can be turned on whenever at least 2.25 time units have elapsed since the last turn off or on.
Switches off automatically 15.5 time units after the last on.

automaton Switch
  internal push; pop
    variables
      internal x, y:Real := 0, loc:{on,off} := off
    transitions
      push
        pre x >=2.25
        eff if loc = on **then** y := 0 **fi;** x := 0; loc := off
      pop
        pre y = 15.5 ∧ loc = off
        eff x := 0
    trajectories
      invariant loc = on ∨ loc = off
      stop when y = 15.5 ∧ loc = off
      evolve d(x) = 1; d(y) = 1



Control State (Location) Reachability Problem

- Given an RTA, check if a particular location is reachable from the initial states
- Is problem decidable?
- Yes
- Key idea:
  - Construct a ITA that is time-abstract bisimilar to the given RTA
  - Check CSR for ITA

Construction of ITA from RTA

- Multiply all rational constants by a factor q that make them integral
- Make d(x) = q for all the clocks

- RTA Switch is bisimilar to ITA Iswitch

- Simulation relation R is given by
- (u,s) ∈ R iff **u**.x = 4 **s**.x and **u**.y = 4 **s**.y

automaton ISwitch
internal push; pop
variables
  internal x, y:Real := 0, loc:{on,off} := off
transitions
  push
    pre x >= 9
    eff if loc = on **then** y := 0 **fi;** x := 0; loc := off
  pop
    pre y = 62 ∧ loc = off
    eff x := 0
trajectories
  invariant loc = on ∨ loc = off
  stop when y = 62 ∧ loc = off
  evolve d(x) = 4; d(y) = 4

Step 2. Multi-Rate Automaton

- **Definition.** A multirate automaton is $\boldsymbol{\mathcal{A}} = \langle V, Q, \Theta, A, \mathcal{D}, \mathcal{T} \rangle$ where
  - V = X ∪ $\{loc\}$, where X is a set of n continuous variables and loc is a discrete state variable of finite type Ł
  - A is a finite set of actions
  - $\mathcal{D}$ is a set of transitions such that
    - The guards are described by rational clock constraings $\Phi(X)$
    - $\langle x, l \rangle - a \rightarrow \langle x', l' \rangle$ implies either $x' = c \text{ or } x' = x$
  - $\mathcal{T}$ set of trajectories such that

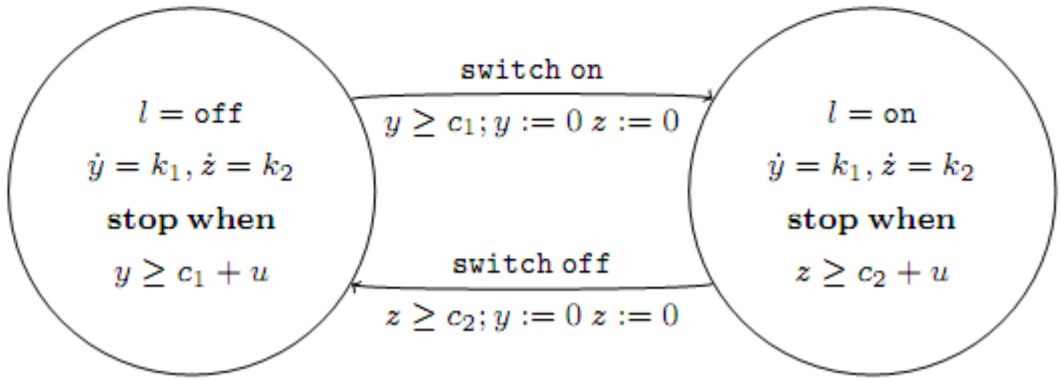
    for each variable $x \in X \; \exists k \text{ such that } \tau \in \mathcal{T}, t \in \tau.dom$

