CTL Model Checking

Sayan Mitra

Verifying cyberphysical systems

mitras@illinois.edu

Verification thus far: Invariants and safety

Given a **hybrid automaton** $\mathcal{A} = \langle X, \Theta, A, \boldsymbol{\mathcal{D}}, \boldsymbol{T} \rangle$ and a candidate invariant $I \subseteq val(X)$ we can check whether I is an inductive invariant.

In which case $Reach_{\mathcal{A}}(\Theta) \subseteq I$

Given an **unsafe set** $U \subseteq val(X)$ we can check whether $I \cap U = \emptyset$ to infer that $Reach_{\mathcal{A}}(\Theta) \cap U = \emptyset$

What about more general types of requirements, 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?

Outline

- Temporal logics
  - Computational Tree Logic (CTL)
- CTL model checking
  - Setup
  - CTL syntax and semantics
  - Model checking algorithms
  - Example

- References: Model Checking, Second Edition, by Edmund M. Clarke, Jr., Orna Grumberg, Daniel Kroening, Doron Peled and Helmut Veith

- Principles of Model Checking, by Christel Baier and Joost-Pieter Katoen

Introduction to temporal logics

Temporal logics give a formal language for representing, and reasoning about, propositions qualified in terms of time or 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, applications in synthesis and monitoring

Setup: States are labeled

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

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

Notations

Automata with state labels but no action labels

$\mathcal{A} = \langle Q, Q_0, T, L \rangle$

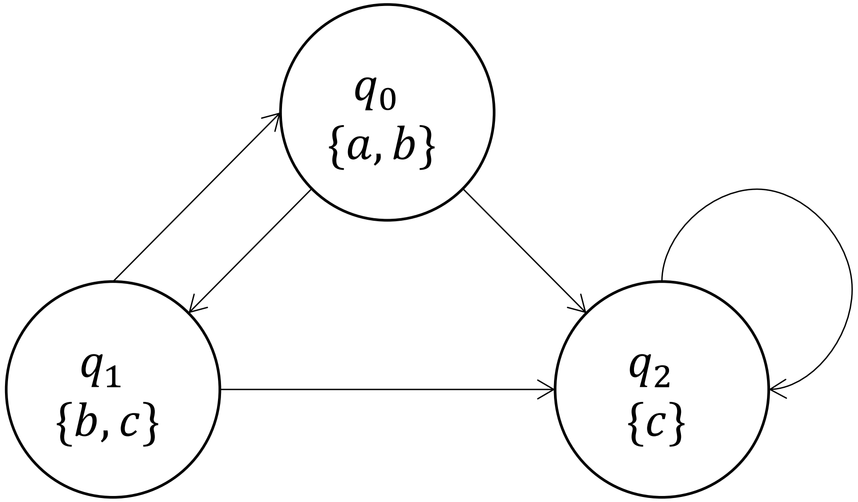
Executions (have no actions) $\alpha = q_0\ q_1 \ldots q_k = \alpha.lstate$

$\alpha[i] = q_i$

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

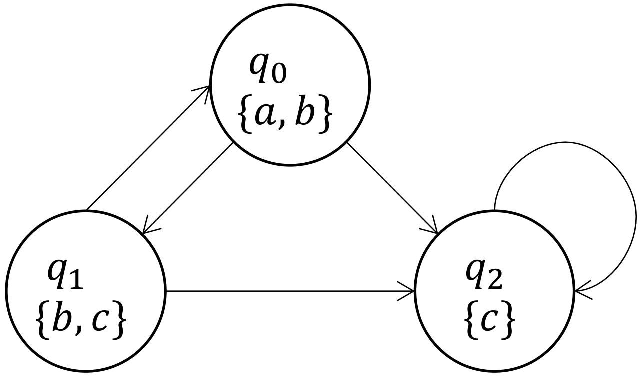
$AP = \{a, b, c\}$

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

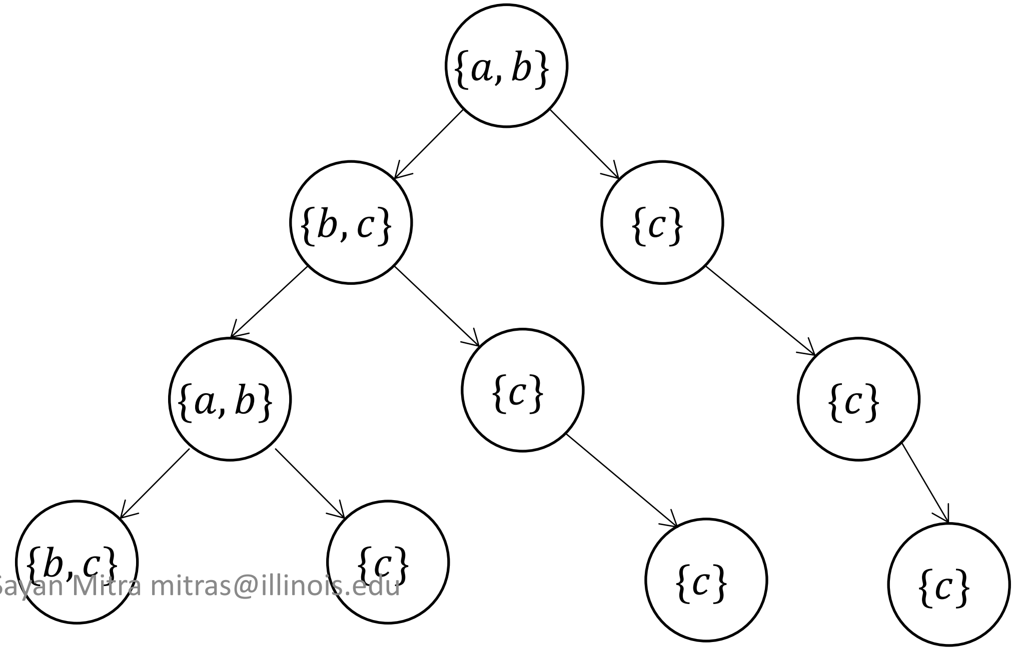


Computational tree logic (CTL)

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

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

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

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

$AXX\ a$; 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:

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

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

Universal CTL operators

$\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{G}$ can be used to derive other operators

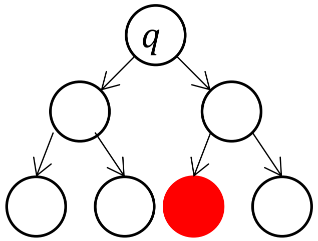
$true\ U\ f \equiv F\ f$

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

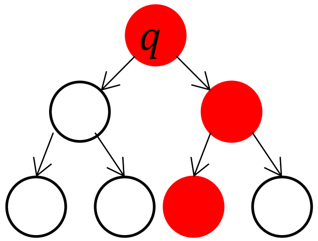
All ten combinations can be expressed using $\boldsymbol{EX}, \boldsymbol{EU}, \boldsymbol{EG}$

