CTL model checking

Angelo Montanari (and Massimo Franceschet)

Departement of Mathematics, Computer Science, and Physics
University of Udine

Outline

1. Computation Tree Logic CTL
2. Some example CTL formulas
3. The microwave oven example
4. Model checking for CTL

Computation Tree Logic CTL

CTL is a logic designed to reason about properties holding along **computation paths** of structures.

It extends propositional logic with unary temporal operators **EX**, **AX** and binary temporal operators **EU**, **AU**.

The informal semantics of the temporal operators is as follows:

- **EX**α means "α holds at **some successor**";
- **AX**α means "α holds at **every successor**";
- **EU**(α, β) means "along **some path** α holds until β will hold";
- **AU**(α, β) means " along **every path** α holds until β will hold".

Computation Tree Logic CTL

CTL formulas are defined as follows:

- $\top$ is a CTL-formula;
- a proposition p is a CTL-formula;
- if α and β are CTL-formulas, then $\neg\alpha$ and $\alpha \wedge \beta$ are CTL-formulas;
- if α is a CTL-formula, then $\mathbf{EX}\alpha$ and $\mathbf{AX}\alpha$ are CTL-formulas;
- if α and β are CTL-formulas, then $\mathbf{EU}(\alpha, \beta)$ and $\mathbf{AU}(\alpha, \beta)$ are CTL-formulas.

Computation Tree Logic CTL

CTL formulas are interpreted over **total Kripke structures** $\mathfrak{M} = \langle M, R, V \rangle$. A structure is total if every state has at least one successor.

A **computation path** of $\mathfrak{M}$ is an infinite sequence of states $\pi = s_0, s_1, \ldots$ such that Rs_i, s_{i+1} for every $i \geq 0$. We denote by π_i the ith state s_i of π.

Computation Tree Logic CTL

Given a total Kripke structures $\mathfrak{M} = \langle M, R, V \rangle$ and a state $s \in M$, the semantics of CTL is as follows:

$$\mathfrak{M}, s \models \mathbf{EX}\alpha \quad \text{iff} \quad \exists s'.Rss' \wedge \mathfrak{M}, s' \models \alpha$$

$$\mathfrak{M}, s \models \mathbf{AX}\alpha \quad \text{iff} \quad \forall s'.Rss' \rightarrow \mathfrak{M}, s' \models \alpha$$

$$\mathfrak{M}, s \models \mathbf{EU}(\alpha, \beta) \quad \text{iff} \quad \exists \pi \text{ starting from } s.\ \exists j \geq 0.\ \mathfrak{M}, \pi_j \models \beta \wedge \forall\ 0 \leq i < j.\ \mathfrak{M}, \pi_i \models \alpha$$

$$\mathfrak{M}, s \models \mathbf{AU}(\alpha, \beta) \quad \text{iff} \quad \forall \pi \text{ starting from } s.\ \exists j \geq 0.\ \mathfrak{M}, \pi_j \models \beta \wedge \forall\ 0 \leq i < j.\ \mathfrak{M}, \pi_i \models \alpha$$

Computation Tree Logic CTL

Shorthands are:

- $\mathbf{EF}\alpha = \mathbf{EU}(\top, \alpha)$;
- $\mathbf{AF}\alpha = \mathbf{AU}(\top, \alpha)$;
- $\mathbf{EG}\alpha = \neg\mathbf{AF}\neg\alpha$;
- $\mathbf{AG}\alpha = \neg\mathbf{EF}\neg\alpha$.

Some example CTL and CTL* formulas

- **AG** $(p \rightarrow$ **AF** $q)$;
- **EF** $p \rightarrow$ **EF EG** p;
- **EF** $(p \rightarrow$ **EG** $p)$;
- **AG** $(p \rightarrow$ **EX** $p) \rightarrow$ **AG** $(p \rightarrow$ **EG** $p)$ (a valid CTL formula);
- **AG** $(p \rightarrow$ **EXF** $p) \rightarrow (p \rightarrow$ **EGF** $p)$ (a valid CTL* formula).

CTL vs. LTL

A CTL formula that expresses a property that cannot be expressed in LTL:

- $\mathbf{AG}\ (p \rightarrow \mathbf{EF}\ q)$

An LTL formula that expresses a property that cannot be expressed in CTL:

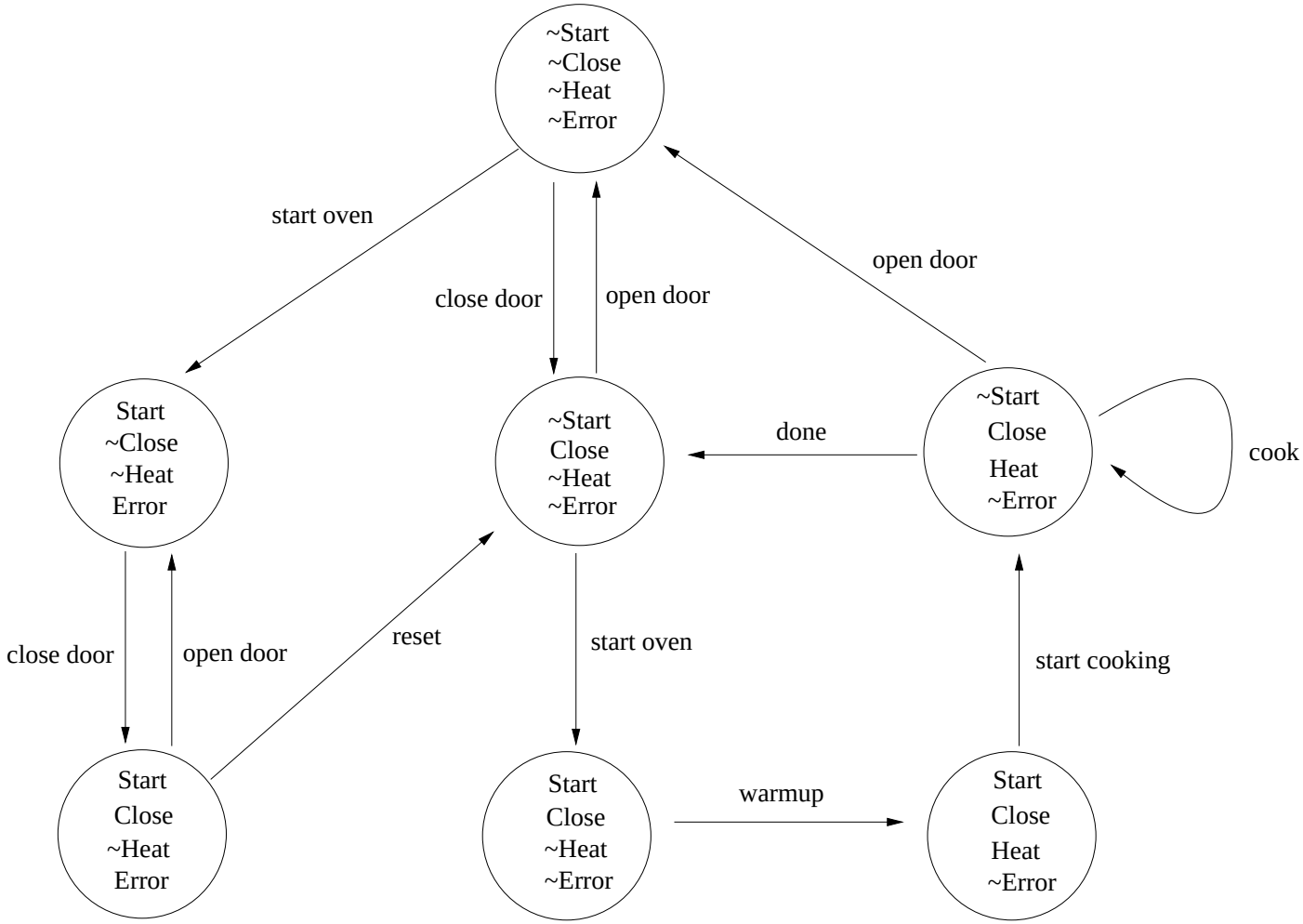
- $\mathbf{G}\ (p \rightarrow \mathbf{FG}\ p)$

The microwave oven example

Informal specification of a microwave oven functioning:

> To cook food in the oven, open the door, put the food inside, and close the door. Do not put metal containers in the oven. Press the start button. The oven will warmup for 30 seconds, and then it will start cooking. When the cooking is done, the oven will stop. The oven will stop also whenever the door is opened during cooking. If the oven is started while the door is open, an error will occur, and the oven will not heat. In such a case, the reset button may be used.

Modelling


~Start
~Close
~Heat
~Error
start oven
close door
open door
open door
Start
~Close
~Heat
Error
~Start
Close
~Heat
~Error
done
~Start
Close
Heat
~Error
cook
close door
open door
reset
start oven
start cooking
Start
Close
~Heat
Error
Start
Close
~Heat
~Error
warmup
Start
Close
Heat
~Error

Specification

1. If the oven heats, then the door is closed:

$$\mathbf{AG}(\texttt{Heat} \rightarrow \texttt{Close})$$

Specification

1. If the oven heats, then the door is closed:

$$\mathbf{AG}(\texttt{Heat} \rightarrow \texttt{Close})$$

2. Whenever the start button is pushed, eventually the oven will heat:

$$\mathbf{AG}(\texttt{Start} \rightarrow \mathbf{AF}\texttt{Heat})$$

Specification

1. If the oven heats, then the door is closed:

$$\mathbf{AG}(\texttt{Heat} \rightarrow \texttt{Close})$$

2. Whenever the start button is pushed, eventually the oven will heat:

$$\mathbf{AG}(\texttt{Start} \rightarrow \mathbf{AF}\texttt{Heat})$$

3. Whenever the oven is correctly started, eventually the oven will heat:

$$\mathbf{AG}((\texttt{Start} \wedge \neg\texttt{Error}) \rightarrow \mathbf{AF}\texttt{Heat})$$