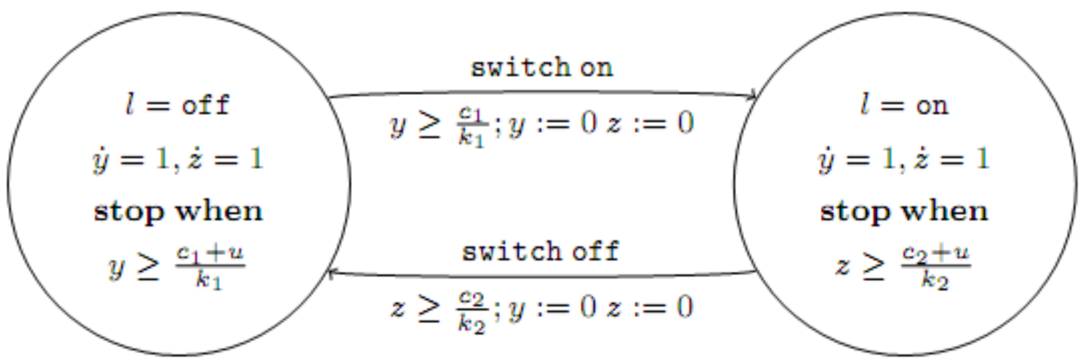
$$\tau(t).x = \tau(0).x + k\,t$$

Control State (Location) Reachability Problem

- Given an MRA, check if a particular location is reachable from the initial states
- Is problem is decidable? Yes
- Key idea:
  - Construct a RTA that is bisimilar to the given MRA

Example: Multi-rate to rational TA





Step 3. Rectangular HA

Definition. An rectangular hybrid automaton (RHA) is a HA $\boldsymbol{\mathcal{A}} = \langle V, A, \mathcal{T}, \mathcal{D} \rangle$ where

- V = X ∪ $\{loc\}$, where X is a set of n continuous variables and loc is a discrete state variable of finite type Ł
- A is a finite set
- $\mathcal{T} = \cup_\ell \mathcal{T}_\ell$ set of trajectories for X
  - For each $\tau \in \mathcal{T}_\ell, x \in X$ either (i) $d(x) = k_\ell$ or (ii) $d(x) \in [k_{\ell 1}, k_{\ell 2}]$
  - Equivalently, (i) $\tau(t)\lceil x = \tau(0)\lceil x + k_\ell t$
    (ii) $\tau(0)\lceil x + k_{\ell 1} t \leq \tau(t)\lceil x \leq \tau(0)\lceil x + k_{\ell 2} t$
- $\mathcal{D}$ is a set of transitions such that
  - Guards are described by rational clock constrainGs
  - $\langle x, l \rangle \rightarrow_a \langle x', l' \rangle$ implies $x' = x\ or x' \in [c_1, c_2]$

CSR Decidable for RHA?

- Given an RHA, check if a particular location is reachable from the initial states?
- Is this problem decidable? No
  - **[Henz95]** Thomas Henzinger, Peter Kopke, Anuj Puri, and Pravin Varaiya. [What's Decidable About Hybrid Automata?. Journal of Computer and System Sciences, pages 373–382. ACM Press, 1995.]
  - CSR for RHA reduction to Halting problem for 2 counter machines
  - Halting problem for 2CM known to be undecidable
  - Reduction in next lecture

