Lecture 19: Computation tree logic
CTL Model Checking

Huan Zhang

huan@huan-zhang.com

Slides adapted from Prof. Sayan Mitra's slides in Fall 2021

Outline

- Temporal logics
    - Computational Tree Logic (CTL)
- CTL model checking for automata
    - 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

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”, “oven is hot”

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

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

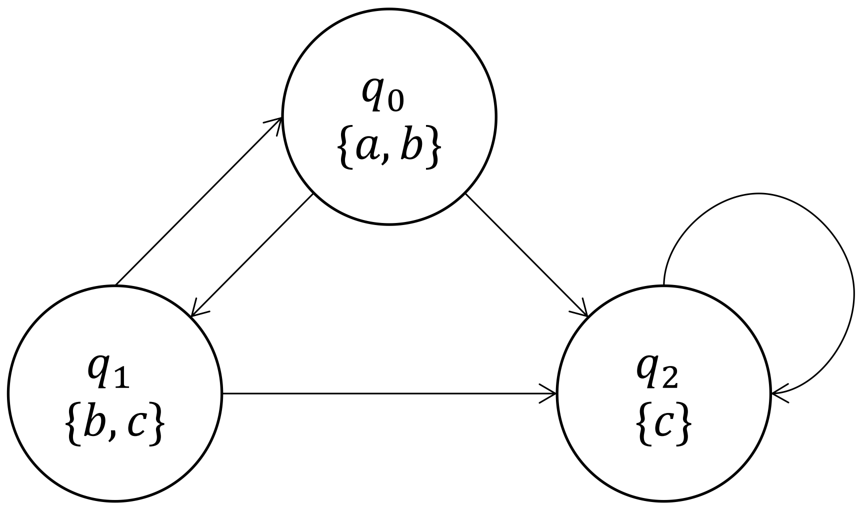
Notations

Automata with state labels but no action labels (“Kripke structure”)

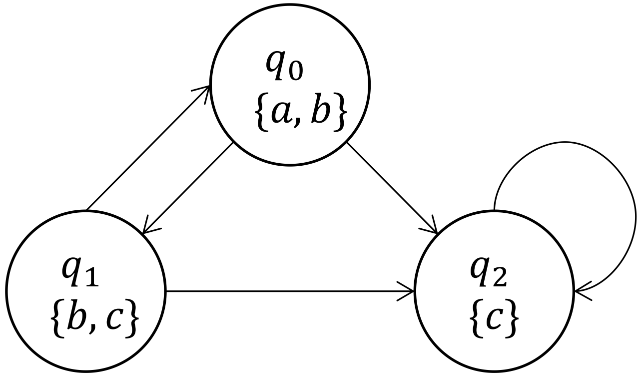
$\mathcal{A} = \langle Q, Q_0, T, L \rangle$

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

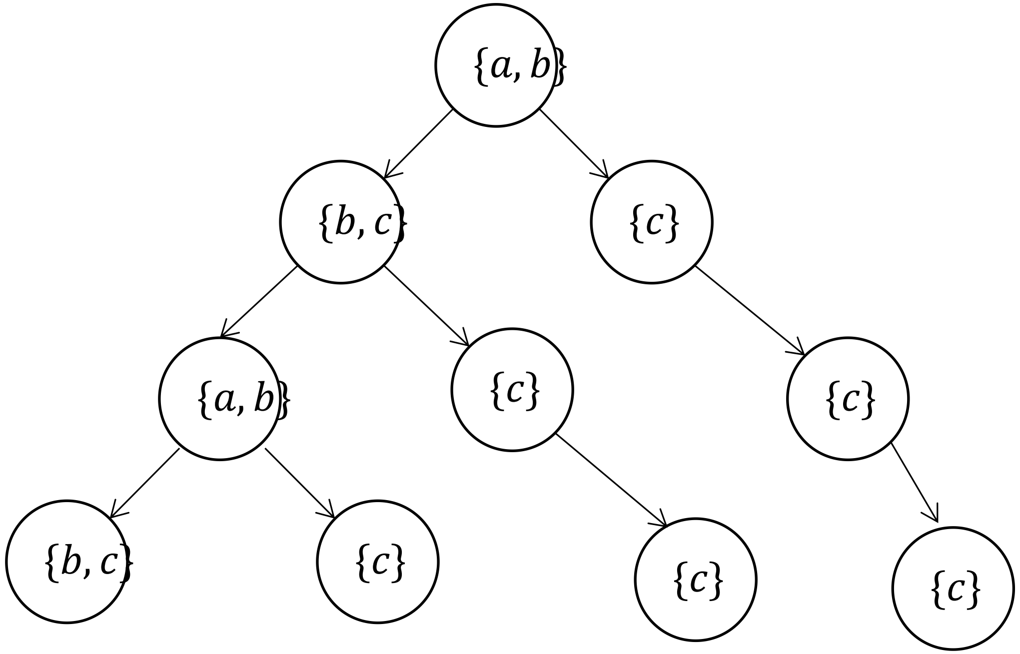
$L(q_0) = \{a, b\}$



Computational tree logic (CTL)



Unfolding the automaton



We get a tree, representing all possible compuations

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 ("p U q" means "p holds until q holds")
F: Eventually (some time in future)
G: Globally (always)

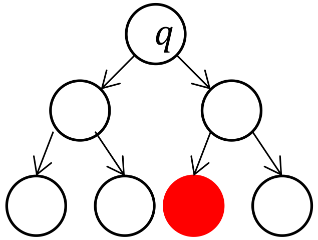
Visualizing CTL semantics

Path quantifiers

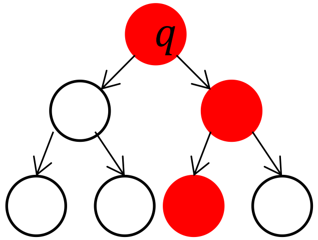
E: Exists some path
A: All paths

Temporal operators

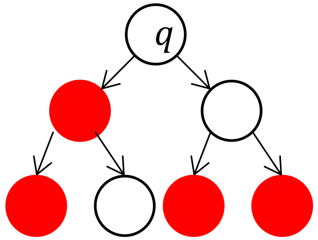
X: Next state
U: Until
F: Eventually
G: Globally (Always)



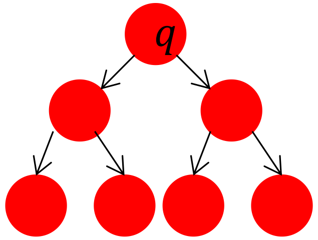
$q \vDash EF\, red$



$q \vDash EG\, red$



$q \vDash AF\, red$



$q \vDash AG\, red$

Slides adapted from Prof. Sayan Mitra's slides in Fall 2021

Visualizing CTL semantics

Path quantifiers

E: Exists some path

A: All paths

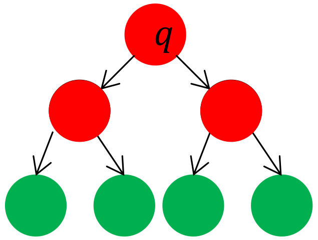
Temporal operators

X: Next state

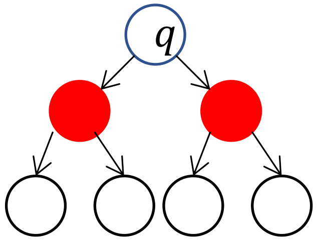
U: Until

F: Eventually

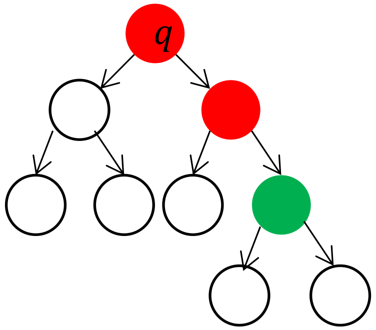
G: Globally (Always)



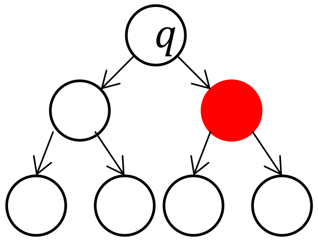
$$q \models \; A\,[red\; U\; green]$$



$$q \models AX\, red$$



$$q \models E\,[\,red\; U\; green]$$



$$q \models EX\, red$$

Slides adapted from Prof. Sayan Mitra's slides in Fall 2021

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

Examples:

everything in the previous two slides;

AG (p_1 => **AF** p_2)

Non-examples

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

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

E$X\ a$; (depth 3)

AX EX a; (depth 5)

AG AF green; (depth 5)

AF AG single token (depth 5)

CTL semantics

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

CTL formula ϕ

For a state q, $q \vDash \phi$ denotes that q *satisfies* ϕ

For a execution α, $\alpha \vDash \phi$ denotes that path (execution) α satisfies ϕ

CTL semantics (cont.)