Specification

1. If the oven heats, then the door is closed:

$$\mathbf{AG}(\texttt{Heat} \rightarrow \texttt{Close})$$

2. Whenever the start button is pushed, eventually the oven will heat:

$$\mathbf{AG}(\texttt{Start} \rightarrow \mathbf{AF}\texttt{Heat})$$

3. Whenever the oven is correctly started, eventually the oven will heat:

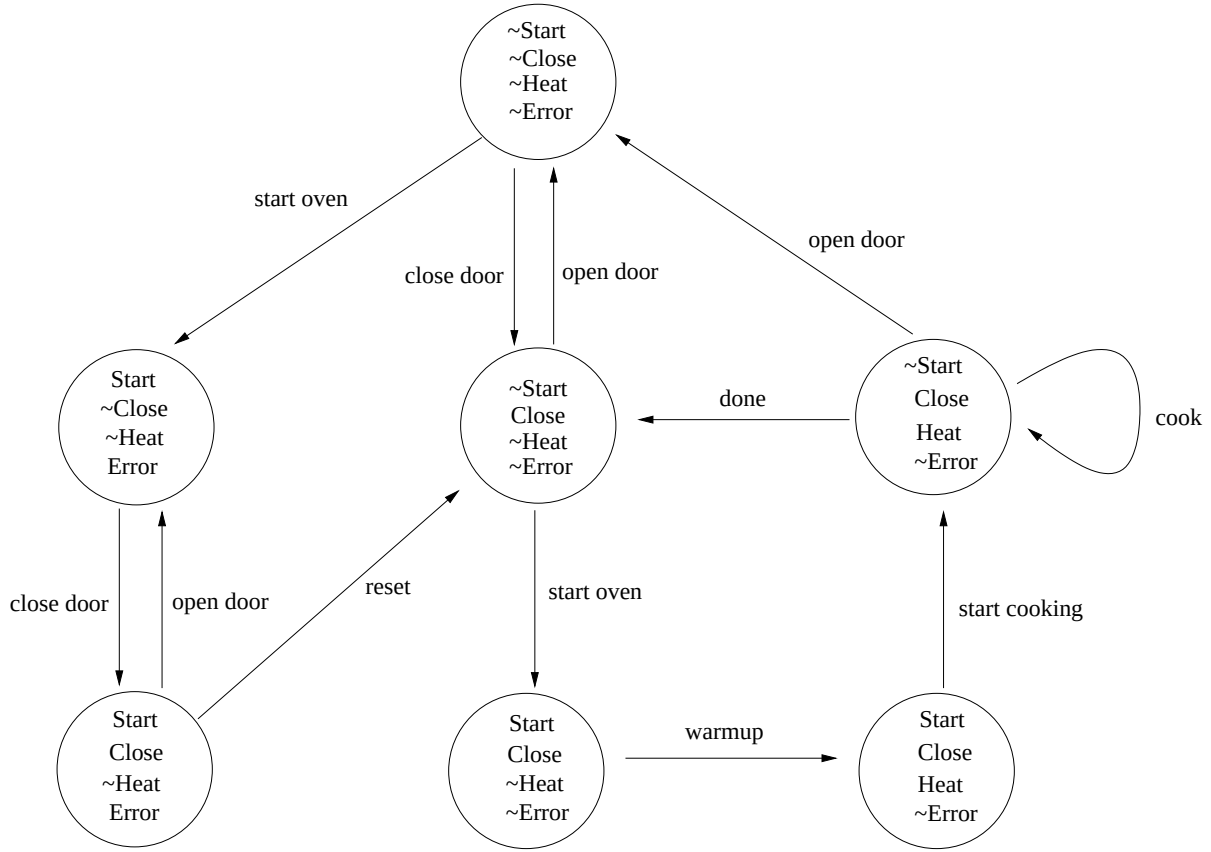
$$\mathbf{AG}((\texttt{Start} \wedge \neg\texttt{Error}) \rightarrow \mathbf{AF}\texttt{Heat})$$