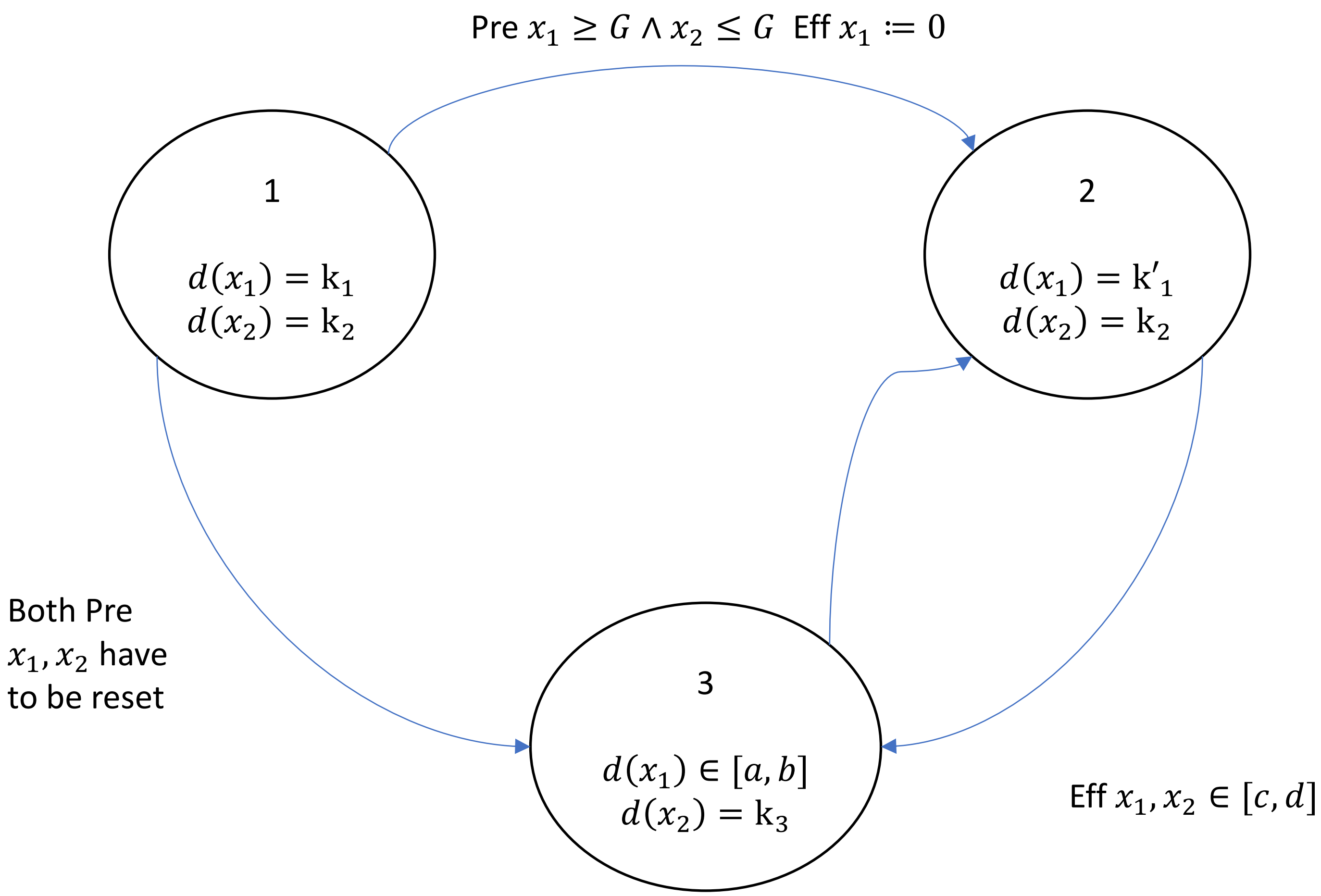
Step 4. Initialized Rectangular HA

Definition. An initialized rectangular hybrid automaton (IRHA) is a RHA $\boldsymbol{\mathcal{A}}$ where

- V = X ∪ $\{loc\}$, where X is a set of n continuous variables and $\{loc\}$ is a discrete state variable of finite type Ł
- A is a finite set
- $\mathcal{T} = \cup_{\ell} \mathcal{T}_{\ell}$ set of trajectories for X
  - For each $\tau \in \mathcal{T}_{\ell}, x \in X$ either (i) $d(x) = k_{\ell}$ or (ii) $d(x) \in [k_{\ell 1}, k_{\ell 2}]$
  - Equivalently, (i) $\tau(t)\lceil x = \tau(0)\lceil x + k_{\ell} t$

    (ii) $\tau(0)\lceil x + k_{\ell 1} t \leq \tau(t)\lceil x \leq \tau(0)\lceil x + k_{\ell 2} t$
- $\mathcal{D}$ is a set of transitions such that
  - Guards are described by rational clock constrings
  - $\langle x, l\rangle \rightarrow_a \langle x', l'\rangle$ implies if dynamics changes from ℓ to ℓ' then $x' \in [c_1, c_2]$, otherwise $x' = x$