Here $\vDash$ is defined inductively as:

Example: $q_1 \vDash$ **AG** (Start => **AF** Heat)

$q \vDash p \quad \Leftrightarrow p \in L(q)$, for $p \in AP$

$q \vDash \neg f_1 \quad \Leftrightarrow q \nvDash f_1$

$q \vDash f_1 \wedge f_2 \quad \Leftrightarrow q \vDash f_1 \wedge q \vDash f_2$

$q \vDash E\phi \quad \Leftrightarrow \exists\, \alpha,\ \alpha.fstate = q,\ \alpha \vDash \phi$

$q \vDash A\phi \quad \Leftrightarrow \forall\, \alpha,\ \alpha.fstate = q,\ \alpha \vDash \phi$

$\alpha \vDash Xf \quad \Leftrightarrow \alpha[1] \vDash f$

$\alpha \vDash f_1\, U\, f_2 \quad \Leftrightarrow \exists i \geq 0,\ \alpha[i] \vDash f_2\ and\ \forall j < i\ \alpha[j] \vDash f_1$

$\alpha \vDash F\, f_1 \quad \Leftrightarrow \exists i \geq 0,\ \alpha[i] \vDash f_1$

$\alpha \vDash G\, f_1 \quad \Leftrightarrow \forall i \geq 0,\ \alpha[i] \vDash f_1$

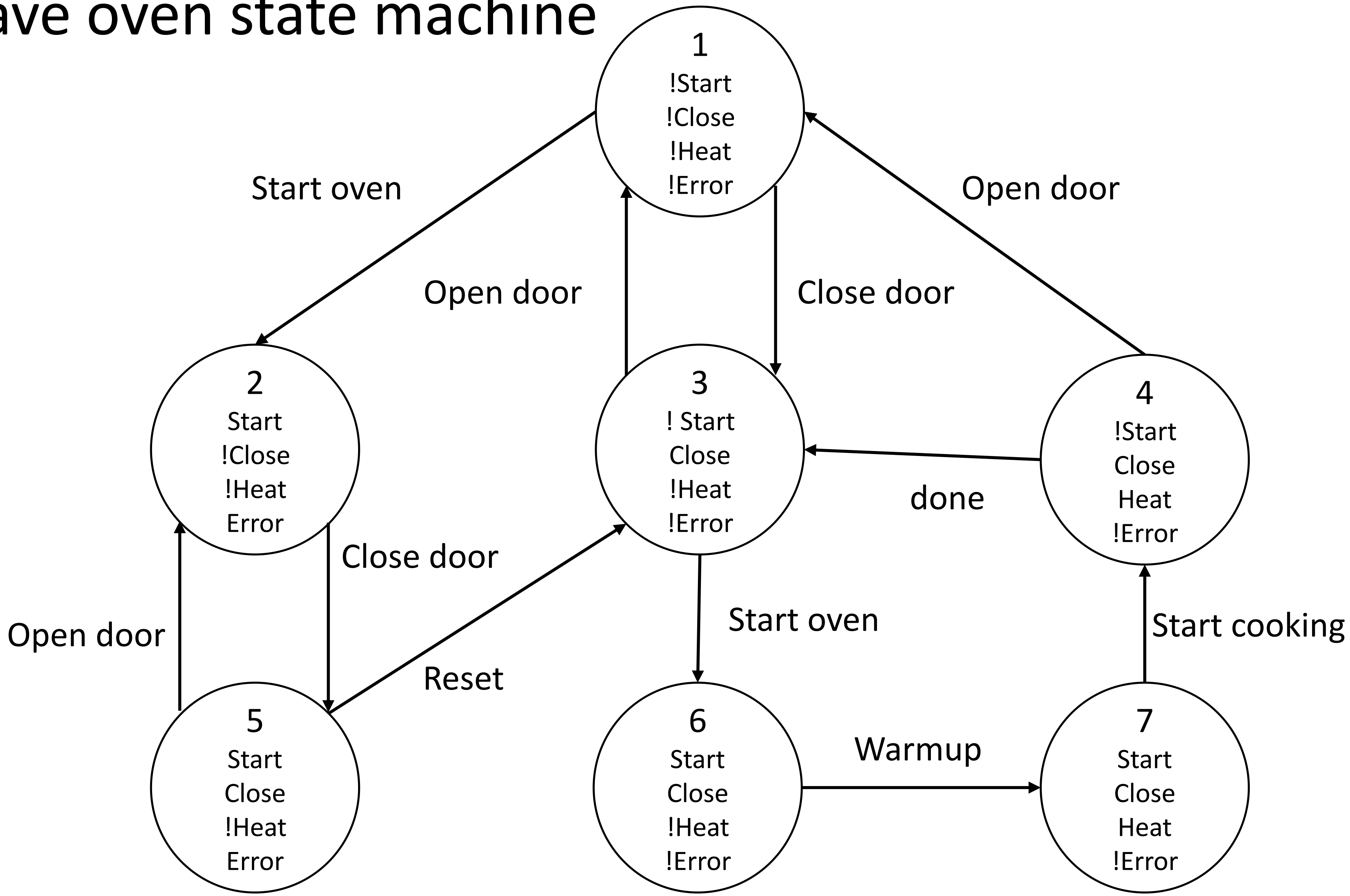
Automaton satisfies property:
$\mathcal{A} \vDash f$ iff $\forall q \in Q_0, q \vDash f$

Universal CTL operators

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

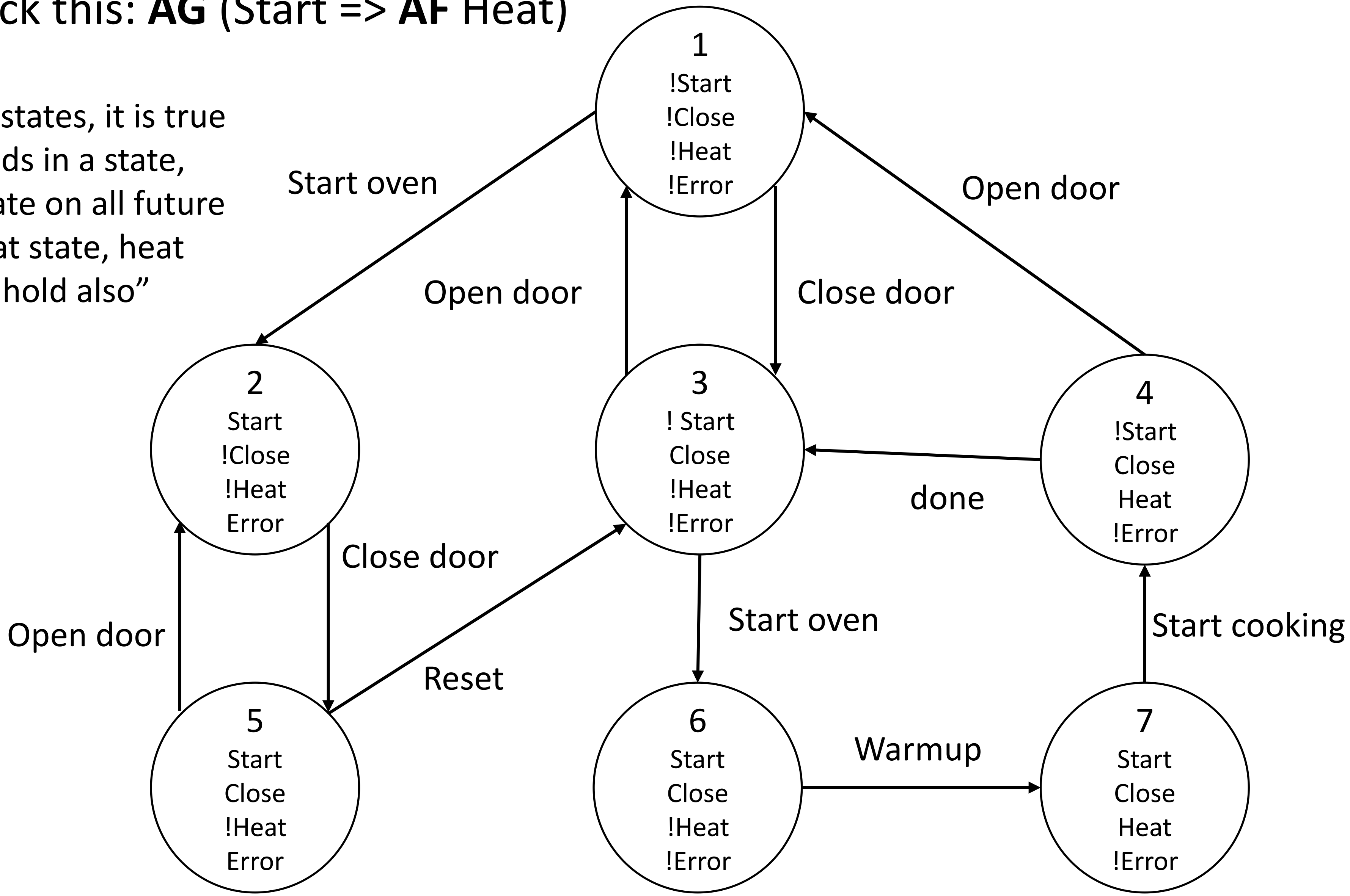
AXf	AGf	AFf	$A[f_1 U f_2]$
$\neg EX(\neg f)$	$\neg EF(\neg f)$	$\neg EG(\neg f)$	$\neg(E[(\neg f_1)U\neg(f_1 \vee f_2)] \vee EG(\neg f_2))$
EX	EG	EF	EU
EX	EG	$E(true\ U\ f)$	EU