4. Whenever an error occur, it will be still possible to cook:

$$\mathbf{AG}(\texttt{Error} \rightarrow \mathbf{EF}\texttt{Heat})$$

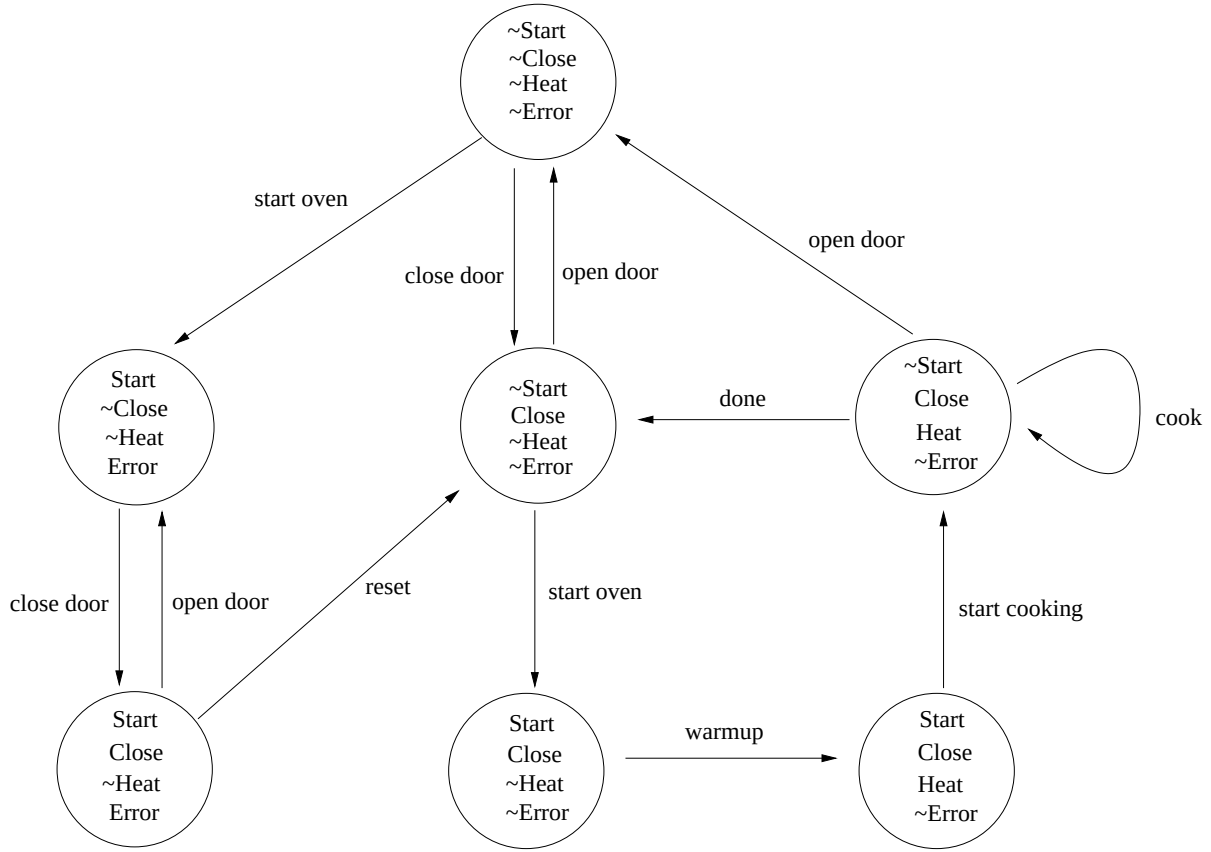
Verification

AG(`Heat` $\rightarrow$ `Close`) is ?



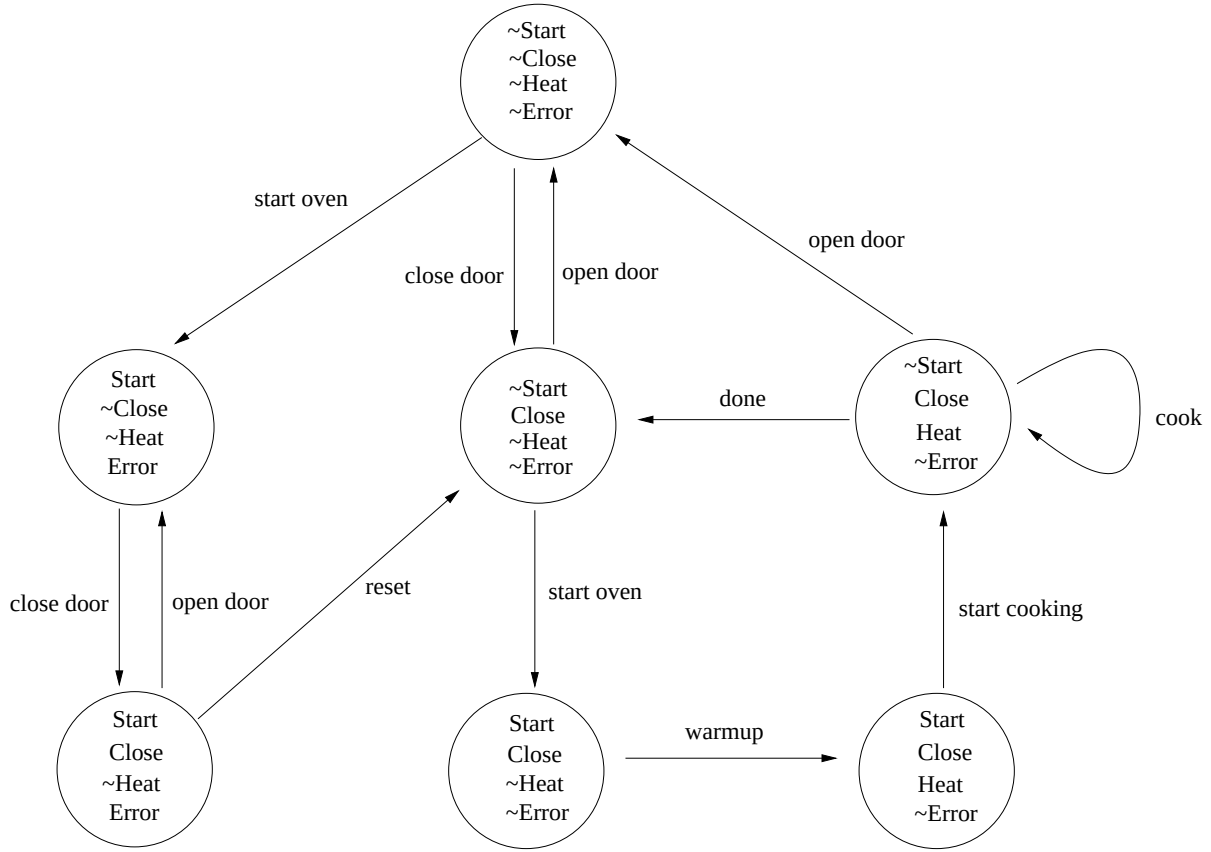
Verification

AG(`Heat` $\rightarrow$ `Close`) is true!