Example: Rectangular Initialized HA

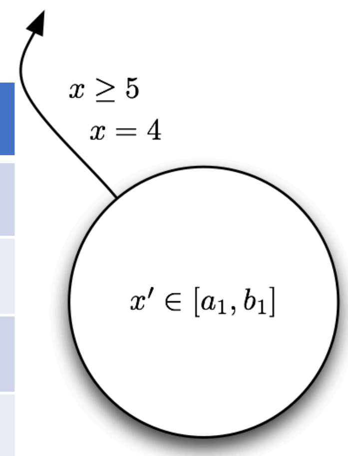


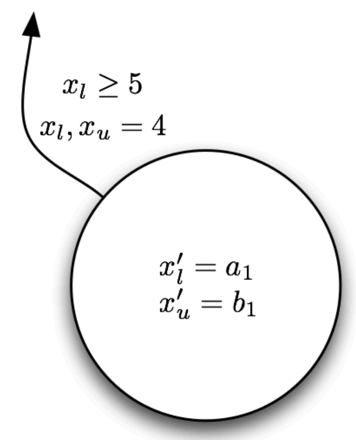
CSR Decidable for IRHA?

- Given an IRHA, check if a particular location is reachable from the initial states
- Is this problem decidable? Yes
- Key idea:
  - Construct a 2n-dimensional **initialized m**ulti-rate automaton that is bisimilar to the given IRHA
  - Construct a ITA that is bisimilar to the Singular TA

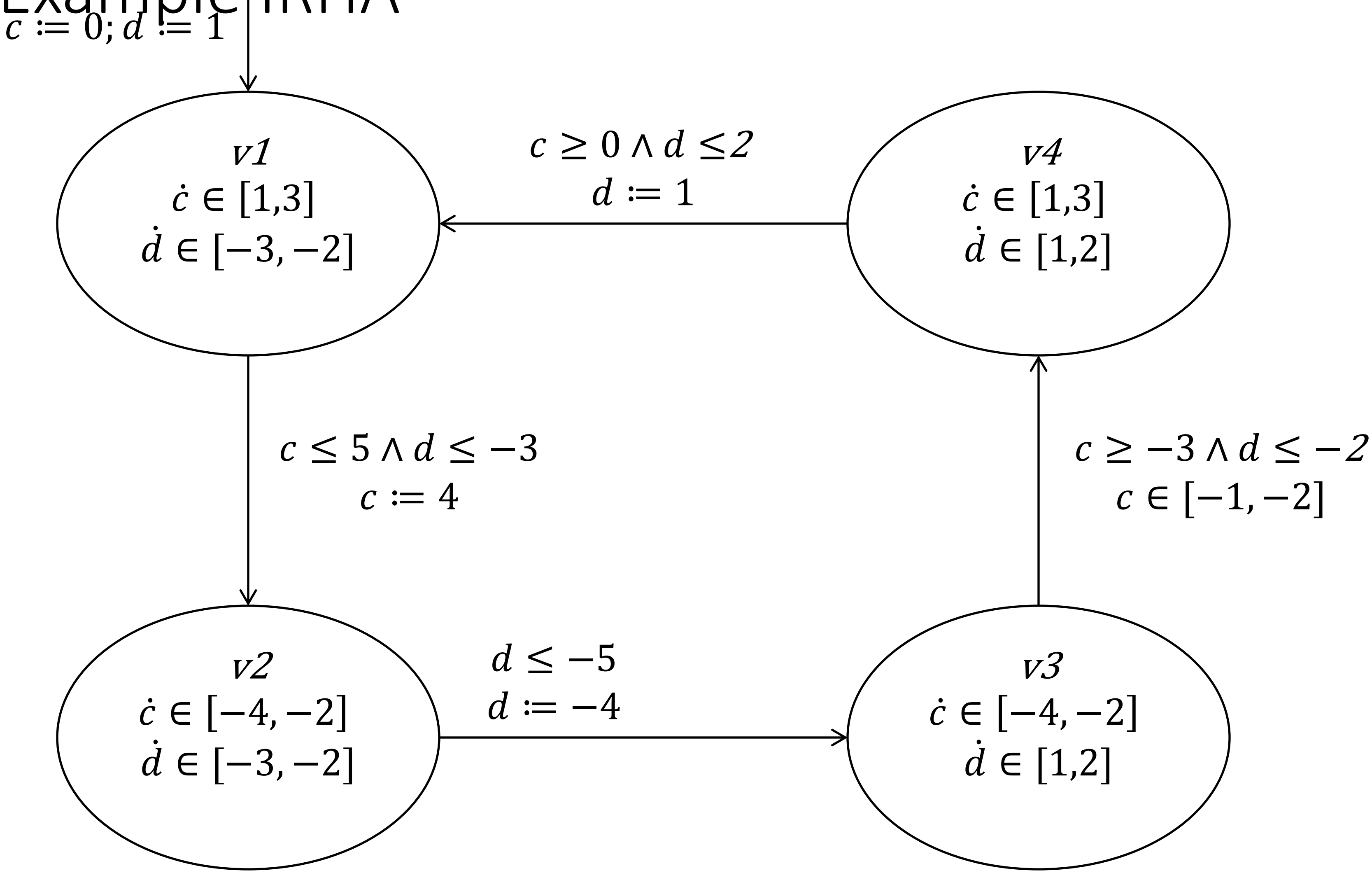
Split every variable into two variables---
tracking the upper and lower bounds