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	

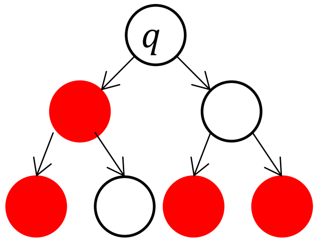
Visualizing semantics



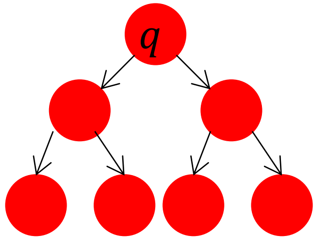
$q \vDash EF\ red$



$q \vDash EG\ red$



$q \vDash AF\ red$



$q \vDash AG\ red$

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

Algorithm works by structural induction on the depth of the formula

Explicit state model checking

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

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

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

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

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

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

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

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

(1) $q \in Q'$

(2) $\exists \alpha \in Execs_{\mathcal{A}}$, with $\alpha.fstate = q$ and $\alpha.lstate$ is in a nontrivial ***Strongly Connected Components*** 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 $q \in \boldsymbol{T}$
  $label(q) := label(q) \cup \{EGf_1\}$
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 $\mathrm{E}Gf_1 \notin label(q')$ then
        $label(q') := label(q') \cup \{EGf_1\}$
          $\boldsymbol{T} := \boldsymbol{T} \cup \{q\}$

Proposition. For any state $label(q) \ni EGf_1$ iff $q \vDash EGf_1$.

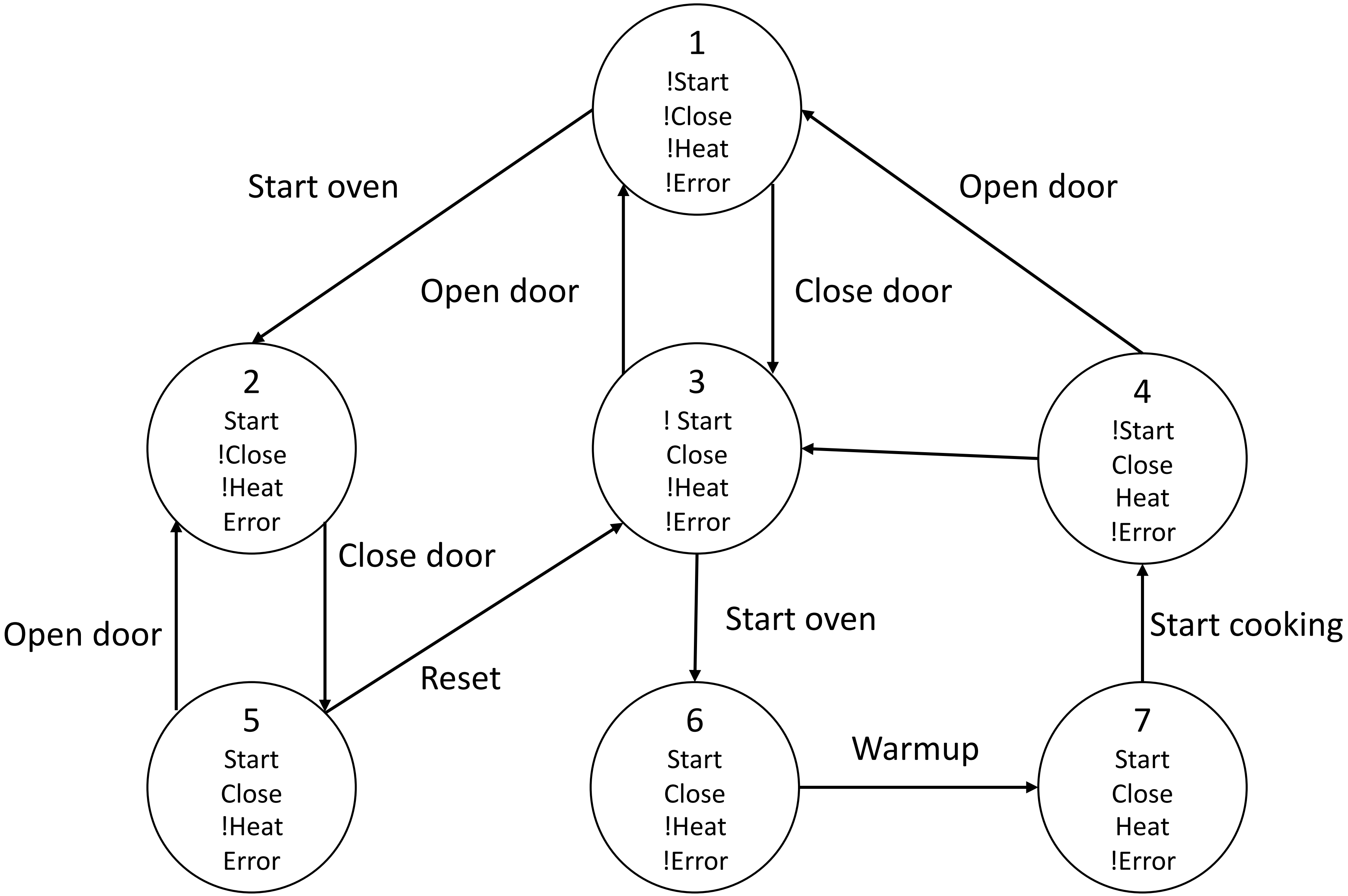
Proposition. Finite Q therefore terminates and in $O(|Q| + |T|)$ steps.

