Universität Bremen

dfki Deutsches Forschungszentrum für Künstliche Intelligenz
German Research Center for Artificial Intelligence

Systems of High Safety and Security

Lecture 6 from 19.11.2025: Temporal Logic with LTL and CTL

Winter term 2025/26

Christoph Lüth

Roadmap

- Introduction
- Legal Requirements - Norms and Standards
- The Development Process
- Hazard Analysis
- The Big Picture: Hybrid Systems
- Temporal Logic with LTL and CTL
- Operational Semantics
- Axiomatic Semantics - Specifying Correctness
- Floyd-Hoare Logic
- A Simple Compiler and its Correctness
- Hardware Verification
- A Simple TinyRV32 Core
- Conclusions

Introduction

- We have seen that **state machines** are a general system model.
- Now the question is: how do we state and **prove** properties of systems modelled as finite state machines?
- There are many answers, depending on the level of **abstraction**. On the most abstract level, we can use **temporal logic**.
- On this abstract level, the question is how to prove a propert ϕ of a system modelled as a FSM $\mathcal{M}$.

The Model-Checking Problem

The Basic Question

Given a model $\mathcal{M}$, and a property ϕ, we want to know whether

$$\mathcal{M} \models \phi$$

- What is $\mathcal{M}$?
- What is ϕ?
- How to prove it?

The Model-Checking Problem

The Basic Question

Given a model $\mathcal{M}$, and a property ϕ, we want to know whether

$$\mathcal{M} \models \phi$$

- What is $\mathcal{M}$? **Finite state machines**
- What is ϕ?
- How to prove it?

The Model-Checking Problem

The Basic Question

Given a model $\mathcal{M}$, and a property ϕ, we want to know whether

$$\mathcal{M} \models \phi$$

- What is $\mathcal{M}$? **Finite state machines**
- What is ϕ? **Temporal logic**
- How to prove it?

The Model-Checking Problem

The Basic Question

Given a model $\mathcal{M}$, and a property ϕ, we want to know whether

$$\mathcal{M} \models \phi$$

- What is $\mathcal{M}$? **Finite state machines**
- What is ϕ? **Temporal logic**
- How to prove it? Enumerating states — **model checking**

The Model-Checking Problem

The Basic Question

Given a model $\mathcal{M}$, and a property ϕ, we want to know whether

$$\mathcal{M} \models \phi$$

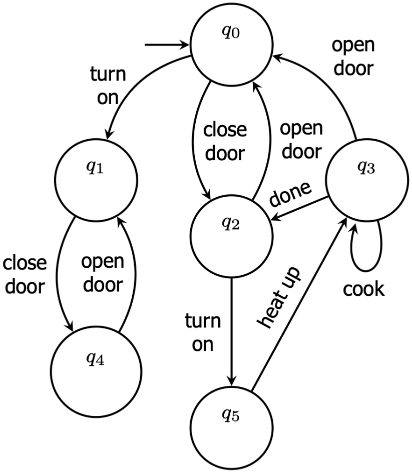
- What is $\mathcal{M}$? **Finite state machines**
- What is ϕ? **Temporal logic**
- How to prove it? Enumerating states — **model checking**
  - The basic **problem**: **state explosion**

Example: A Simple Baking Oven

- The oven has states and operations:
  - open and close door,
  - turn oven on and off,
  - warm up and cook.
- How do they interact?

Example: A Simple Baking Oven

- The oven has states and operations:
  - open and close door,
  - turn oven on and off,
  - warm up and cook.
- How do they interact?



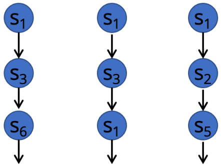
Questions to Ask

- We want to answer **questions** about the system **behaviour** like
  - Can the cooker heat up with the door open?
  - When the start button is pushed, will the cooker eventually heat up?
  - When the cooker is correctly started, will the cooker eventually heat up?
  - When an error occurs, will it be still possible to cook?
- We are interested in questions on the evolution of the system over time, i.e. possible **traces** of the system given by a succession of states.
- The tool to formalize and answer these questions is **temporal logic**.