IRHA	MRA
x	x_ℓ ; x_u
Evolve: $d(x) \in [a_1, b_1]$	Evolve: $d(x_\ell) = a_1; d(x_u) = b_1$
Eff: $x' \in [a_1, b_1]$	Eff: $x_\ell = a_1; x_u = b_1$
$x' = c$	$x_\ell = x_u = c$
Guard: $x \geq 5$	$x_l \geq 5$
	$x_l < 5 \wedge x_u \geq 5$ Eff $x_l = 5$

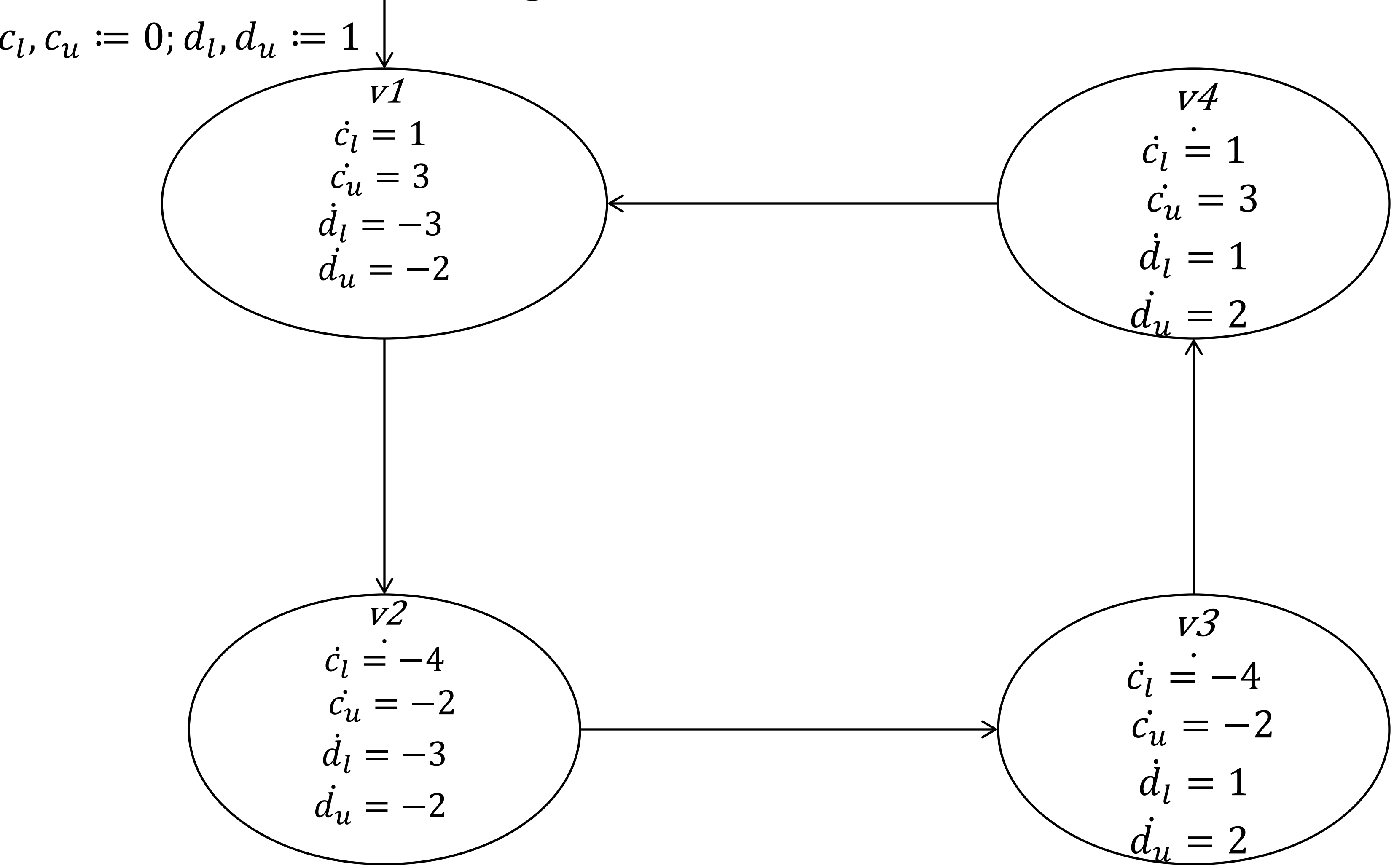




Example IRHA



Initialized Singular HA



Transitions

$v1$

$\dot{c_l} = 1$

$\dot{c_u} = 3$

$\dot{d_l} = -3$