Summary

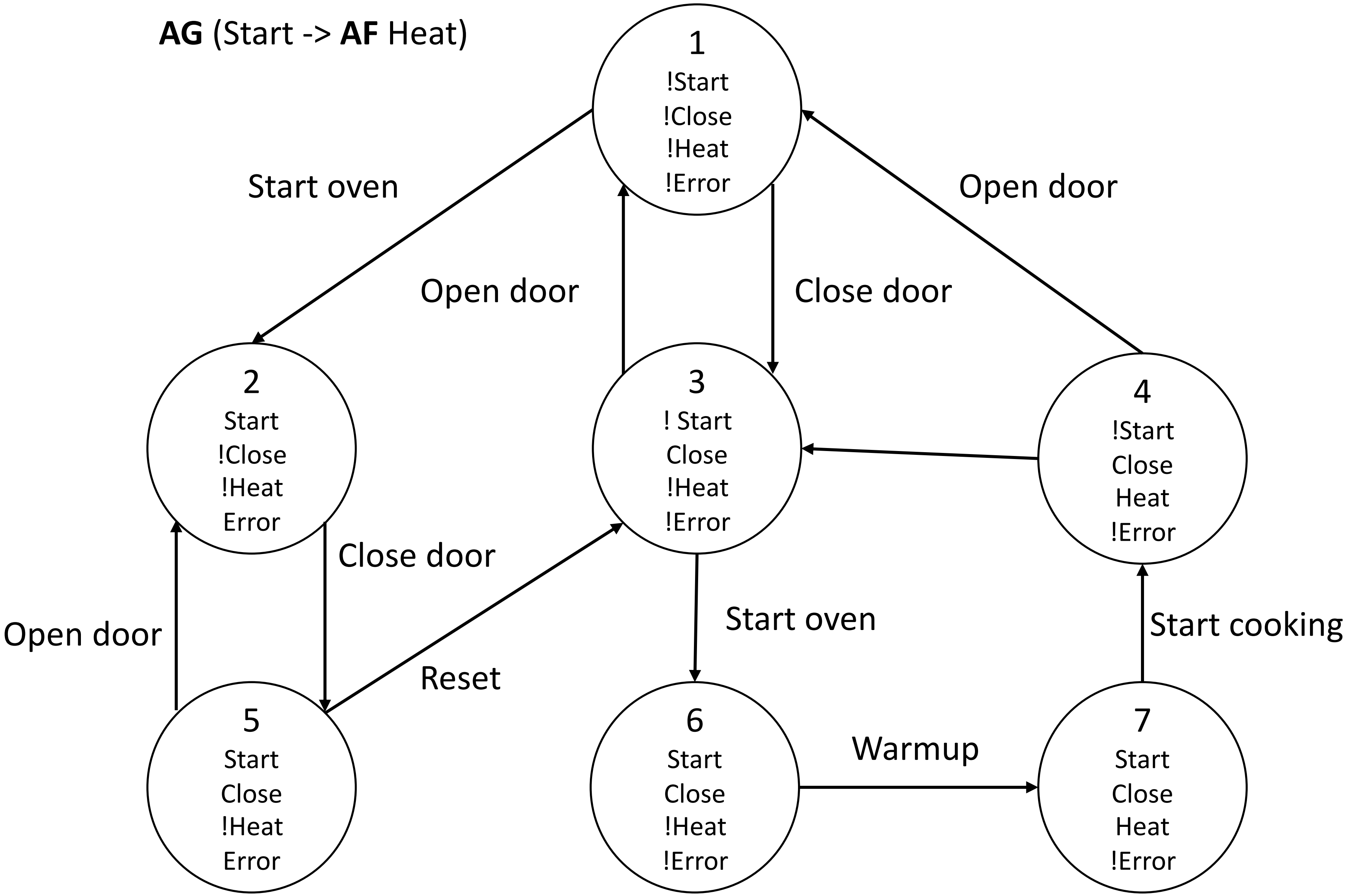
Explicit model checking algorithm input $\mathcal{A} \vDash f$?
Structural induction over CTL formula

$f = p,$	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 $label(q) := label(q) \cup f$
$f = EXf_1$	if $\exists q' \in \mathrm{Q}$ such that $(q, q') \in T$ and $f_1 \in label(q')$ then $label(q) := label(q) \cup f$
$f = E[f_1 U f_2]$	$CheckEU(f_1, f_2, Q, T, L)$
$f = EGf_1$	$CheckEG(f_1, Q, T, L)$

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

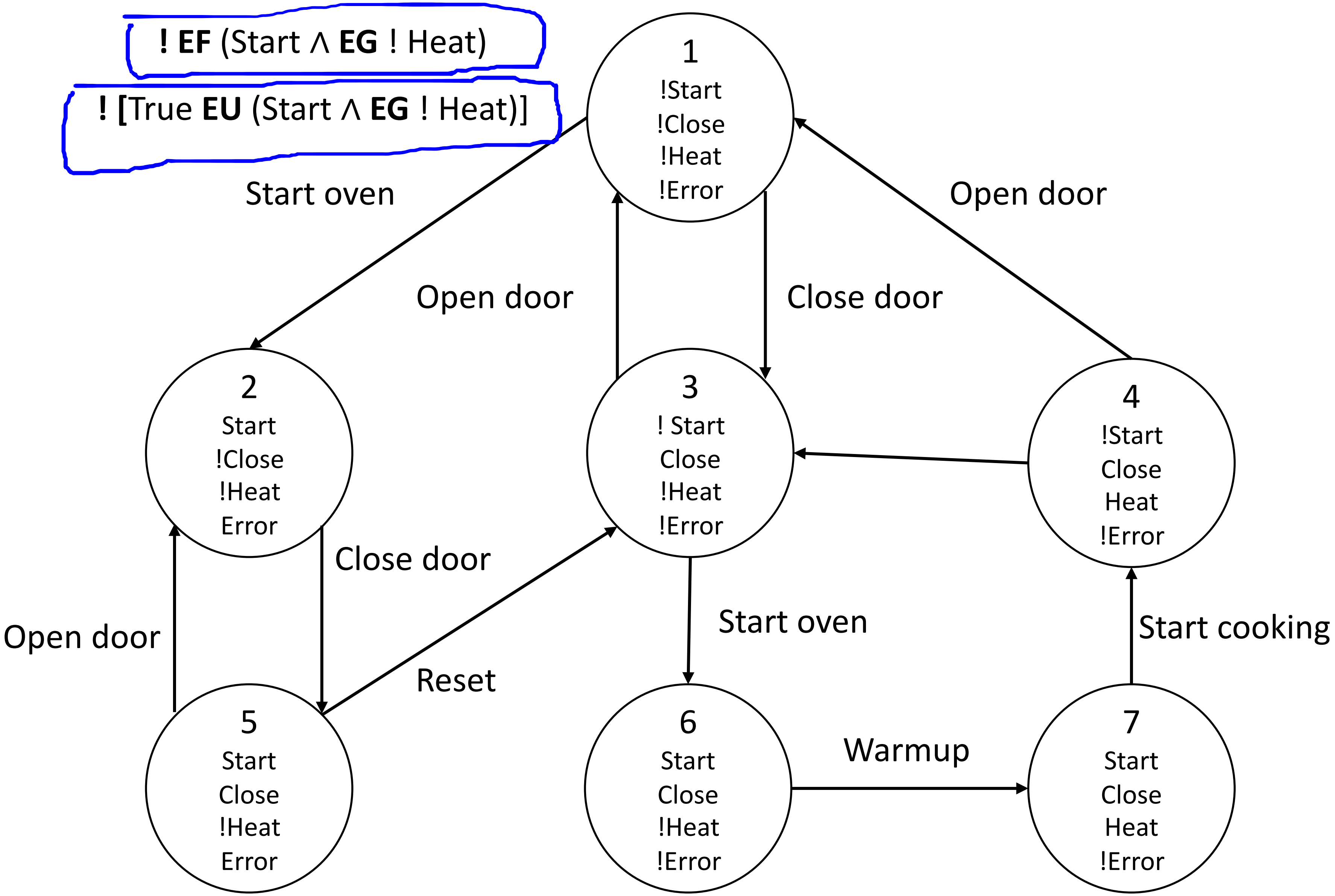

1
!Start
!Close
!Heat
!Error
Start oven
Open door
Close door
Open door
2
Start
!Close
!Heat
Error
3
! Start
Close
!Heat
!Error
4
!Start
Close
Heat
!Error
Close door
Open door
Reset
Start oven
Start cooking
5
Start
Close
!Heat
Error
6
Start
Close
!Heat
!Error
Warmup
7
Start
Close
Heat
!Error