microwave oven state machine



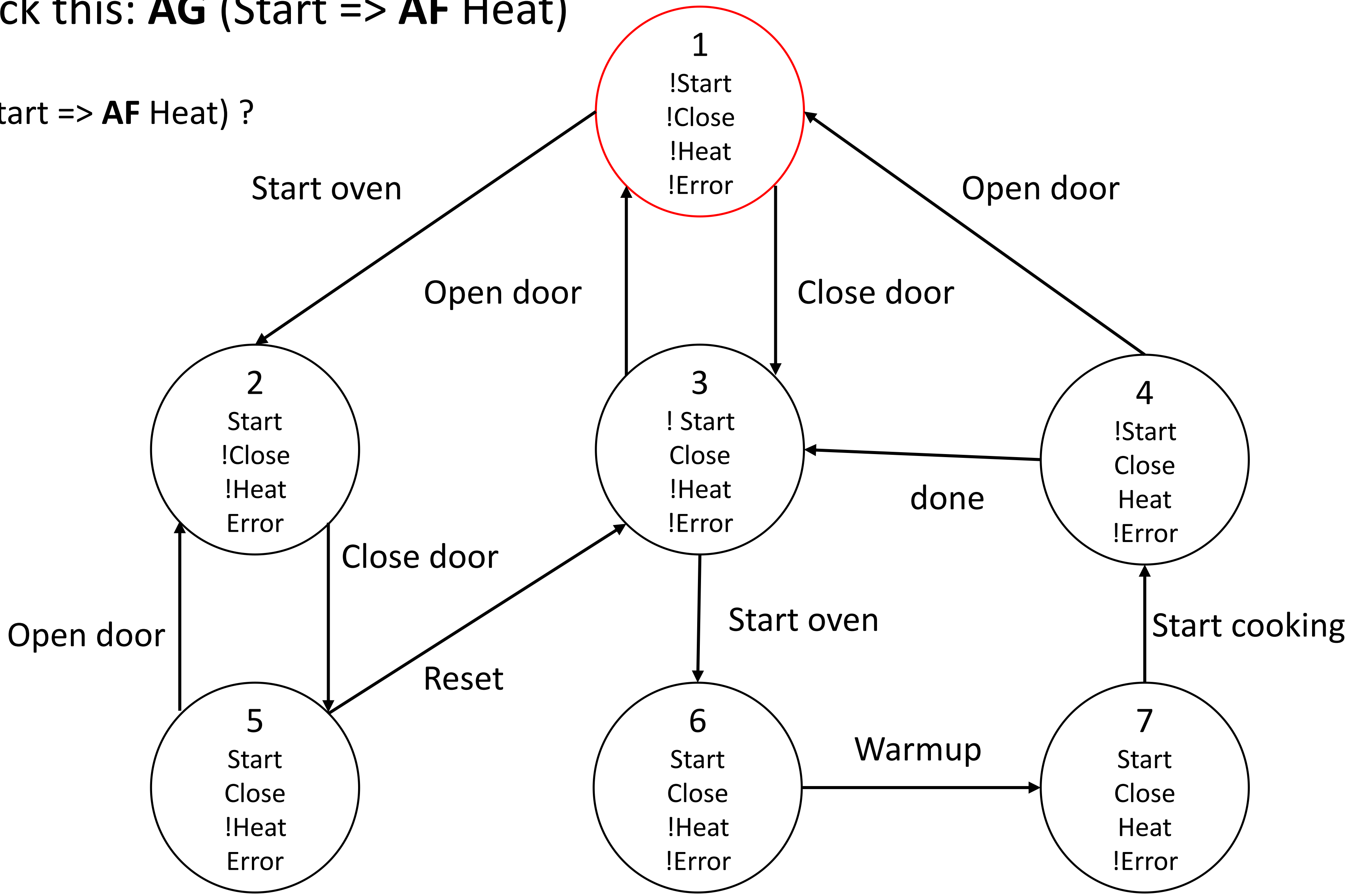
Let's check this: **AG** (Start => **AF** Heat)

English: "In all states, it is true that if start holds in a state, the in some state on all future paths from that state, heat will eventually hold also"



Let's check this: **AG** (Start => **AF** Heat)

$q_1 \vDash$ **AG** (Start => **AF** Heat) ?



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

Algorithm works by structural induction on the depth of the formula

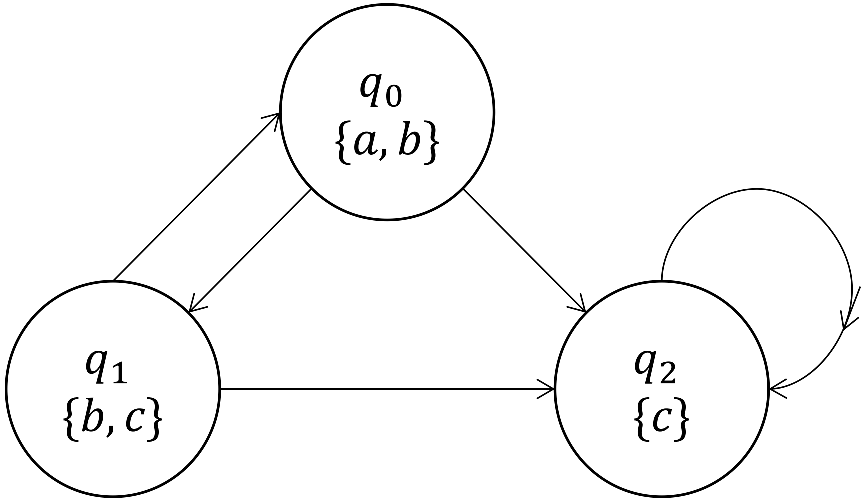
Explicit state model checking

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

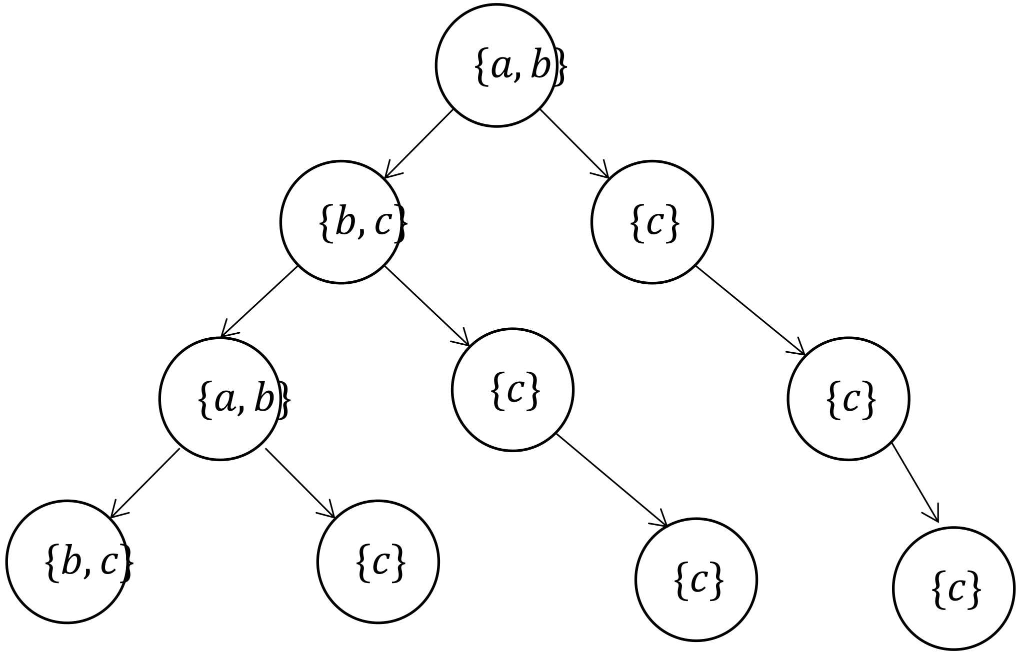
If $Q_0 \subseteq Q'$ then we can conclude $\mathcal{A} \models f$

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

Since all CTL operators can be expressed by EX, EU, EG, we just need to figure out how to check these operators



$\mathcal{A}$



CTL: e.g., whether $\mathcal{A} \vDash$ **AG** (a => **AF** b)

Induction on depth of formula

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

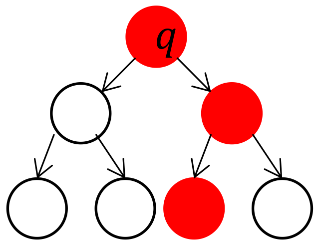
- Initially, $label(q) = L(q)$ for each $q \in$ Q
- At i^{th} iteration $label(q)$ contains all sub-formulas of f of depth $(i - 1)$ that q satisfies

At termination $f \in label(q) \Leftrightarrow q \vDash f$

CheckEG(f_1, Q, T, L)