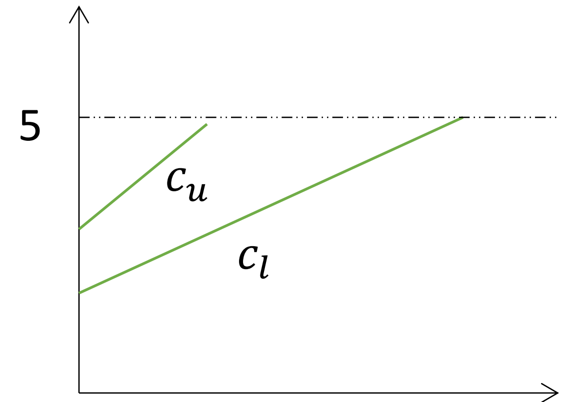
$\dot{d_u} = -2$

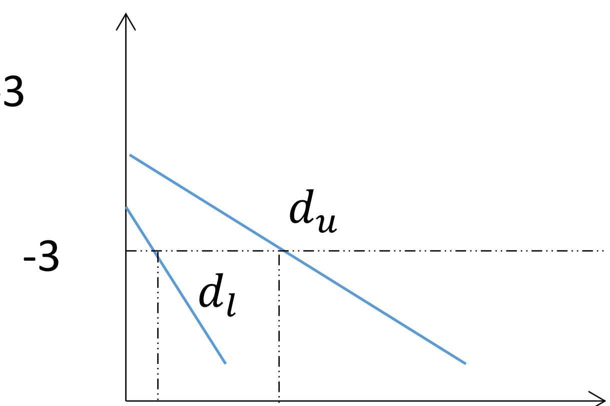
$c_l \leq 5$

$c_l, c_u \coloneqq 4$

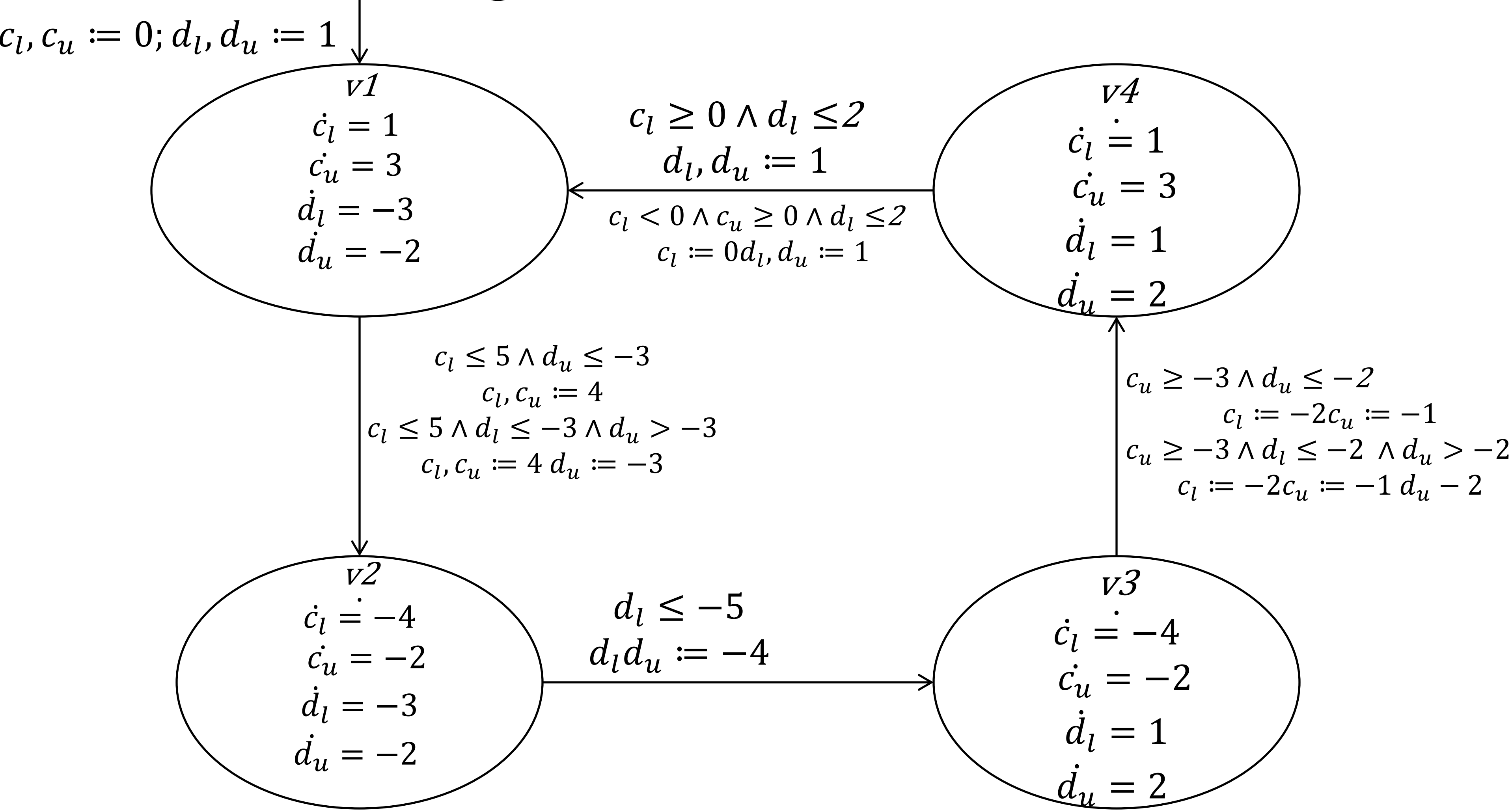
$d_u \leq -3$ *no reset*

$d_u > -3 \wedge d_l \leq -3$ $d_u \coloneqq$-3





Initialized Singular HA



Can this be further generalized ?

- For initialized Rectangular HA, control state reachability is decidable
  - Can we drop the initialization restriction?
    - No, problem becomes undecidable
  - Can we drop the rectangular restriction?
    - No, problem becomes undecidable
  - Tune in in a week

Data structures for representing sets

- Hyperrectangles
  - $[[g_1; g_2]] = \{x \in R^n \mid \; ||x - g_1||_\infty \leq ||g_2 - g_1||_\infty\} = \Pi_i [g_{1i}, g_{2i}]$
- Polyhedra
- Zonotopes
- Ellipsoids
- Support functions