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



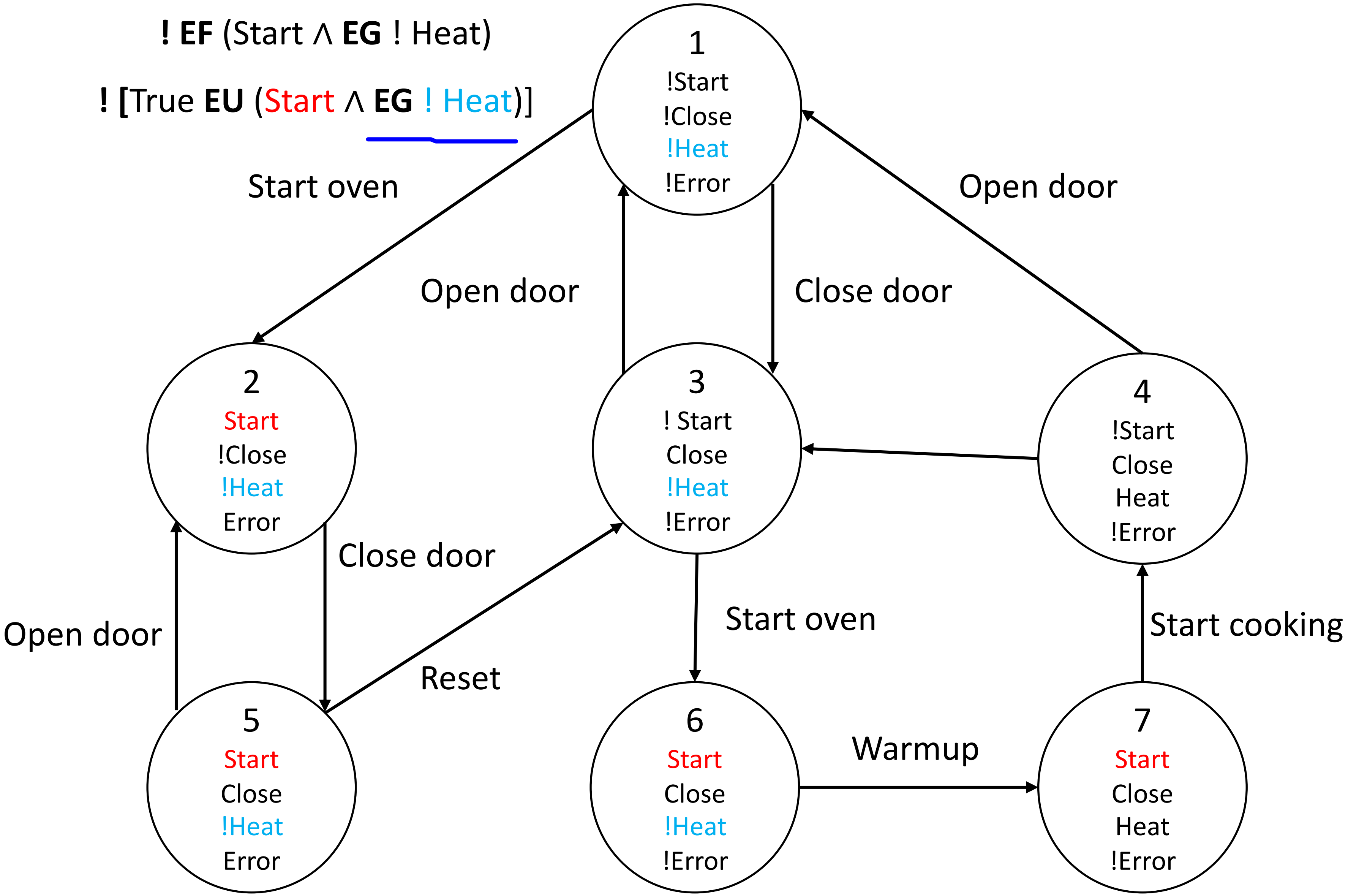
! EF (Start ∧ **EG** ! Heat)

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



! **EF** (Start ∧ **EG** ! Heat)