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

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



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

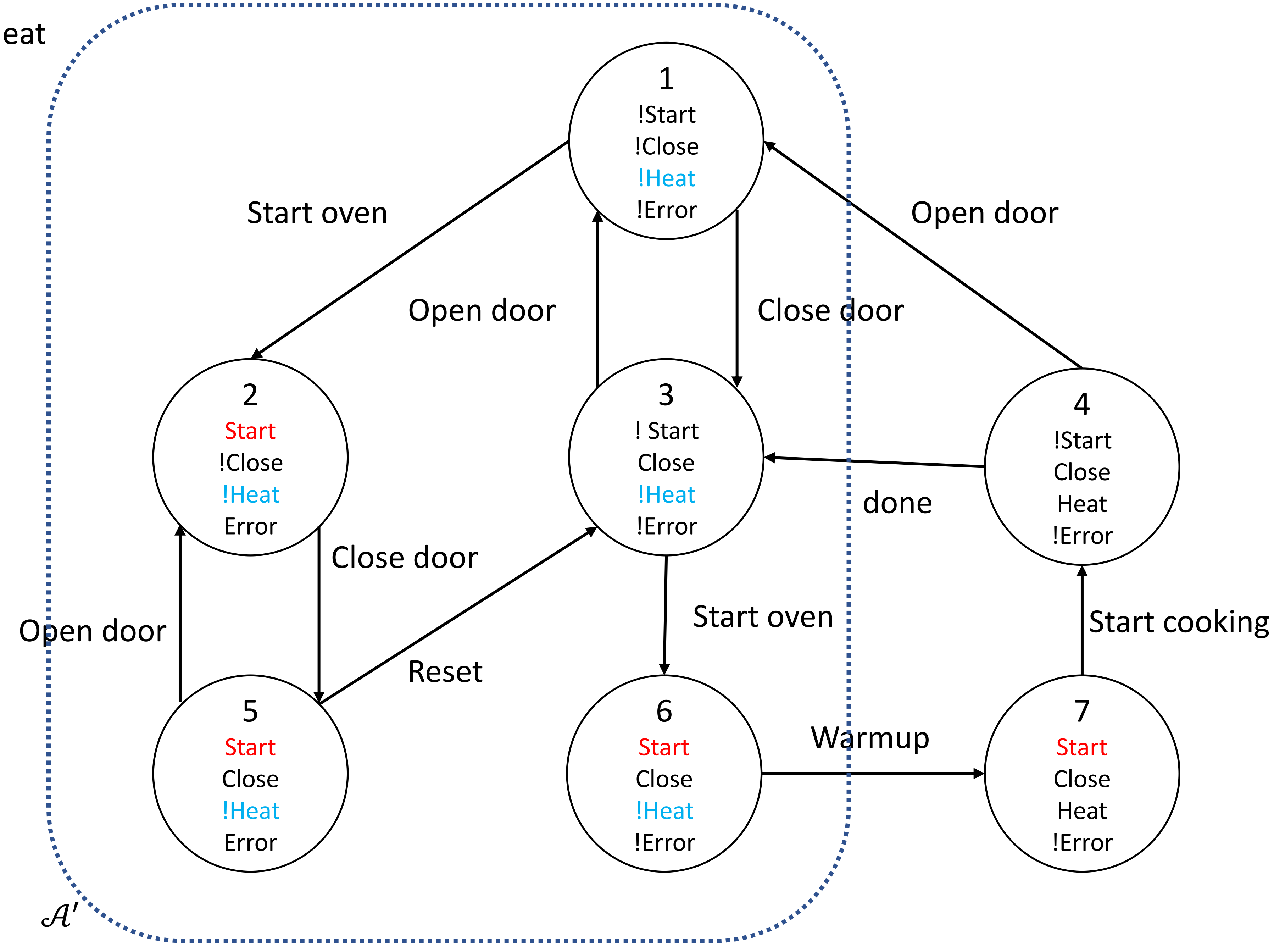
$L': Q' \rightarrow 2^{AP}\ \forall\, q' \in Q',\ L'(q') := L(q')$

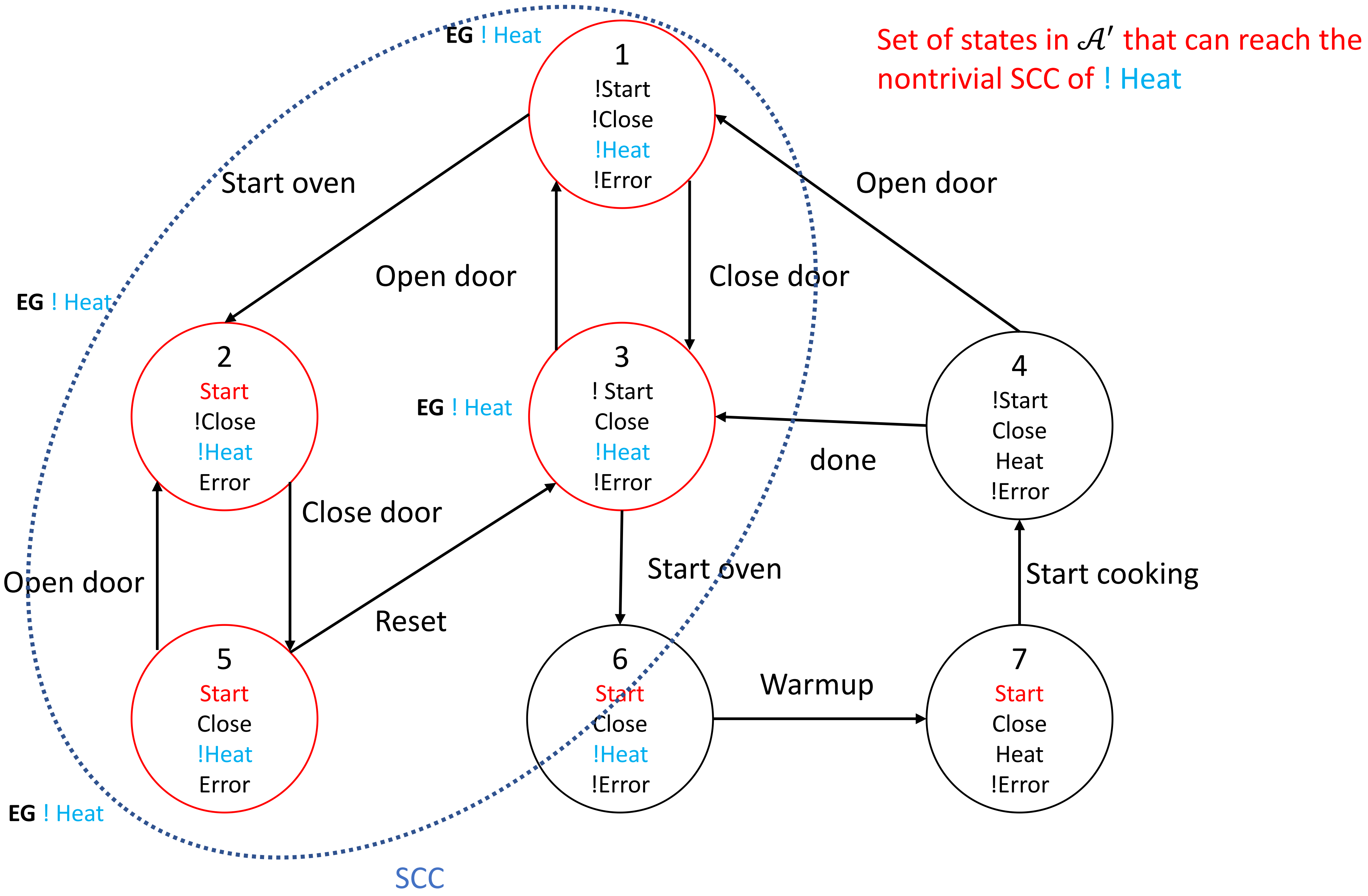
Claim. q $\vDash EGf_1$ iff

(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 $\langle Q', T' \rangle$

Check **EG** ! Heat




Set of states in $\mathcal{A}'$ that can reach the nontrivial SCC of ! Heat
EG ! Heat
1
!Start
!Close
!Heat
!Error
Start oven
Open door
Close door
Open door
EG ! Heat
2
Start
!Close
!Heat
Error
EG ! Heat
3
! Start
Close
!Heat
!Error
4
!Start
Close
Heat
!Error
done
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
SCC

Claim. $\mathcal{A}$, q ⊨ 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 $\mathcal{A}$, q ⊨ 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 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 $Q' = \{q \in Q \mid f_1 \in label(q)\}$ $T' = \{\langle q_1, q_2 \rangle \in T \mid q_1 \in Q'\}$

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 $q \in \boldsymbol{T}$

  $label(q) := label(q) \cup \{EGf_1\}$ Everything already in the SCC satisifies

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

        $label(q') := label(q') \cup \{EGf_1\}$

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

Find all states in Q' that can reach the SCCs

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

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

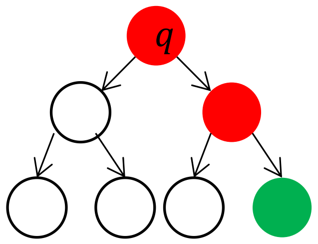
CheckEU(f_1, f_2, Q, T, L)

Let $S = \{q \in Q \mid f_2 \in label(q)\}$

for each $q \in S$

$label(q) := label(q) \cup \{E[f_1 U f_2]\}$

all states where f_2 is true already satsifies



$E[f_1 U f_2]$

while $S \neq \emptyset$

for each $q' \in S$

$S := S \setminus \{q'\}$

for each $q \in T^{-1}(q')$ *Check all states whose next state is q'*

if $f_1 \in label(q)$ then *This if statement will be always true for EF f_2*

$label(q) := label(q) \cup \{E[f_1 U f_2]\}$

$S := S \cup \{q\}$

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.