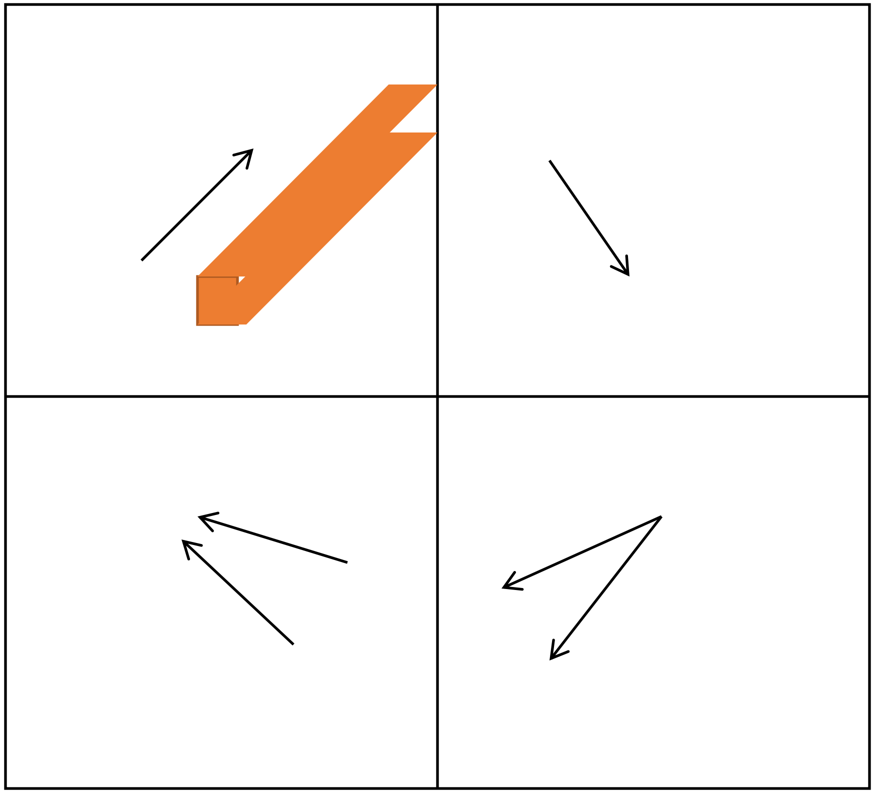
Verification in tools

Algorithm: BasicReach
Input: $\mathbf{A} = \langle V, \Theta, A, \mathbf{D}, \mathbf{T} \rangle$, $d > 0$
    $Rt, Reach: val(V)$
    $Rt := \Theta;$
    $Reach := \emptyset;$
    While $(Rt \not\subseteq \text{Reach})$
        $Reach := Reach \cup Rt;$
        $Rt := Rt \cup Post_{\mathbf{D}}(Rt);$
        $Rt := Post_{\mathbf{T}(d)}(Rt);$
Output: $Reach$

Algorithm: Post_{D}
\\ computes post of all transitions
Input: $A, \mathbf{D}, S_{in}$
    $S_{out} = \emptyset$
    For each $a \in A$
        For each $\langle g_1, g_2 \rangle \in S_{in}$
            If $[\![g_1, g_2]\!] \cap [\![g_{ga1}, g_{ga2}]\!] \neq \emptyset$
                $S_{out} := S_{out} \cup \langle g_{ra1}, g_{ra2} \rangle$
Output: S_{out}

Algorithm: $\text{Post}_{\mathbf{T}(\mathbf{d})}$
\\ computes post of all trajectories
Input: $A, \mathbf{T}, S_{in}, d$
$S_{out} = \emptyset$
    For each $\ell \in L$
        For each $\langle g_1, g_2 \rangle \in S_{in}$
            $P := \cup_{t \leq d} [\![g_1, g_2]\!] \oplus [\![t\, g_{\ell 1}, t\, g_{\ell 2}]\!]$
            $S_{out} := S_{out} \cup Approx(P)$
Output: S_{out}

Reachability Computation with polyhedra



Portion of Navigation benchmark

$x' = k \rightarrow Post([a_1, a_2]) = \exists t \; [a_1 + kt, a_2 + kt] = [a_1, \infty]$

the state is reachable if there exists a time when we reach it.

Summary

- ITA: (very) Restricted class of hybrid automata
  - Clocks, integer constraints
  - No clock comparison, linear
- Control state reachability with Alur-Dill's algorithm (region automaton construction)
- Rational coefficients
- Multirate Automata
- Initialized Rectangular Hybrid Automata
- HyTech, PHAVer use polyhedral reachability computations

Summary

- ITA: (very) Restricted class of hybrid automata
  - Clocks, integer constraints
  - No clock comparison, linear
- Control state reachability
- Alur-Dill's algorithm
  - Construct finite bisimulation (region automaton)
  - Idea is to lump together states that behave similarly and reduce the size of the model
- UPPAAL model checker based on similar model of timed automata