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

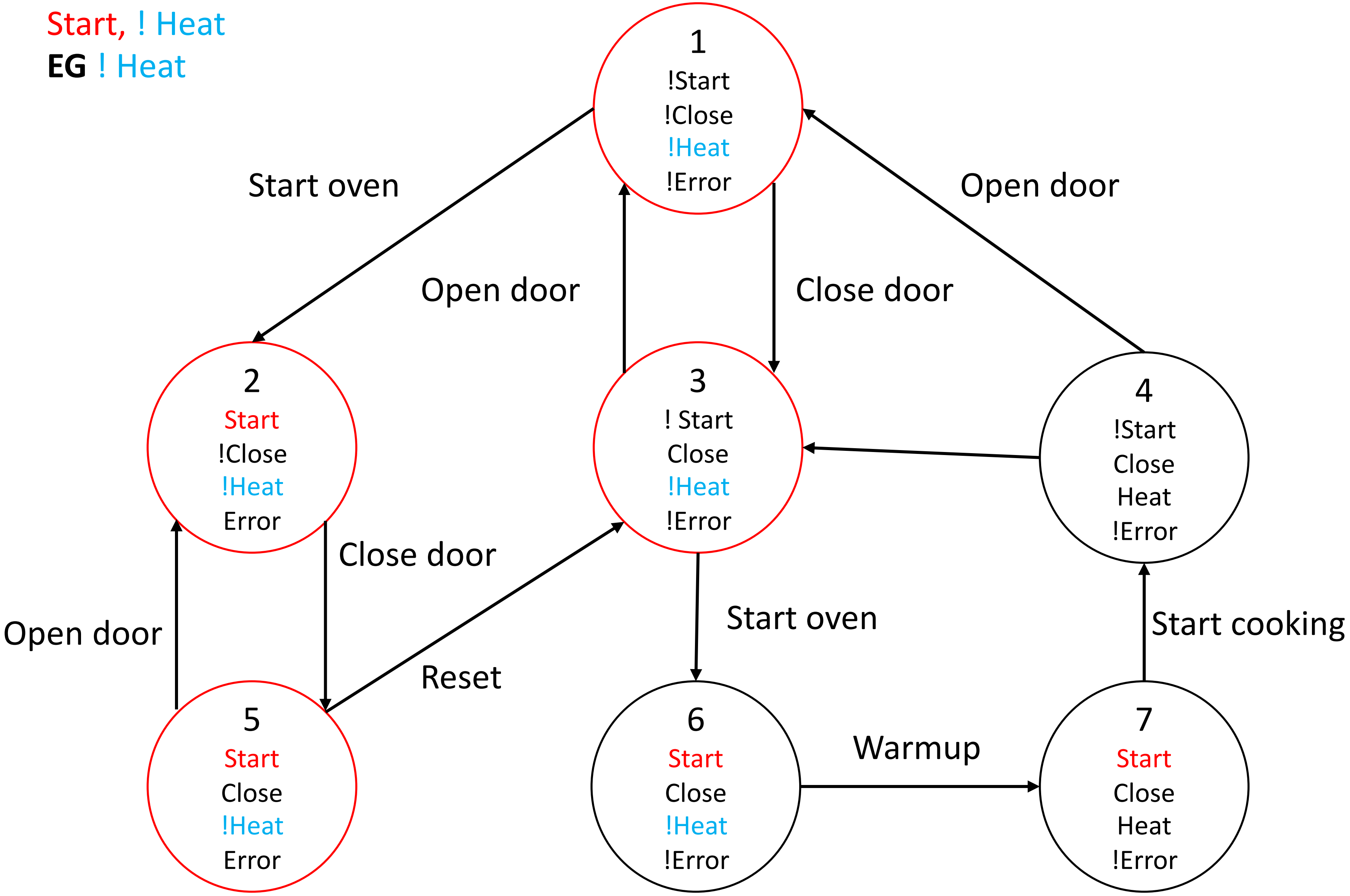


! **EF** (Start ∧ **EG** ! Heat)

Start, ! Heat

EG ! Heat

Nontrivial SCC of ! Heat



! **EF** (Start ∧ **EG** ! Heat)