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 $label(q) \coloneqq label(q) \cup f$

$f = E[f_1 U f_2]$ $\mathrm{CheckEU}(f_1, f_2, Q, T, L)$

$f = EGf_1$ $\mathrm{CheckEG}(f_1, Q, T, L)$

Putting it all together

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

Structural induction over CTL formula

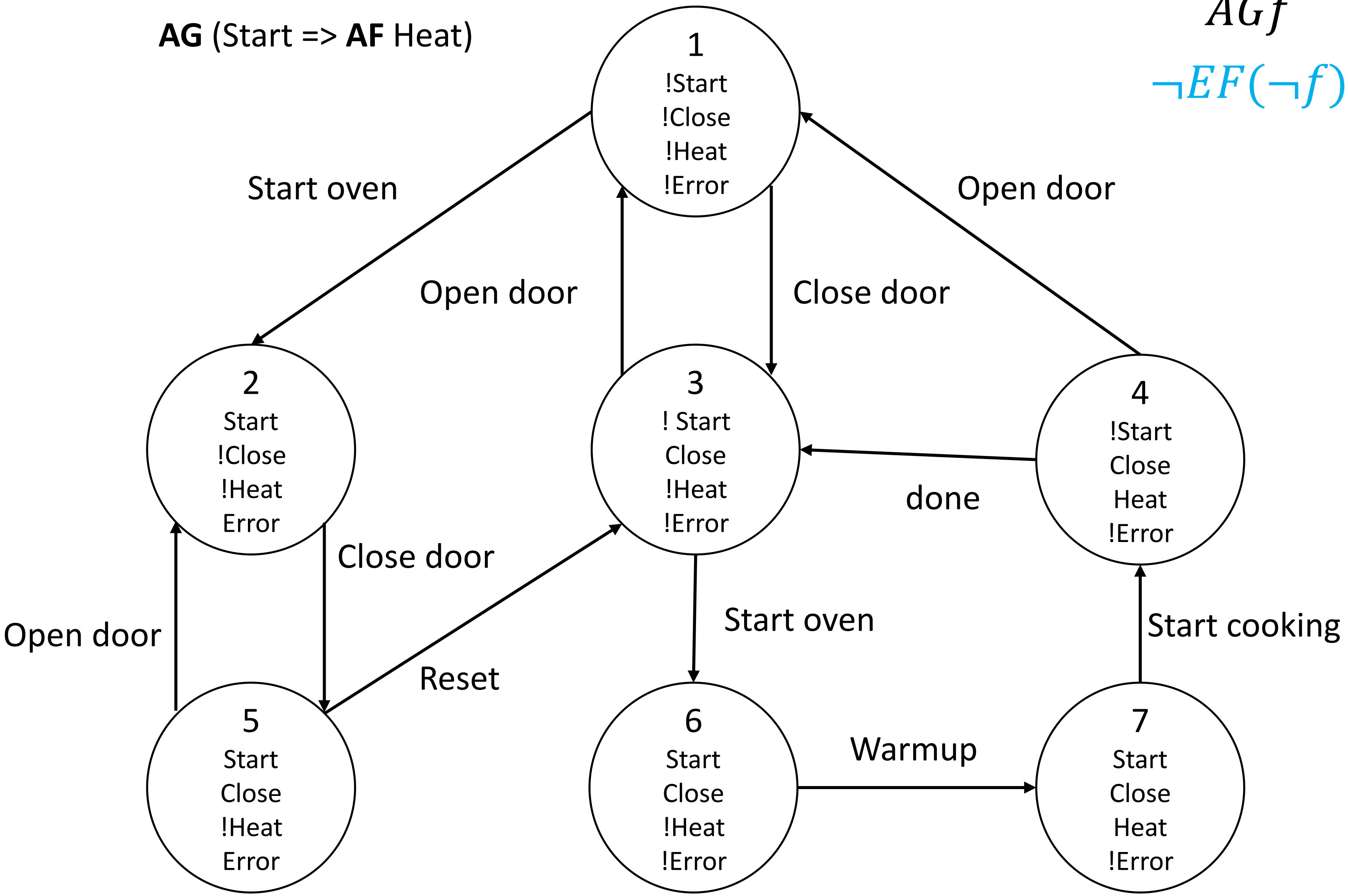
$f = p,$	for some $p \in AP, \forall q,\ label(q) \coloneqq label(q) \cup \{p\}$
$f = \neg f_1$	if $f_1 \notin label(q)$ then $label(q) \coloneqq label(q) \cup f$
$f = f_1 \wedge f_2$	if $f_1,\ f_2 \in label(q)$ then $label(q) \coloneqq label(q) \cup f$
$f = EXf_1$	if $\exists q' \in$ Q such that $(q,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