Basic Concepts of Time

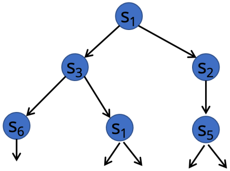
Linear Time

- Every moment has a **unique** successor
- Infinite **sequences** of moments
- Linear Temporal Logic (LTL)



Branching Time

- Every moment has **several** successors
- Infinite **tree** of moments
- Computational Tree Logic (CTL)



Atomic Propositions and States

- The basis of temporal logics are FSMs and **state predicates**.
- At each state of an FSM, a set of state predicates hold.
- This is called a **Kripke structure**.

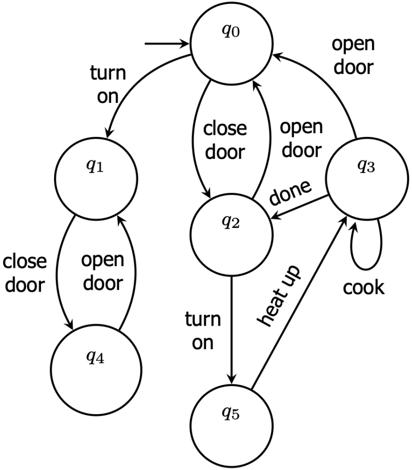
Definition: Kripke Structure

For a set *Prop* of atomic propositions, a **Kripke structure** $\mathcal{K} = \langle \mathcal{M}, V \rangle$ is given by a FSM $\mathcal{M} = \langle Q, Q_0, \rightarrow \rangle$ and a function $V : Q \rightarrow 2^{Prop}$ mapping each state to the set of atomic propositions holding in that state.

For $q \in Q, p \in Prop$, we write $q \models p$ if $p \in V(q)$ (p holds in q).

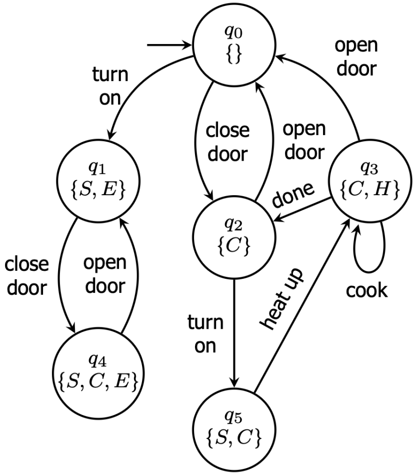
Example: The Simple Baking Oven

- Atomic propositions:
  - C: door closed.
  - S: oven started
  - H: oven hot
  - E: error occured
- Label states appropriately



Example: The Simple Baking Oven

- Atomic propositions:
  - C: door closed.
  - S: oven started
  - H: oven hot
  - E: error occured
- Label states appropriately



Linear Temporal Logic (LTL)

$$
\begin{array}{rcll}
\phi & ::= & \top \mid \bot \mid a & \text{— True, false, atomic} \\
& \mid & \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \longrightarrow \phi_2 & \text{— Propositional formulae} \\
& \mid & \mathsf{X}\,\phi & \text{— Next state} \\
& \mid & \Diamond\phi & \text{— Some Future State} \\
& \mid & \Box\phi & \text{— All future states (Globally)} \\
& \mid & \phi_1\ U\ \phi_2 & \text{— Until}
\end{array}
$$

- Operator precedence: Unary operators; then U; then $\wedge$, $\vee$; then $\longrightarrow$.
- An atomic formula p above denotes a **state predicate**.
- From these, we can define other operators, such as $\phi\ R\ \psi$ (release) or $\phi\ W\ \psi$ (weak until).

Satifsaction and Models of LTL

Given a path (infinite trace) p and an LTL formula ϕ, the **satisfaction relation** $p \models \phi$ is defined inductively as follows:

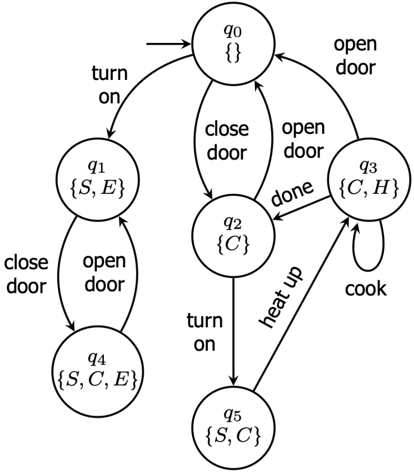
$$
\begin{array}{lll}
p & \models & \mathit{true} \\
p & \not\models & \mathit{false} \\
p & \models & a \text{ iff } p[0] \models a \\
p & \models & \neg\phi \text{ iff } p \not\models \phi
\end{array}
\qquad
\begin{array}{lll}
p & \models & \phi \wedge \psi \text{ iff } p \models \phi \text{ and } p \models \psi \\
p & \models & \phi \vee \psi \text{ iff } p \models \phi \text{ or } p \models \psi \\
p & \models & \phi \longrightarrow \psi \text{ iff whenever } p \models \phi \text{ then } p \models \psi
\end{array}
$$

$$
\begin{array}{lll}
p & \models & \mathrm{X}\,\phi \text{ iff } p^1 \models \phi \\
p & \models & \Box\phi \text{ iff for all } i\text{, we have } p^i \models \phi \\
p & \models & \Diamond\phi \text{ iff there is } i \text{ such that } p^i \models \phi \\
p & \models & \phi\, U\, \psi \text{ iff there is } i\ p^i \models \psi \text{ and for all } j = 1, \ldots, i-1,\ \ p^j \models \phi
\end{array}
$$

Models of LTL formulae

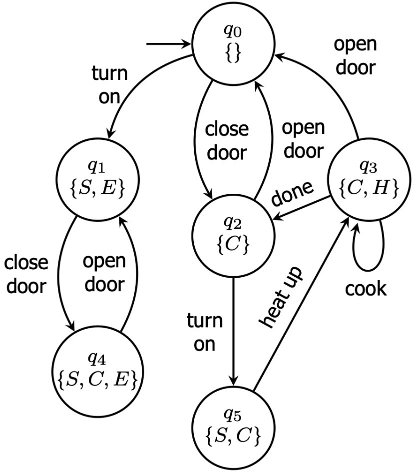
A Kripke structure $\mathcal{K} = \langle M, V\rangle$ satisfies an LTL formula ϕ, $\mathcal{K} \models \phi$, iff for every path $p \in \mathrm{Tr}(\mathcal{M})$, $p \models \phi$.

Example: The Simple Baking Oven



- If the cooker heats, then is the door closed?

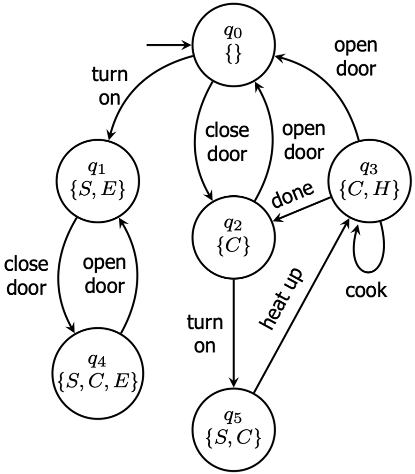
Example: The Simple Baking Oven



- If the cooker heats, then is the door closed?

$$\Box H \longrightarrow C$$

Example: The Simple Baking Oven

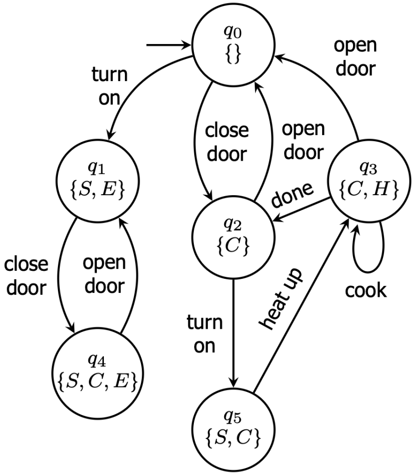


- If the cooker heats, then is the door closed?

$$\Box H \longrightarrow C \quad \checkmark$$

- Is it always possible to recover from an error?