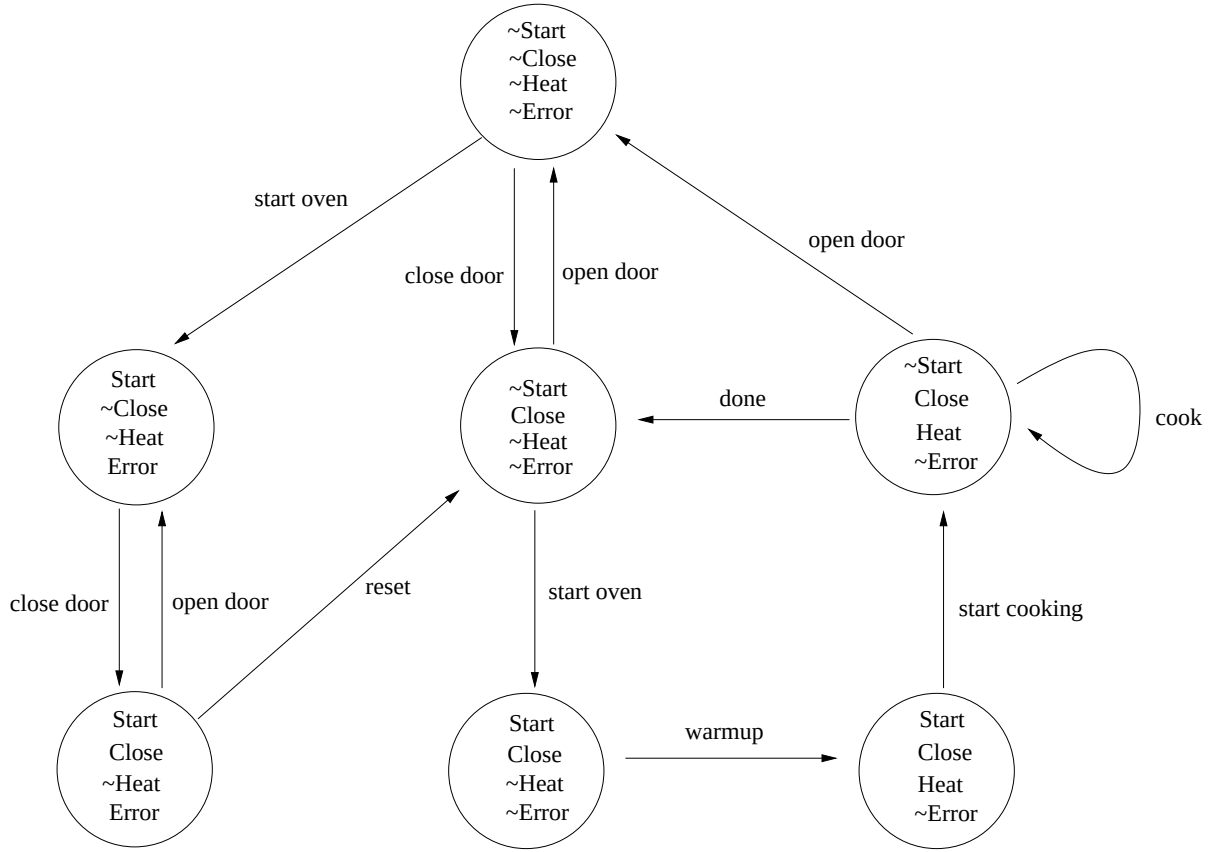
Verification

$\mathbf{AG}(\texttt{Start} \rightarrow \mathbf{AF}\texttt{Heat})$ is ?