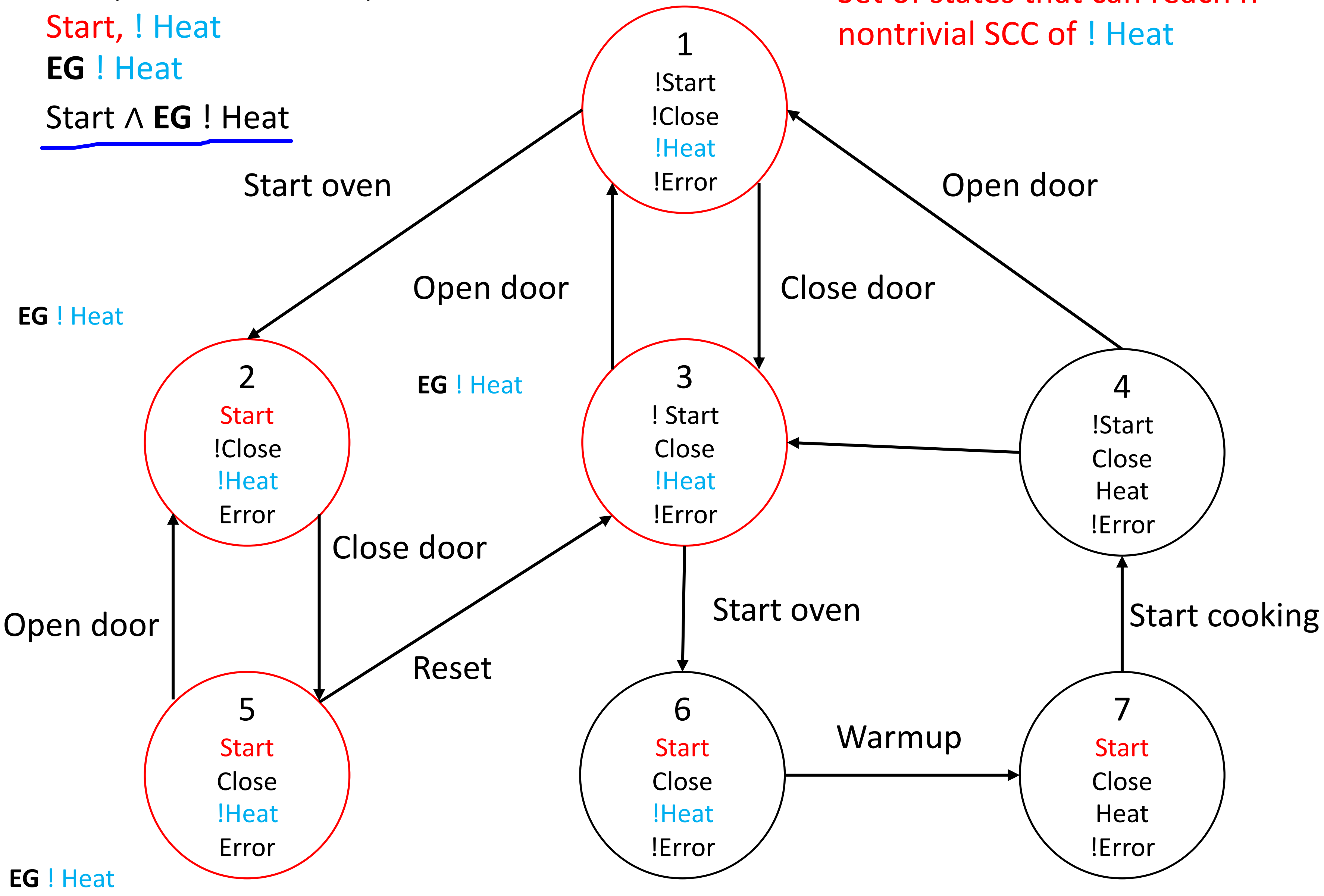
Start, ! Heat

EG ! Heat

Start ∧ **EG** ! Heat

EG ! Heat

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

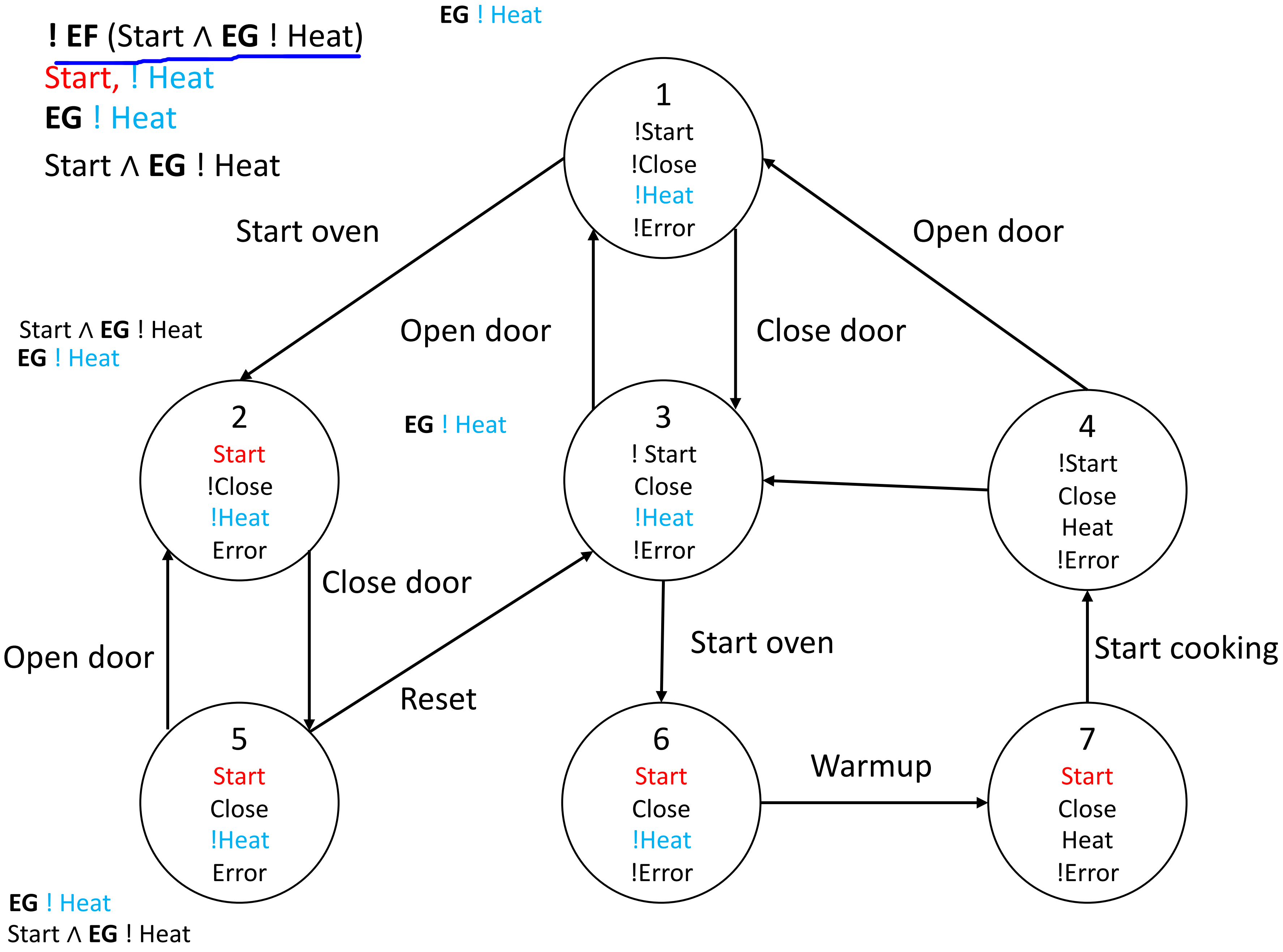


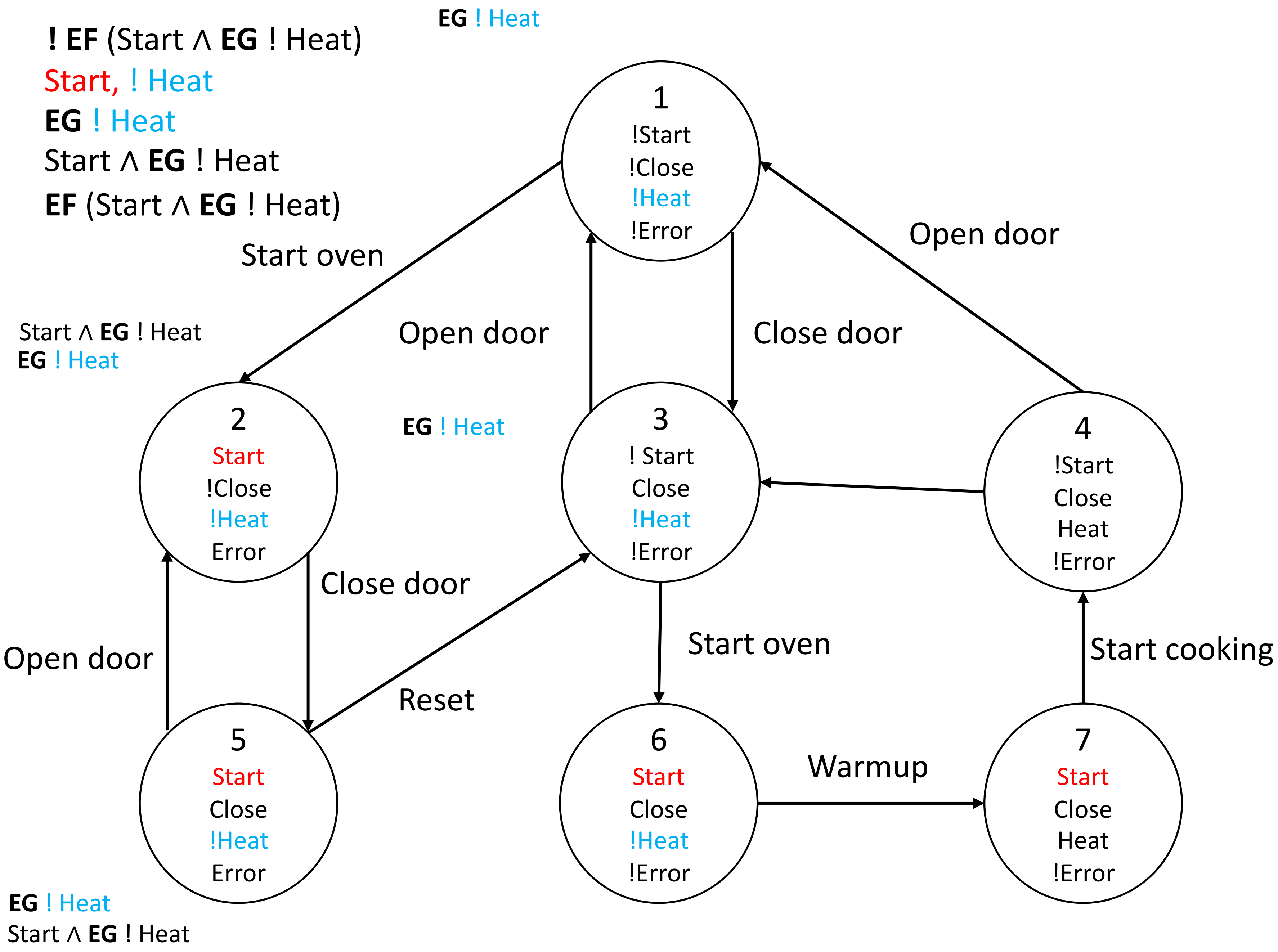
! **EF** (Start ∧ **EG** ! Heat)