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

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

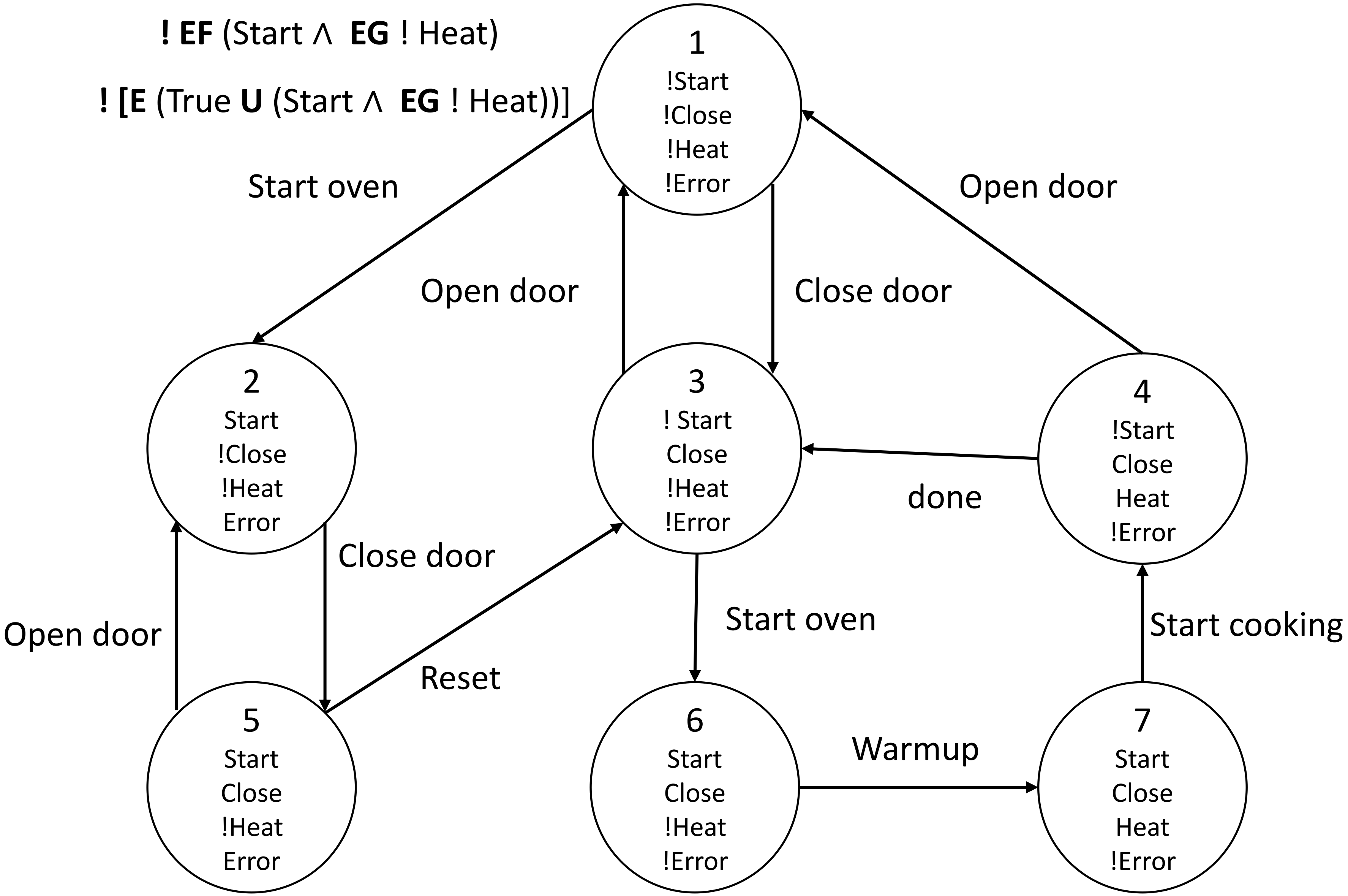
AGf $\quad$ AFf

$\neg EF(\neg f)$ $\quad$ $\neg EG(\neg f)$



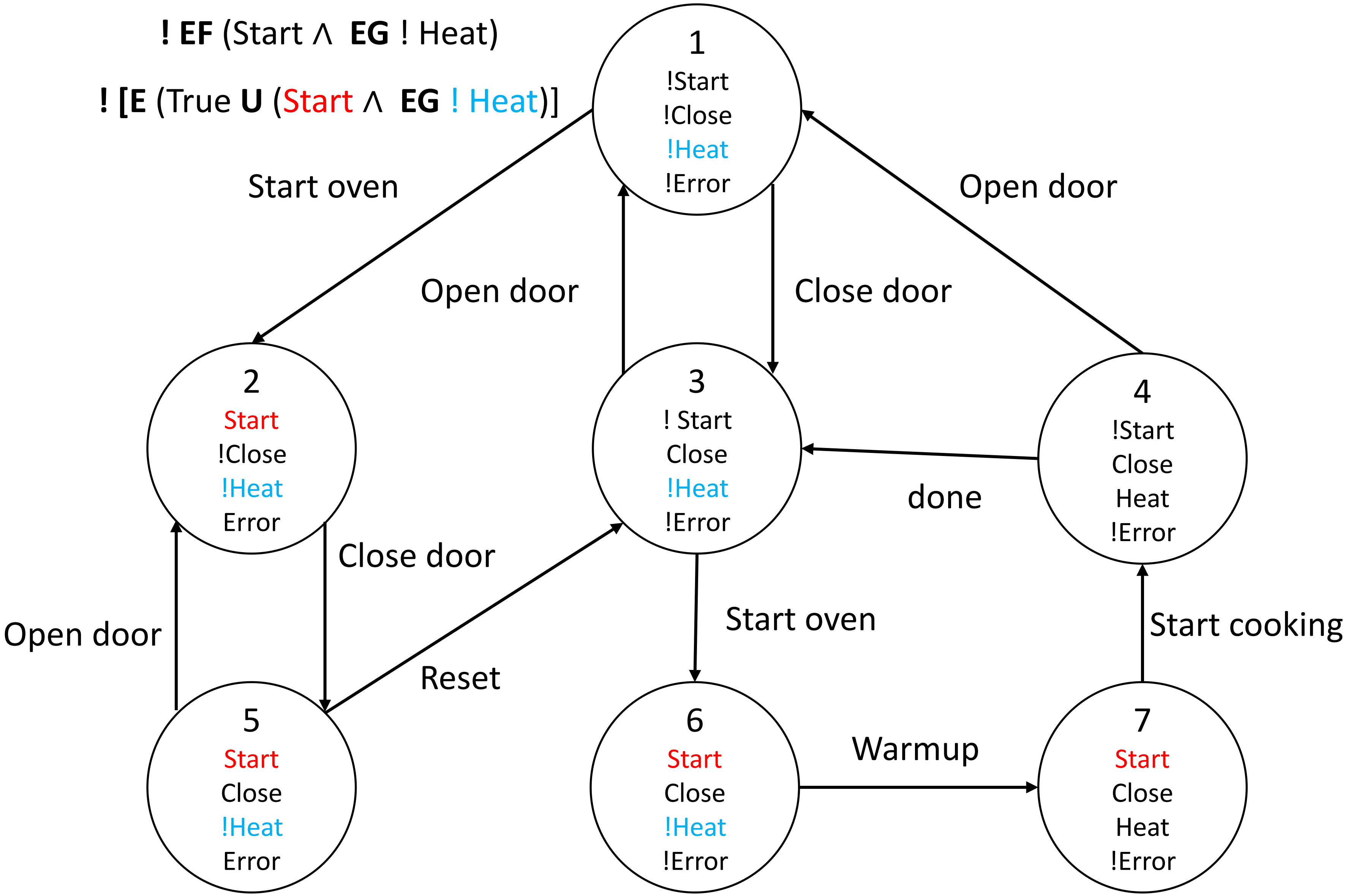
! **EF** (Start ∧ **EG** ! Heat)

! **[E** (True **U** (Start ∧ **EG** ! Heat))]



$$! \mathbf{EF} \, (Start \wedge \mathbf{EG} \, ! \, Heat)$$

$$! \, [\mathbf{E} \, (True \; \mathbf{U} \; (Start \wedge \mathbf{EG} \, ! \, Heat)]$$

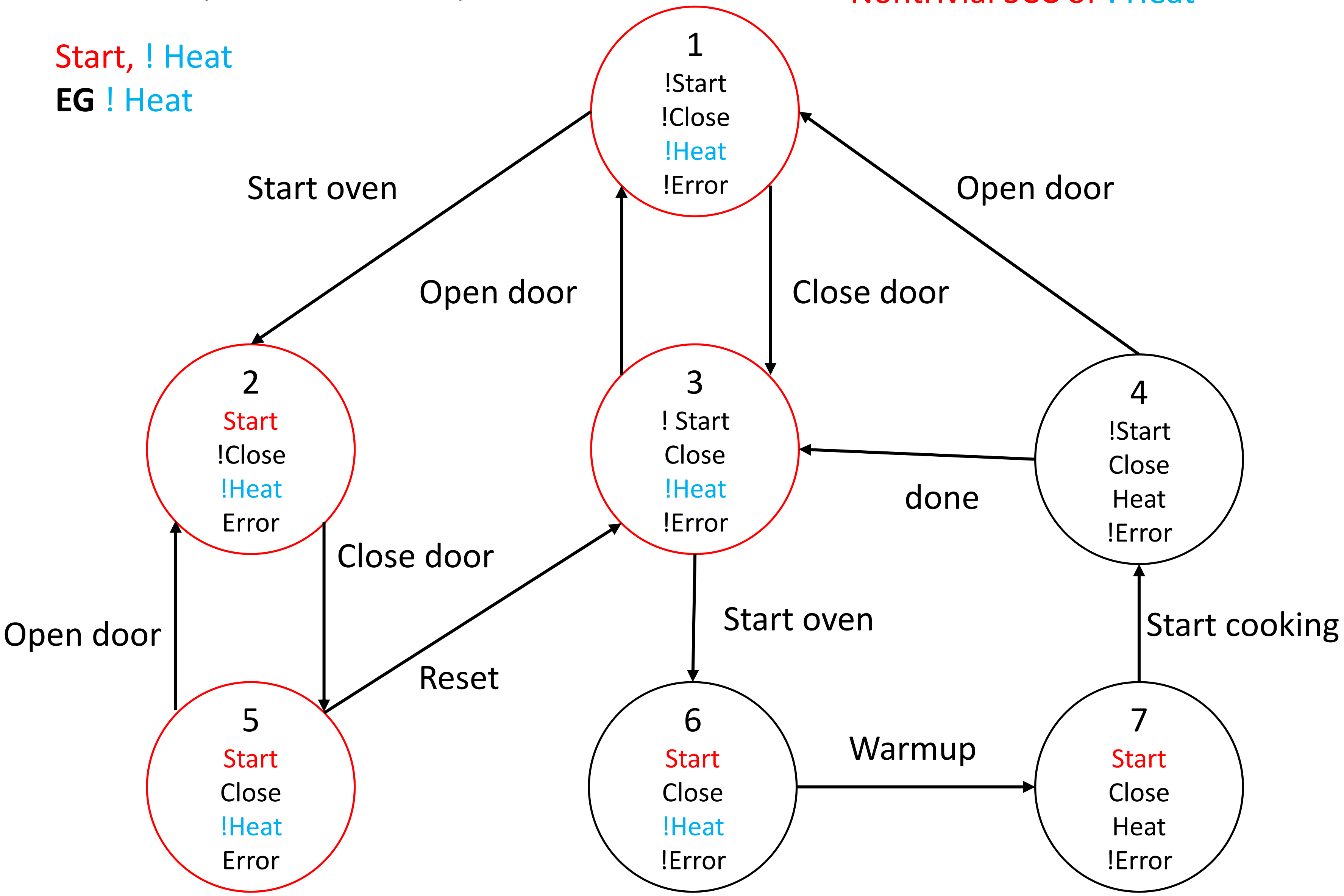


Goal: ! EF (Start ∧ **EG** ! Heat)