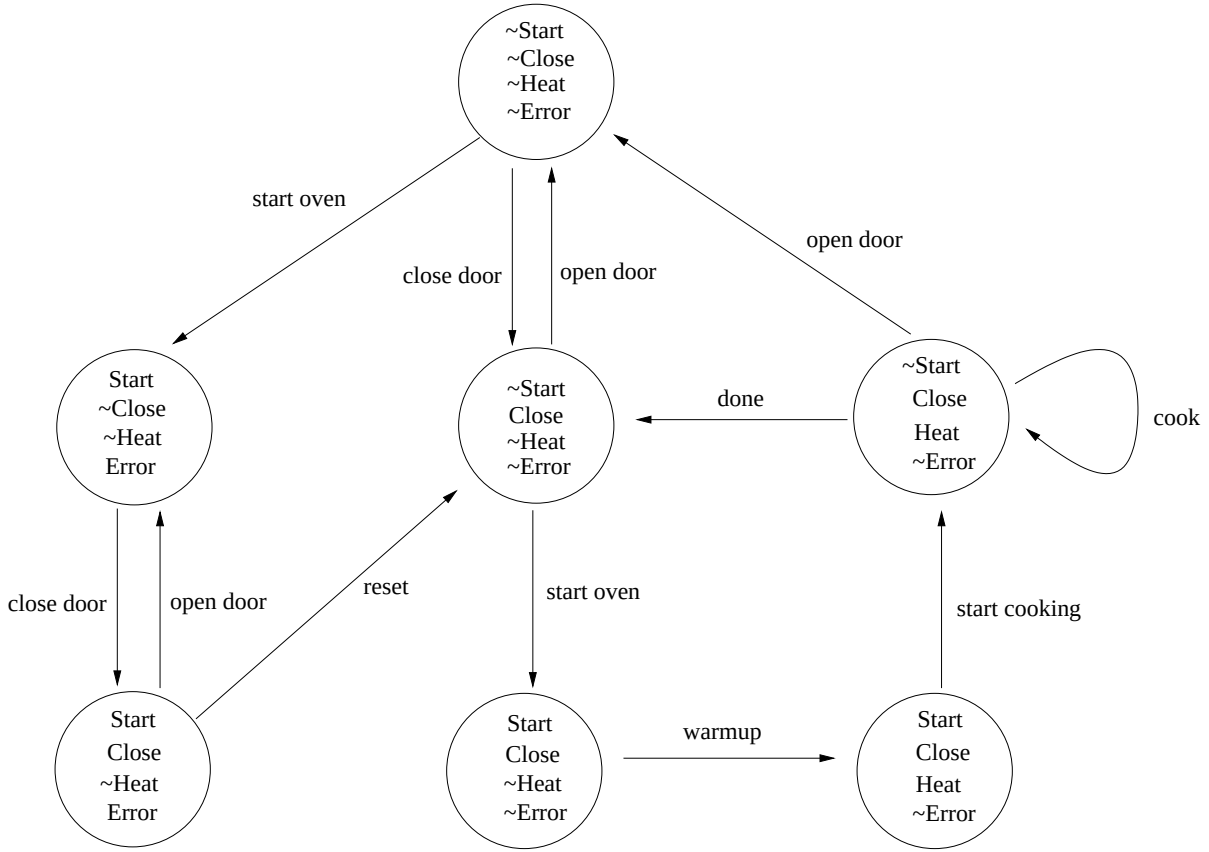
Verification

$\mathbf{AG}(\texttt{Start} \rightarrow \mathbf{AF}\texttt{Heat})$ is false!



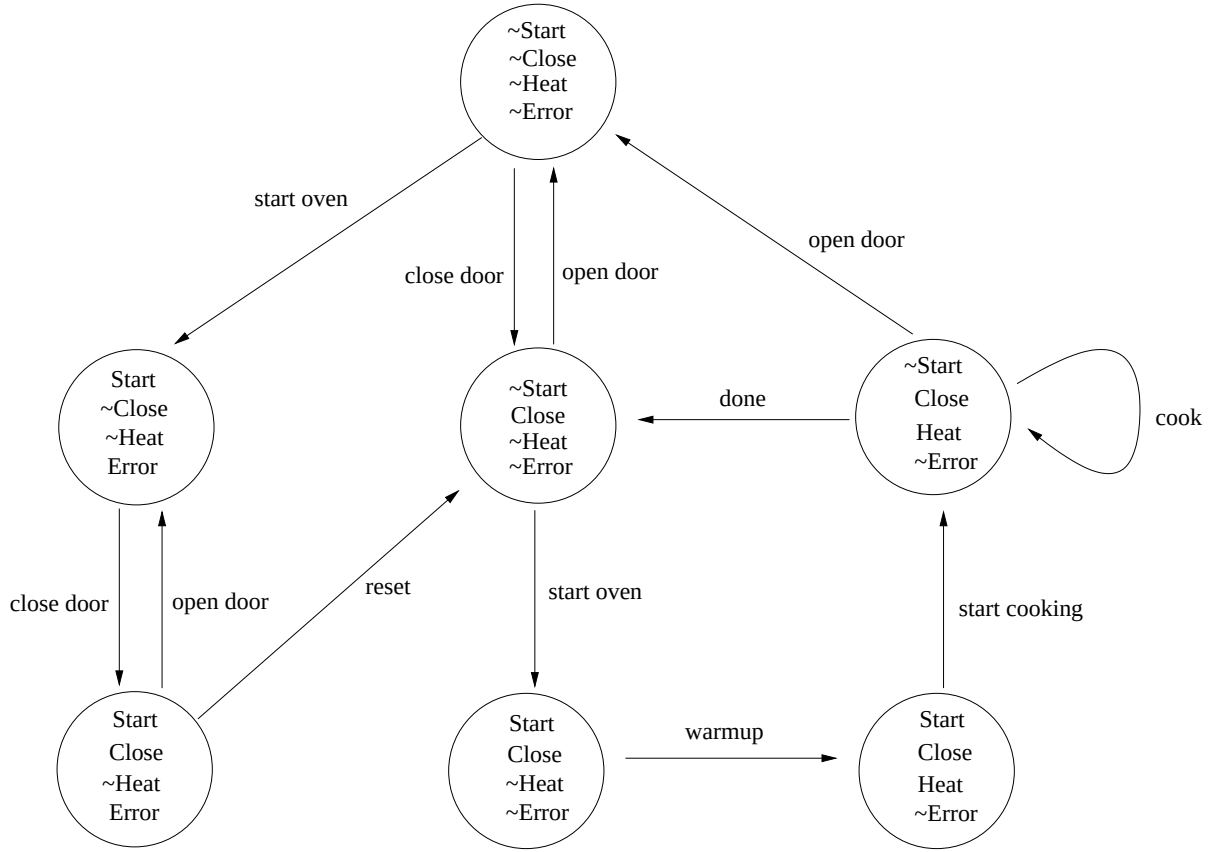
Verification

$\mathbf{AG}((\texttt{Start} \wedge \neg\texttt{Error}) \rightarrow \mathbf{AF}\texttt{Heat})$ is ?