Example: The Simple Baking Oven



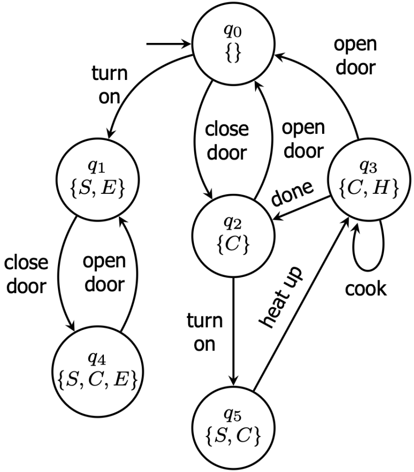
- If the cooker heats, then is the door closed?

$$\Box H \longrightarrow C \quad \checkmark$$

- Is it always possible to recover from an error?

$$\Box(E \longrightarrow \Diamond(\neg E))$$

Example: The Simple Baking Oven



- If the cooker heats, then is the door closed?

$$\Box H \longrightarrow C \quad \checkmark$$

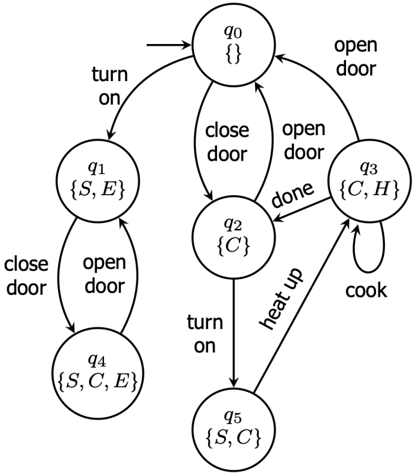
- Is it always possible to recover from an error?

$$\Box(E \longrightarrow \Diamond(\neg E)) \quad \times$$

  - Need to add a reset transition.

- Is it always possible to heat up, then cook?

Example: The Simple Baking Oven



- If the cooker heats, then is the door closed?

$$\Box H \longrightarrow C \quad \checkmark$$

- Is it always possible to recover from an error?

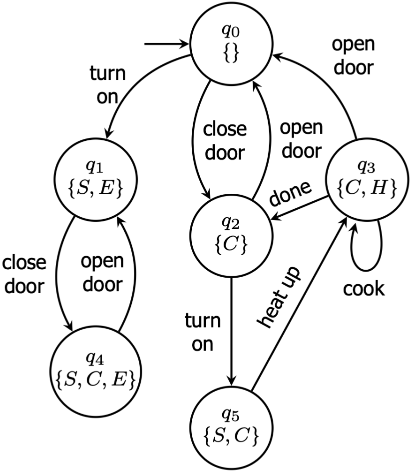
$$\Box(E \longrightarrow \Diamond(\neg E)) \quad \times$$

  - Need to add a reset transition.

- Is it always possible to heat up, then cook?

$$\Diamond(S \longrightarrow \mathsf{X}\, H)$$

Example: The Simple Baking Oven



- If the cooker heats, then is the door closed?

$$\square H \longrightarrow C \quad \checkmark$$

- Is it always possible to recover from an error?

$$\square(E \longrightarrow \lozenge(\neg E)) \quad \times$$

  - Need to add a reset transition.

- Is it always possible to heat up, then cook?

$$\lozenge(S \longrightarrow \mathsf{X}\, H) \quad \times$$

  - Always possible to avoid cooking.
  - Cannot express '*there are paths in which we can always cook*'.

Computational Tree Logic (CTL)

- LTL does not allow us the quantify over paths, e.g. assert the existance of a path satisfying a particular property.
- To a limited degree, we can solve this problem by negation: instead of asserting a property ϕ, we check wether $\neg\phi$ is satisfied; if that is not the case, ϕ holds. But this does not work for mixtures of universal and existential quantifiers.
- Computational Tree Logic (CTL) is an extension of LTL which allows this by adding universal and existential quantifiers to the modal operators.
- The name comes from considering paths in the **computational tree** obtained by **unwinding** the FSM.