Start, ! Heat

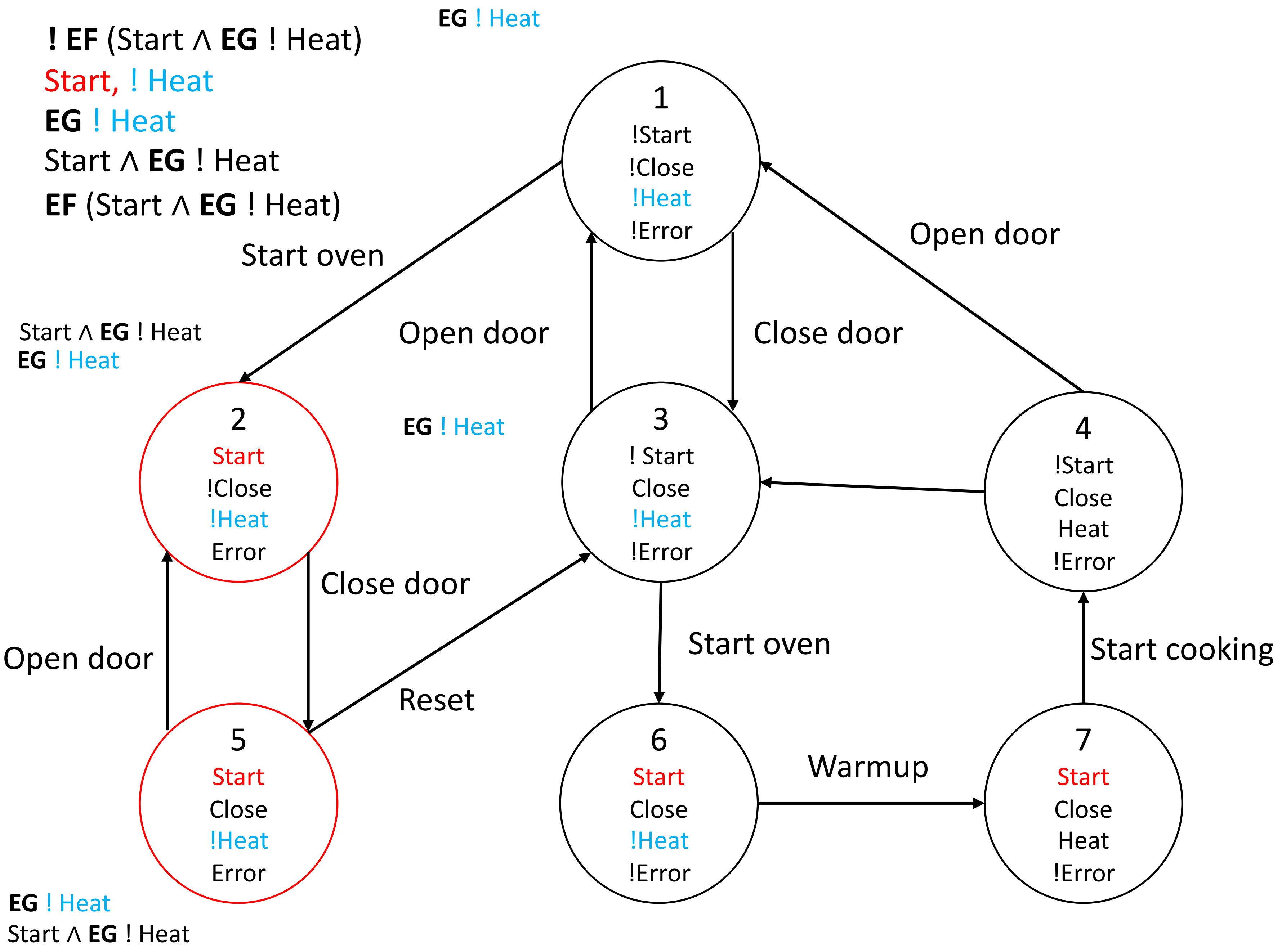
EG ! Heat

Start ∧ **EG** ! Heat

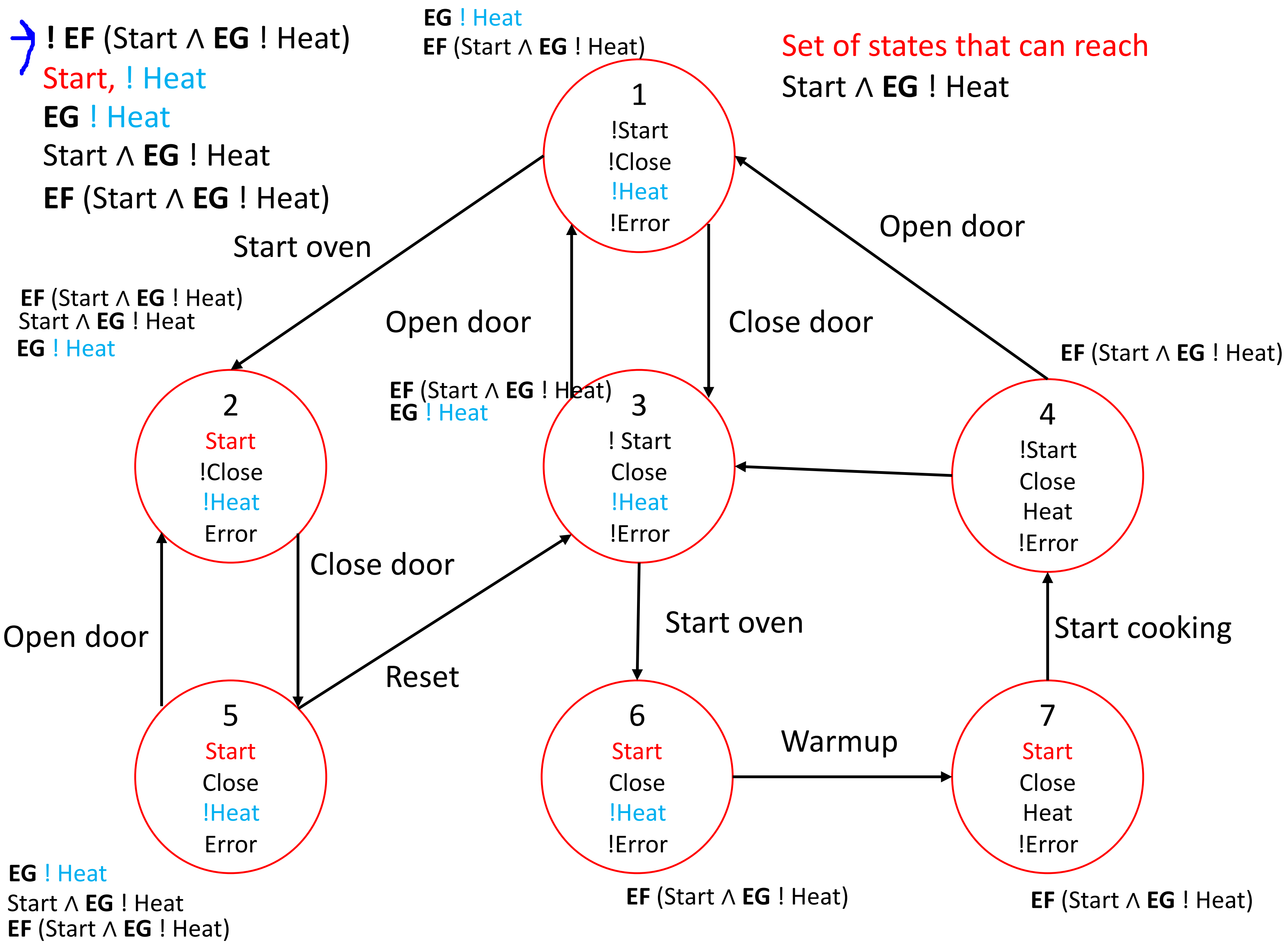


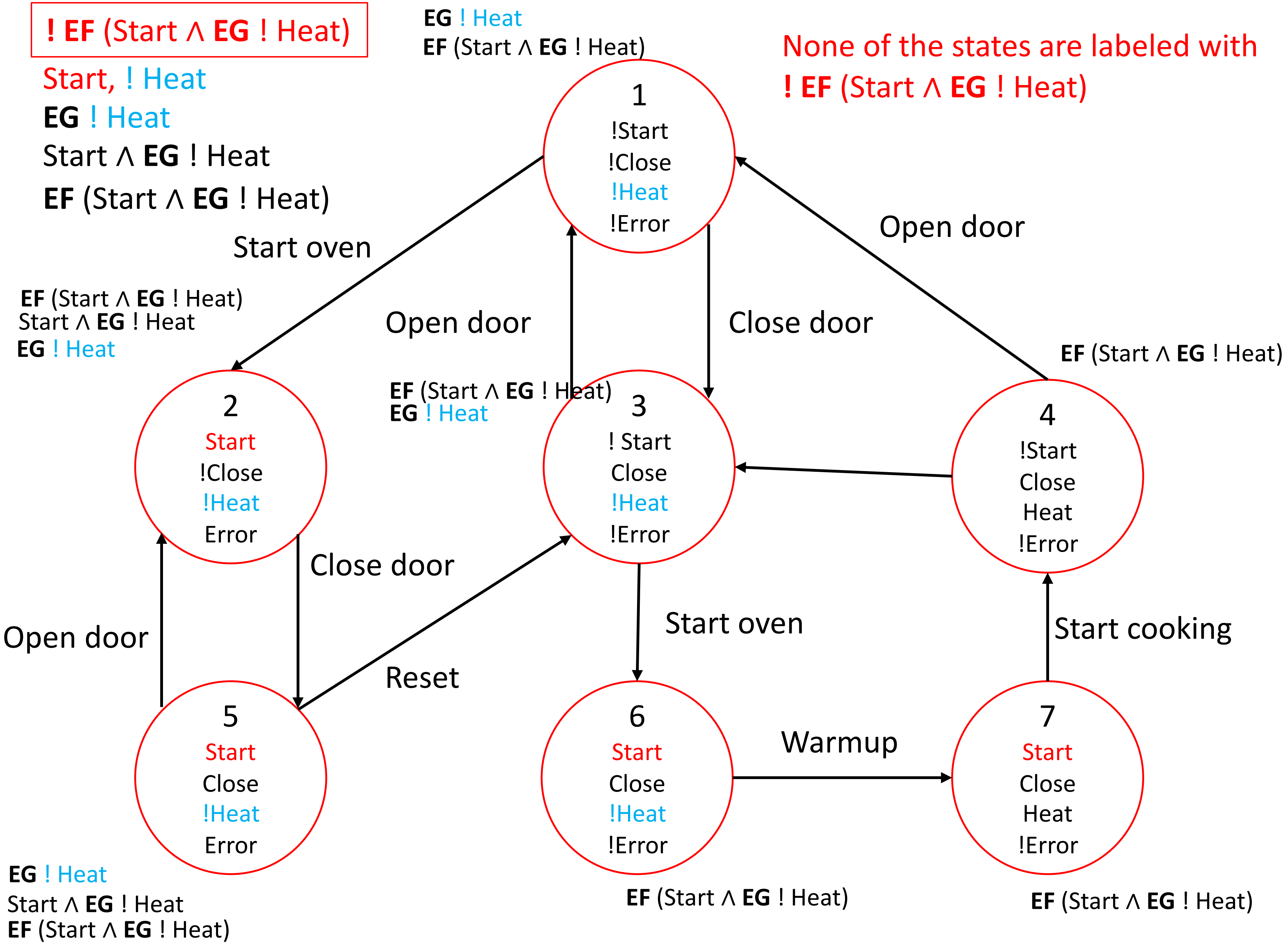


! EF (Start ∧ EG ! Heat)
Start, ! Heat
EG ! Heat
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)
EG ! Heat
1
!Start
!Close
!Heat
!Error
Start oven
Open door
Close door
Open door
Start ∧ EG ! Heat
EG ! Heat
2
Start
!Close
!Heat
Error
EG ! Heat
3
! Start
Close
!Heat
!Error
4
!Start
Close
Heat
!Error
Close door
Open door
Reset
Start oven
Start cooking
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)
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
Open door
Reset
Start oven
Start cooking
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)
Set of states that can reach
Start ∧ EG ! Heat
EG ! Heat
EF (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
Open door
Reset
Start oven
Start cooking
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)
EG ! Heat
EF (Start ∧ EG ! Heat)
None of the states are labeled with
! EF (Start ∧ EG ! Heat)
1
!Start
!Close
!Heat
!Error
Start oven
Open door
Open door
Close 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
Open door
Reset
Start oven
Start cooking
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)