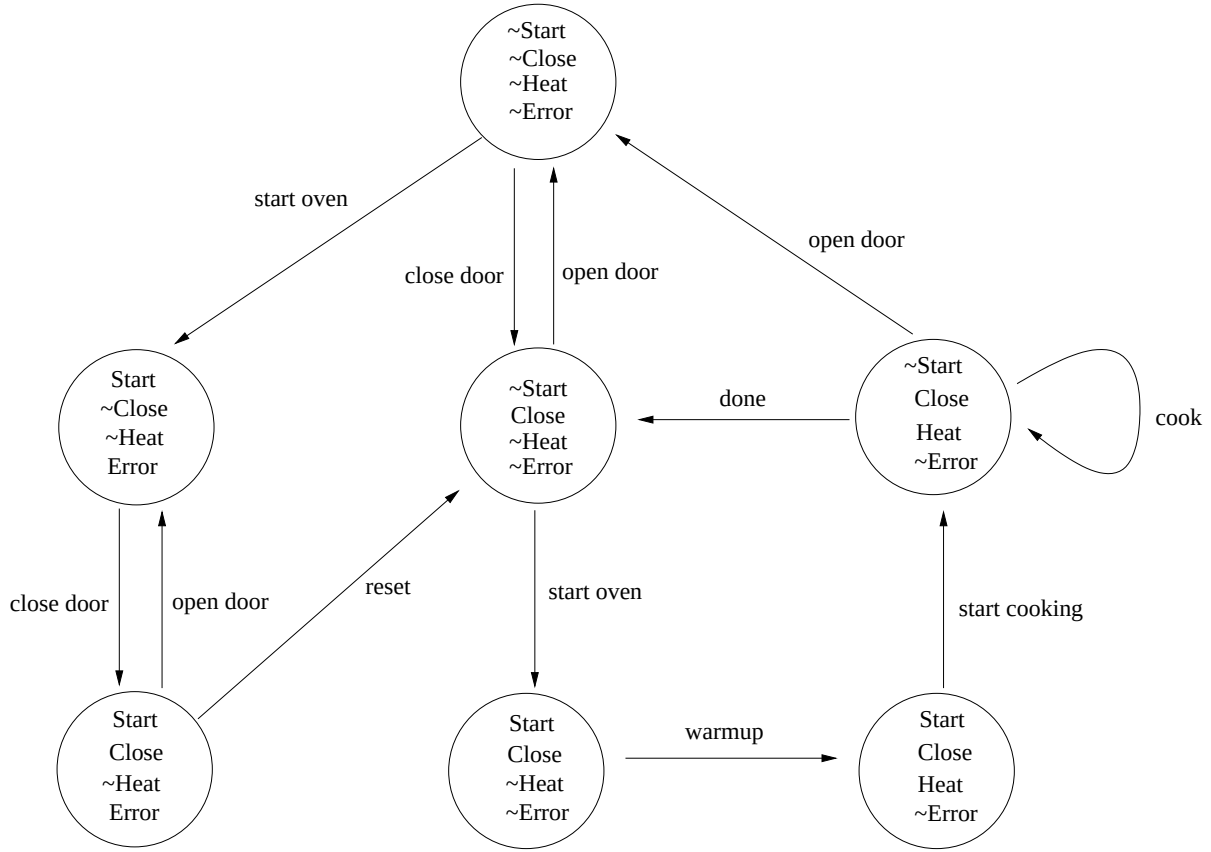
Verification

$\mathbf{AG}((\texttt{Start} \wedge \neg\texttt{Error}) \rightarrow \mathbf{AF}\texttt{Heat})$ is true!