Start, ! Heat

EG ! Heat

Nontrivial SCC of ! Heat



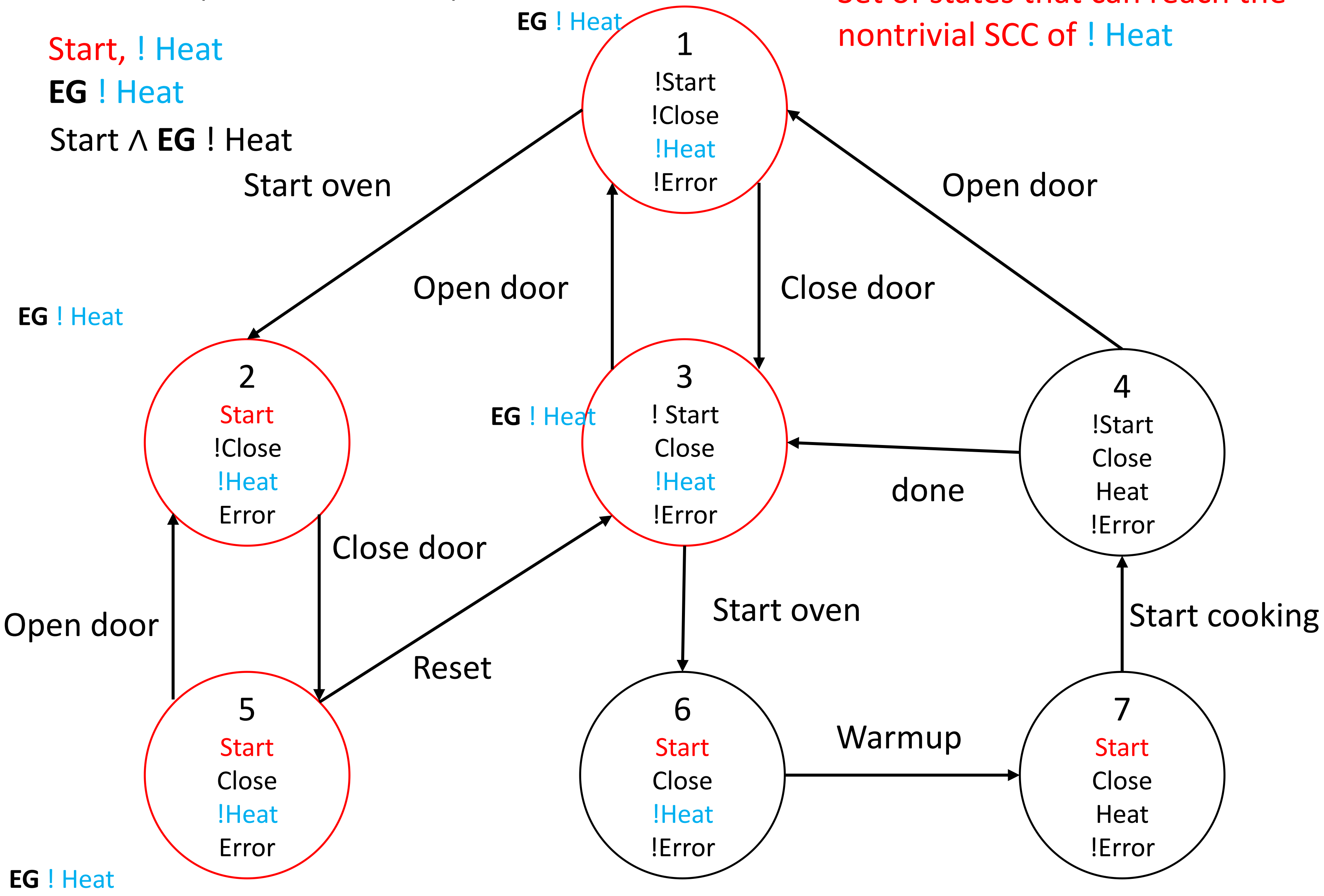
Goal: ! EF (Start ∧ **EG** ! Heat)

Start, ! Heat

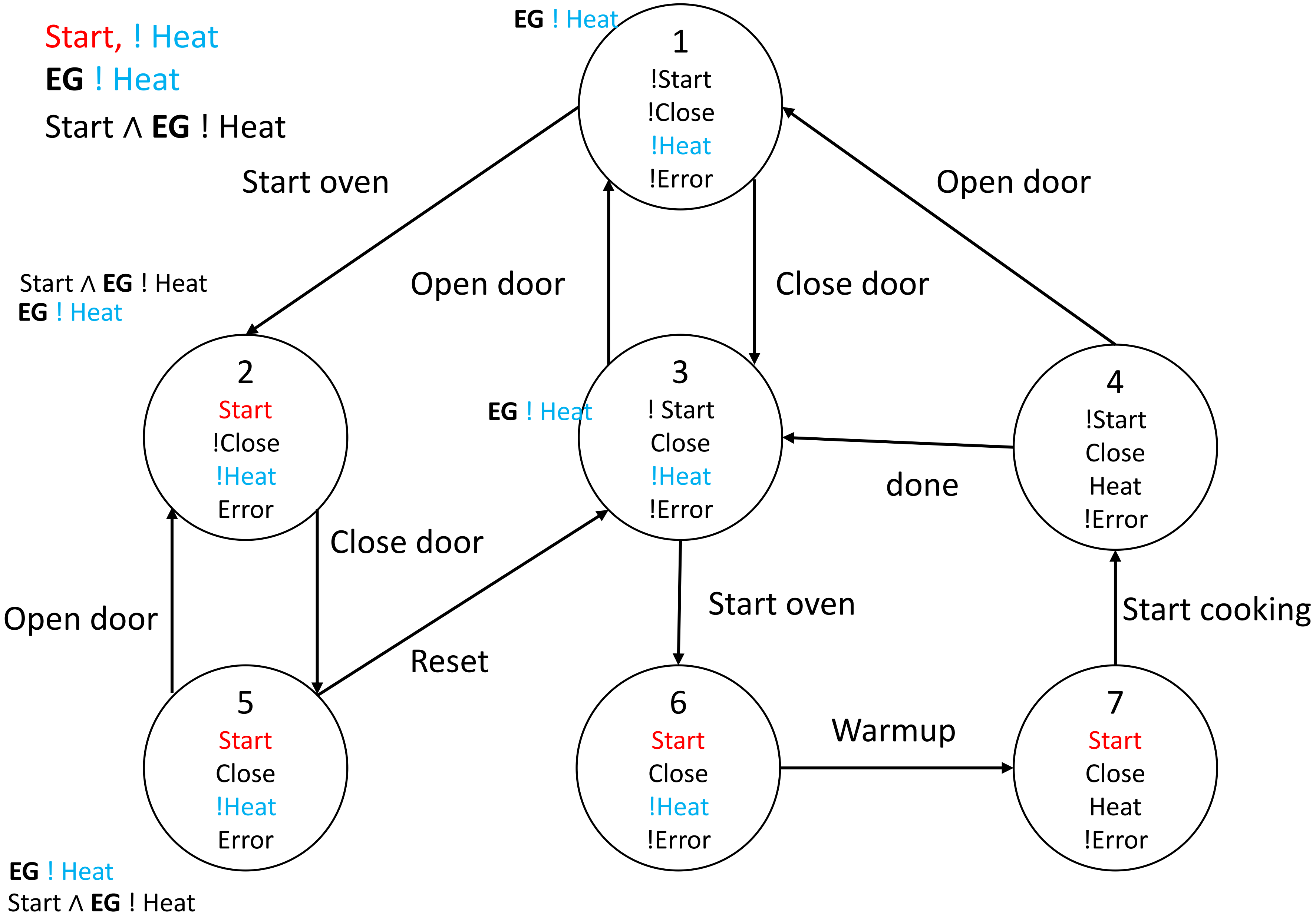
EG ! Heat

Start ∧ **EG** ! Heat

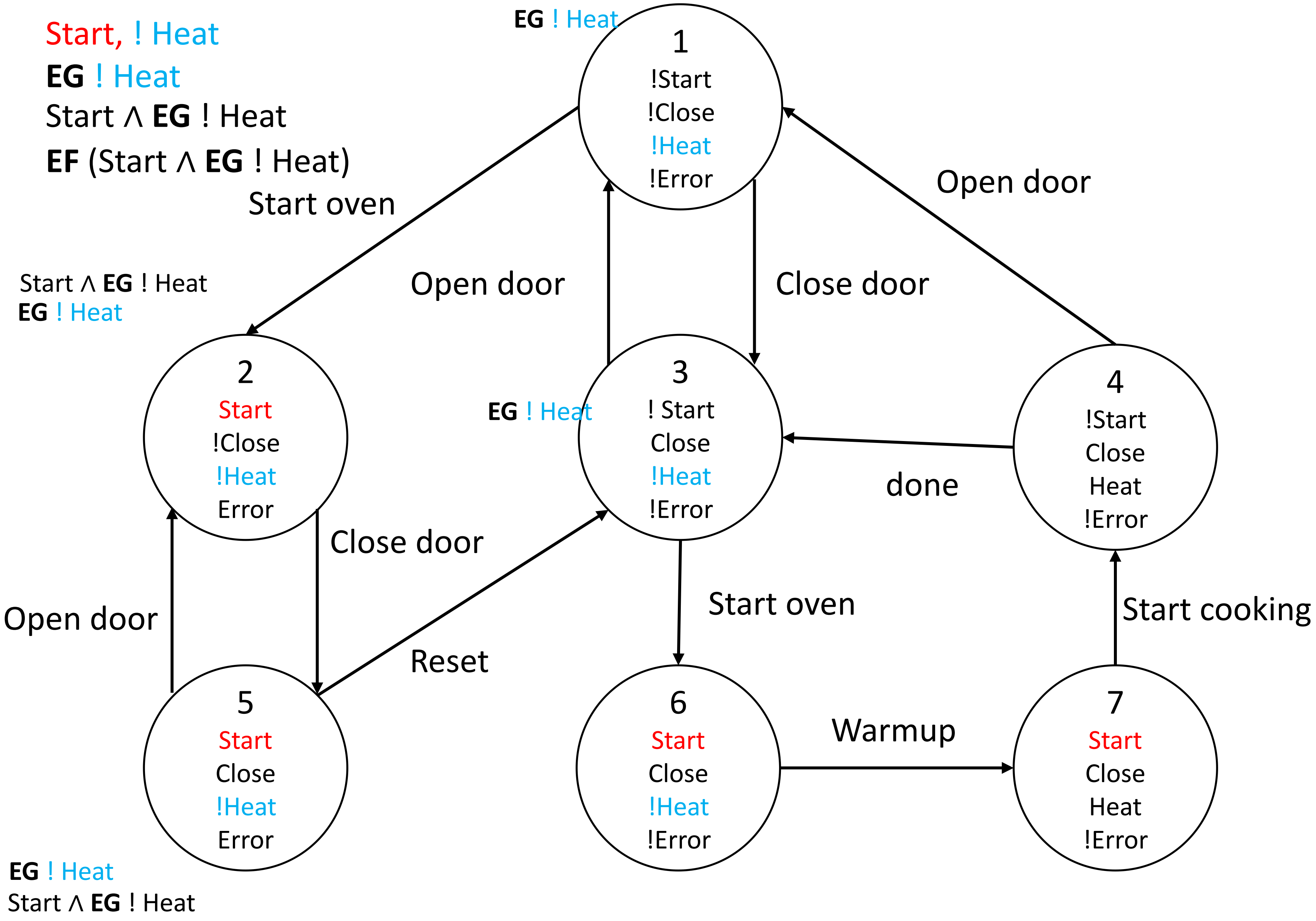
Set of states that can reach the nontrivial SCC of ! Heat