CTL Formulae

$$
\begin{array}{rcll}
\phi & ::= & \top \mid \bot \mid p & \text{— True, false, atomic} \\
& \mid & \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \longrightarrow \phi_2 & \text{— Propositional formulae} \\
& \mid & \mathsf{AX}\,\phi \mid \mathsf{EX}\,\phi & \text{— All or some next state} \\
& \mid & \mathsf{AF}\,\phi \mid \mathsf{EF}\,\phi & \text{— All or some future states} \\
& \mid & \mathsf{AG}\,\phi \mid \mathsf{EG}\,\phi & \text{— All or some global future} \\
& \mid & \mathsf{A}[\phi_1\ U\ \phi_2] \mid \mathsf{E}[\phi_1\ U\ \phi_2] & \text{— Until all or some}
\end{array}
$$

Satifsfaction

- Note that CTL formulae can be considered to be a LTL formulae with a 'modality' (A or E) added on top of each temporal operator.
- Generally speaking, the A modality says the temporal operator holds for all paths, and the E modality says the temporal operator only holds for all least one path.
  - Of course, that strictly speaking is not true, because the arguments of the temporal operators are in turn CTL forumulae, so we need recursion.
- This all explains why we do not define a satisfaction for a single path p, but satisfaction with respect to a specific **state** in an FSM.

Satisfaction for CTL

Given a Kripke-structure $\mathcal{K} = \langle \mathcal{M}, V \rangle$, $s \in \Sigma$ and a CTL formula ϕ, then $\mathcal{K}, s \models \phi$ is defined inductively as follows:

$$
\begin{array}{lcl}
\mathcal{K}, s & \models & \mathit{true} \\
\mathcal{K}, s & \not\models & \mathit{false} \\
\mathcal{K}, s & \models & p \text{ iff } s \models p \\
\mathcal{K}, s & \models & \phi \wedge \psi \text{ iff } \mathcal{K}, s \models \phi \text{ and } \mathcal{K}, s \models \psi \\
\mathcal{K}, s & \models & \phi \vee \psi \text{ iff } \mathcal{K}, s \models \phi \text{ or } \mathcal{K}, s \models \psi \\
\mathcal{K}, s & \models & \phi \longrightarrow \psi \text{ iff whenever } \mathcal{K}, s \models \phi \text{ then } \mathcal{K}, s \models \psi \\
\ldots & &
\end{array}
$$

Satisfaction for CTL (c'ed)

Given a Kripke-structure $\mathcal{K} = \langle \mathcal{M}, V \rangle$, $s \in \Sigma$ and a CTL formula ϕ, then $\mathcal{K}, s \models \phi$ is defined inductively as follows:

$$
\begin{array}{lcl}
\ldots & & \\
\mathcal{K}, s & \models & \mathrm{AX}\,\phi \text{ iff for all } s_1 \text{ with } s \rightarrow s_1 \text{, we have } \mathcal{K}, s_1 \models \phi \\
\mathcal{K}, s & \models & \mathrm{EX}\,\phi \text{ iff for some } s_1 \text{ with } s \rightarrow s_1 \text{, we have } \mathcal{K}, s_1 \models \phi
\end{array}
$$

Satisfaction for CTL (c'ed)

Given a Kripke-structure $\mathcal{K} = \langle \mathcal{M}, V \rangle$, $s \in \Sigma$ and a CTL formula ϕ, then $\mathcal{K}, s \models \phi$ is defined inductively as follows:

$$
\begin{array}{lcl}
\ldots & & \\
\mathcal{K}, s & \models & \mathrm{AX}\,\phi \text{ iff for all } s_1 \text{ with } s \to s_1 \text{, we have } \mathcal{K}, s_1 \models \phi \\
\mathcal{K}, s & \models & \mathrm{EX}\,\phi \text{ iff for some } s_1 \text{ with } s \to s_1 \text{, we have } \mathcal{K}, s_1 \models \phi \\
\mathcal{K}, s & \models & \mathrm{AG}\,\phi \text{ iff for all paths } p \text{ with } p[0] = s \text{, we have } \mathcal{K}, p[i] \models \phi \text{ for all } i \geq 1 \\
\mathcal{K}, s & \models & \mathrm{EG}\,\phi \text{ iff there is a path } p \text{ with } p[0] = s \text{ and } \mathcal{K}, p[i] \models \phi \text{ for all } i \geq 1
\end{array}
$$

Satisfaction for CTL (c'ed)

Given a Kripke-structure $\mathcal{K} = \langle \mathcal{M}, V \rangle$, $s \in \Sigma$ and a CTL formula ϕ, then $\mathcal{K}, s \models \phi$ is defined inductively as follows:

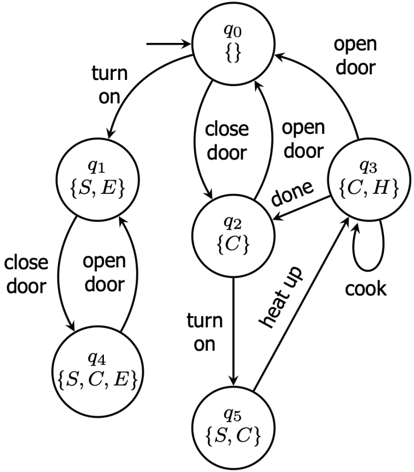
$$
\begin{array}{lcl}
\ldots & & \\
\mathcal{K}, s & \models & \mathrm{AX}\,\phi \text{ iff for all } s_1 \text{ with } s \to s_1\text{, we have } \mathcal{K}, s_1 \models \phi \\
\mathcal{K}, s & \models & \mathrm{EX}\,\phi \text{ iff for some } s_1 \text{ with } s \to s_1\text{, we have } \mathcal{K}, s_1 \models \phi \\
\mathcal{K}, s & \models & \mathrm{AG}\,\phi \text{ iff for all paths } p \text{ with } p[0] = s\text{, we have } \mathcal{K}, p[i] \models \phi \text{ for all } i \geq 1 \\
\mathcal{K}, s & \models & \mathrm{EG}\,\phi \text{ iff there is a path } p \text{ with } p[0] = s \text{ and } \mathcal{K}, p[i] \models \phi \text{ for all } i \geq 1 \\
\mathcal{K}, s & \models & \mathrm{AF}\,\phi \text{ iff for all paths } p \text{ with } p[0] = s\text{, we have } \mathcal{K}, p[i] \models \phi \text{ for some } i \\
\mathcal{K}, s & \models & \mathrm{EF}\,\phi \text{ iff there is a path } p \text{ with } p[0] = s \text{ and } \mathcal{K}, p[i] \models \phi \text{ for some } i
\end{array}
$$

Satisfaction for CTL (c'ed)

Given a Kripke-structure $\mathcal{K} = \langle \mathcal{M}, V \rangle$, $s \in \Sigma$ and a CTL formula ϕ, then $\mathcal{K}, s \models \phi$ is defined inductively as follows:

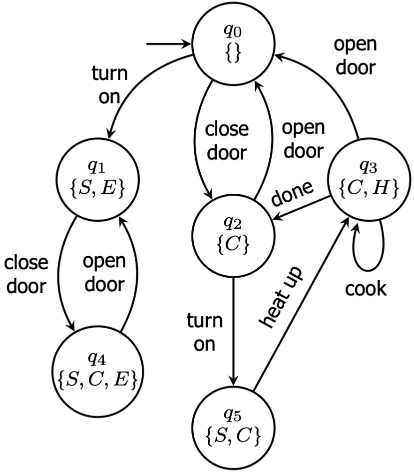
$$
\begin{array}{lcl}
\ldots & & \\
\mathcal{K}, s & \models & \mathsf{AX}\,\phi \text{ iff for all } s_1 \text{ with } s \to s_1 \text{, we have } \mathcal{K}, s_1 \models \phi \\
\mathcal{K}, s & \models & \mathsf{EX}\,\phi \text{ iff for some } s_1 \text{ with } s \to s_1 \text{, we have } \mathcal{K}, s_1 \models \phi \\
\mathcal{K}, s & \models & \mathsf{AG}\,\phi \text{ iff for all paths } p \text{ with } p[0] = s \text{, we have } \mathcal{K}, p[i] \models \phi \text{ for all } i \geq 1 \\
\mathcal{K}, s & \models & \mathsf{EG}\,\phi \text{ iff there is a path } p \text{ with } p[0] = s \text{ and } \mathcal{K}, p[i] \models \phi \text{ for all } i \geq 1 \\
\mathcal{K}, s & \models & \mathsf{AF}\,\phi \text{ iff for all paths } p \text{ with } p[0] = s \text{, we have } \mathcal{K}, p[i] \models \phi \text{ for some } i \\
\mathcal{K}, s & \models & \mathsf{EF}\,\phi \text{ iff there is a path } p \text{ with } p[0] = s \text{ and } \mathcal{K}, p[i] \models \phi \text{ for some } i \\
\mathcal{K}, s & \models & \mathsf{A}[\phi\; U\; \psi] \text{ iff for all paths } p \text{ with } p[0] \text{, there is } i \\
 & & \qquad \text{with } \mathcal{K}, p[i] \models \psi \text{ and for all } j < i,\ \mathcal{K}, p[j] \models \phi \\
\mathcal{K}, s & \models & \mathsf{E}[\phi\; U\; \psi] \text{ iff there is a path } p \text{ with } p_1 = s \text{ and there is } i \\
 & & \qquad \text{with } \mathcal{K}, p[i] \models \psi \text{ and for all } j < i,\ \mathcal{K}, p[j] \models \phi
\end{array}
$$