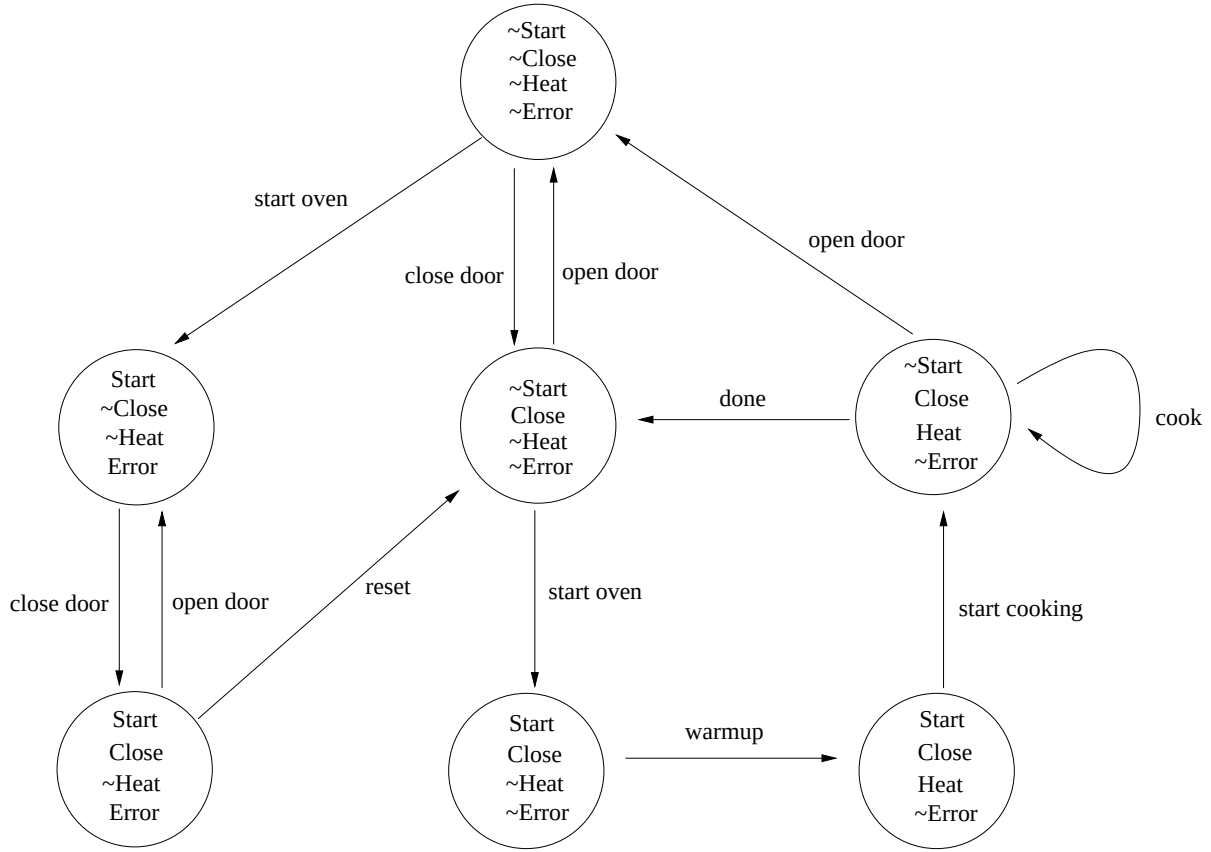
Verification

AG(`Error` $\rightarrow$ **EF**`Heat`) is ?



Verification

AG(`Error` $\rightarrow$ **EF**`Heat`) is true!



Outline - Class II

1. Modal logic ML
2. Model checking for ML
3. Computation Tree Logic CTL
4. The microwave oven example
5. **Model checking for CTL**

Model checking for CTL

We elaborate subformulas in increasing length order (like in the modal case).

Cases **EX** and **AX** are resolved by visiting the successors of the current state. However, cases **EU** and **AU** impose us to visit, in the worst-case, all the nodes reachable by a path from the current state. We will take advantage of the following **recursive** definitions:

$$\begin{aligned} \mathbf{EU}(\alpha, \beta) &= \beta \vee (\alpha \wedge \mathbf{EXEU}(\alpha, \beta)) \\ \mathbf{AU}(\alpha, \beta) &= \beta \vee (\alpha \wedge \mathbf{AXAU}(\alpha, \beta)) \end{aligned}$$

Procedure `CTL Model Checker`$(\mathfrak{M} = \langle M, R, V\rangle, \alpha)$
for **every** $i = 1, \ldots, |\alpha|$ **do**
  for **every** $\beta \in Sub(\alpha)$ **such that** $|\beta| = i$ **do**
    case on the form of β
      $\star$ propositional cases
      $\star$ $\beta = \mathbf{EX}\gamma$
      for **every** $w \in M$ **do**
        if $\exists v.\ Rwv \wedge \gamma \in V(v)$ **then**
          $V(w) = V(w) \cup \{\beta\}$
      $\star$ $\beta = \mathbf{AX}\gamma$
      for **every** $w \in M$ **do**
        if $\forall v.\ Rwv \rightarrow \gamma \in V(v)$ **then**
          $V(w) = V(w) \cup \{\beta\}$

$\star\ \beta = \mathbf{EU}(\gamma, \delta)$
for **every** $w \in M$ **do** $\texttt{marked}(w) = \texttt{False}$
for **every** $w \in M$ **do**
    if $w \in V(\delta)$ **then** $\texttt{CheckEU}(\mathfrak{M}, w, \gamma, \delta)$
$\star\ \beta = \mathbf{AU}(\gamma, \delta)$
for **every** $w \in M$ **do** $\texttt{marked}(w) = \texttt{False}$
for **every** $w \in M$ **do**
    if not $\texttt{marked}(w)$ **then** $\texttt{CheckAU}(\mathfrak{M}, w, \gamma, \delta)$

Procedure `CheckEU`$(\mathfrak{M}, w, \gamma, \delta)$
if not `marked`(w) **then**
`marked`$(w) =$ `True`
$V(w) = V(w) \cup \{\mathbf{EU}(\gamma, \delta)\}$
for every v **such that** Rvw **do**
if $v \in V(\gamma)$ **then** `CheckEU`$(\mathfrak{M}, v, \gamma, \delta)$

Function `CheckAU`$(\mathfrak{M}, w, \gamma, \delta)$
if `marked`(w) **then**
  if $\mathbf{AU}(\gamma, \delta) \in V(w)$ **then return** `True` **else return** `False`
`marked`$(w) =$ `True`
if $\delta \in V(w)$ **then**
  $V(w) = V(w) \cup \{\mathbf{AU}(\gamma, \delta)\}$
  return `True`
if $\gamma \notin V(w)$ **then**
  return `False`
for **every** v **such that** Rwv **do**
    if not `CheckAU`$(\mathfrak{M}, v, \gamma, \delta)$ **then**
      return `False`
$V(w) = V(w) \cup \{\mathbf{AU}(\gamma, \delta)\}$
return `True`

Computational complexity

Let $n = |M|$ be the **number of nodes**, $m = |R|$ be the **number of edges**, and k be the **length of** α.

The main for loop runs for k times. The Boolean cases cost $O(n)$. The temporal cases cost $O(n + m)$. Hence, the model checker for CTL runs in time $O(k \cdot (n + m))$ in the worst-case.

The complexity is **linear** in the product of the length of the formula and the size of the model.

References

- E. M. Clarke, O. Grumberg, D. Kroening, D. A. Peled, and H. Veith, Model Checking, second edition, The MIT Press. 2018.
- M. R. A. Huth and M. D. Ryan. Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press, 2000.