Goal: ! EF (Start ∧ **EG** ! Heat)



Goal: ! EF (Start ∧ **EG** ! Heat)



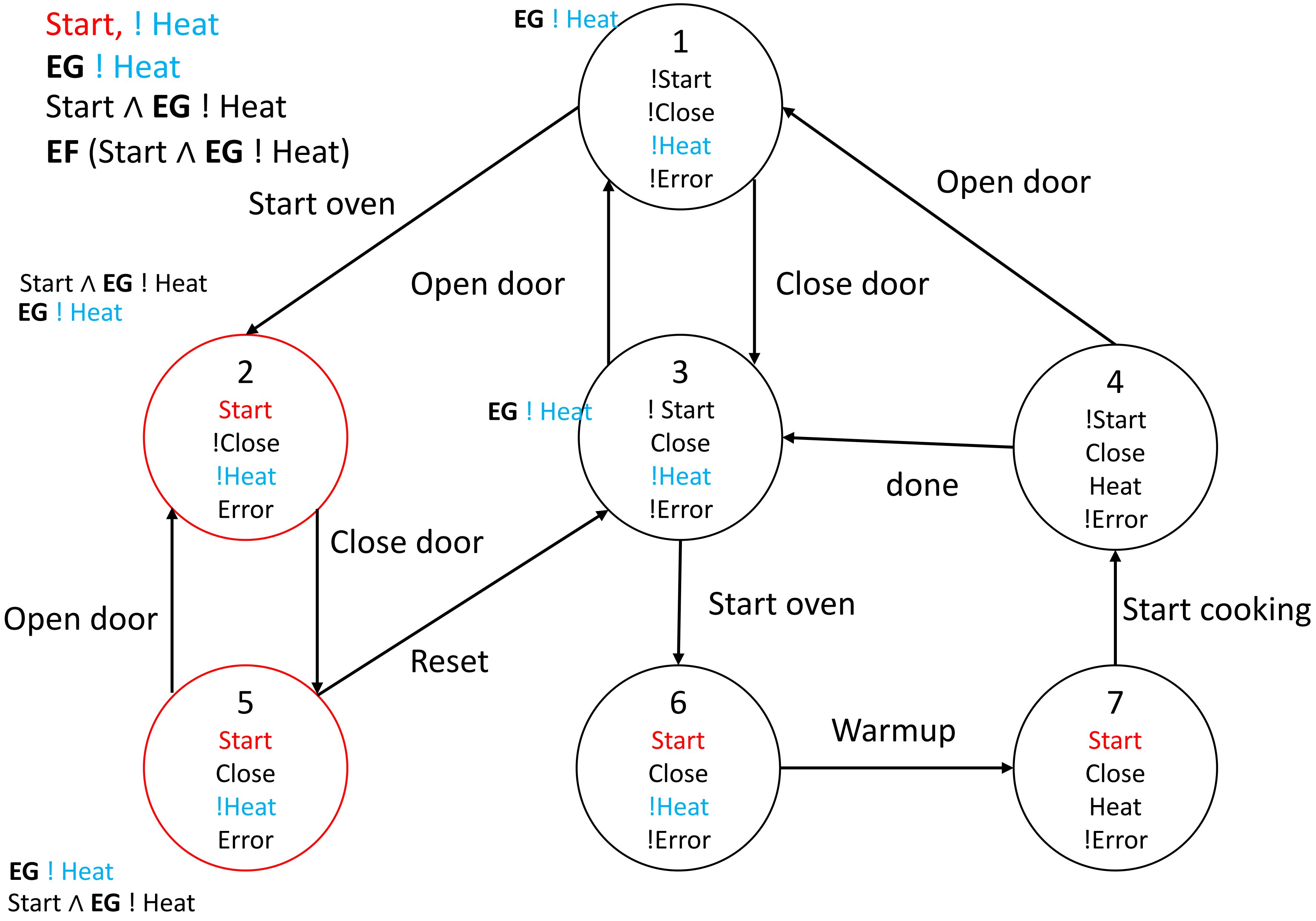
! **EF** (Start ∧ **EG** ! Heat)

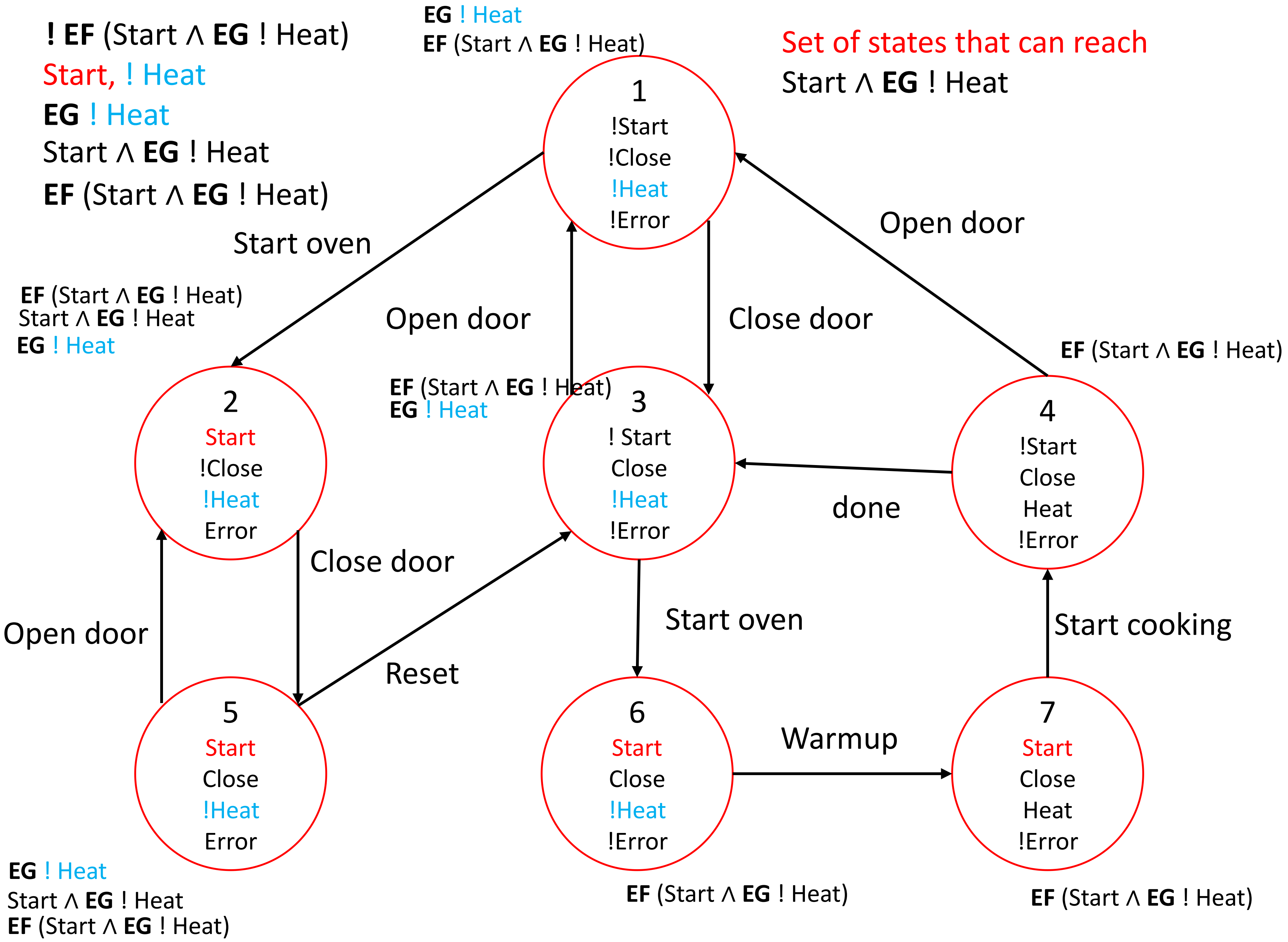
Start, ! Heat

EG ! Heat

Start ∧ **EG** ! Heat

EF (Start ∧ **EG** ! Heat)




! EF (Start ∧ EG ! Heat)
Start, ! Heat
EG ! Heat
Start ∧ EG ! Heat
EF (Start ∧ EG ! Heat)
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
done
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)

None of the states are labeled with
! EF (Start ∧ **EG** ! Heat)
So none of the states ⊨ **AG** (Start => **AF** Heat)
Can you intuitively see why?