Example: The Simple Baking Oven



- Whenever the oven is hot, the door is closed.

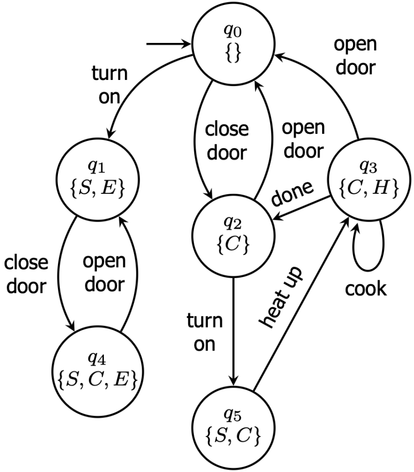
Example: The Simple Baking Oven



- Whenever the oven is hot, the door is closed.

$$\mathsf{AG}\, H \longrightarrow C$$

Example: The Simple Baking Oven

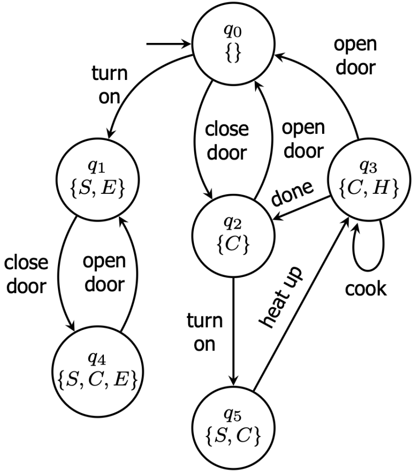


- Whenever the oven is hot, the door is closed.

$$\mathsf{AG}\, H \longrightarrow C \quad \checkmark$$

- After cooking, we will get access to the food:

Example: The Simple Baking Oven



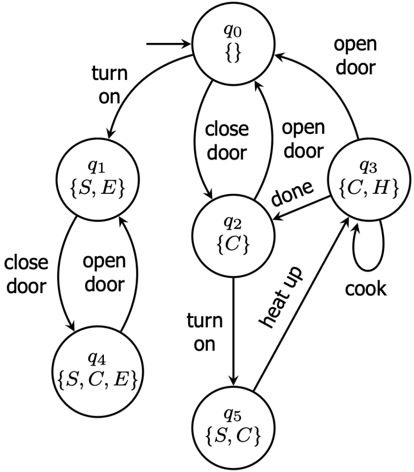
- Whenever the oven is hot, the door is closed.

$$\mathsf{AG}\, H \longrightarrow C \quad \checkmark$$

- After cooking, we will get access to the food:

$$\mathsf{AF}(H \longrightarrow \mathsf{AF}\,\neg C)$$

Example: The Simple Baking Oven



- Whenever the oven is hot, the door is closed.

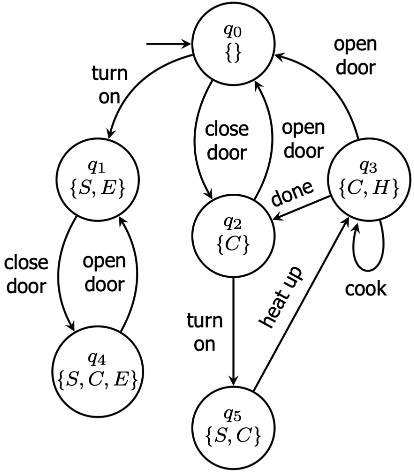
$$\mathsf{AG}\, H \longrightarrow C \quad \checkmark$$

- After cooking, we will get access to the food:

$$\mathsf{AF}(H \longrightarrow \mathsf{AF}\,\neg C) \quad \times$$

- After cooking, we may get access to the food:

Example: The Simple Baking Oven



- Whenever the oven is hot, the door is closed.

$$\mathsf{AG}\, H \longrightarrow C \quad \checkmark$$

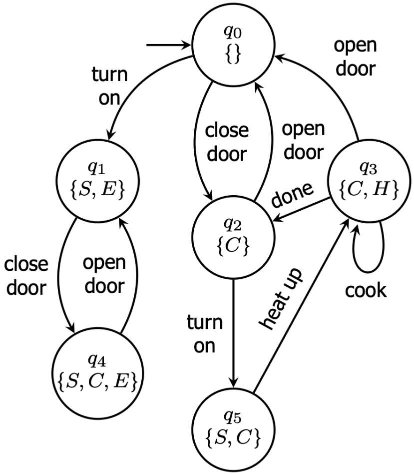
- After cooking, we will get access to the food:

$$\mathsf{AF}(H \longrightarrow \mathsf{AF}\,\neg C) \quad \times$$

- After cooking, we may get access to the food:

$$\mathsf{AF}(H \longrightarrow \mathsf{EF}\,\neg C)$$

Example: The Simple Baking Oven



- Whenever the oven is hot, the door is closed.

$$\mathsf{AG}\, H \longrightarrow C \quad \checkmark$$

- After cooking, we will get access to the food:

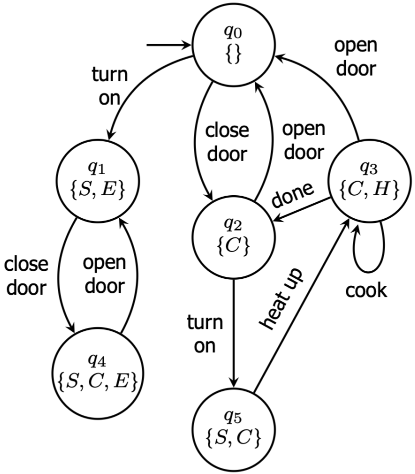
$$\mathsf{AF}(H \longrightarrow \mathsf{AF}\, \neg C) \quad \times$$

- After cooking, we may get access to the food:

$$\mathsf{AF}(H \longrightarrow \mathsf{EF}\, \neg C) \quad \times$$

- The oven will always eventually heat up

Example: The Simple Baking Oven



- Whenever the oven is hot, the door is closed.

$$\mathsf{AG}\, H \longrightarrow C \quad \checkmark$$

- After cooking, we will get access to the food:

$$\mathsf{AF}(H \longrightarrow \mathsf{AF}\,\neg C) \quad \times$$

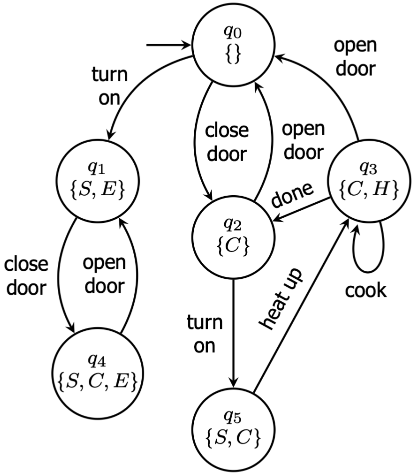
- After cooking, we may get access to the food:

$$\mathsf{AF}(H \longrightarrow \mathsf{EF}\,\neg C) \quad \times$$

- The oven will always eventually heat up

$$\mathsf{AG}(\mathsf{EF}(\neg H \land \mathsf{EX}\, H))$$

Example: The Simple Baking Oven



- Whenever the oven is hot, the door is closed.

$$\mathsf{AG}\, H \longrightarrow C \quad \checkmark$$

- After cooking, we will get access to the food:

$$\mathsf{AF}(H \longrightarrow \mathsf{AF}\,\neg C) \quad \times$$

- After cooking, we may get access to the food:

$$\mathsf{AF}(H \longrightarrow \mathsf{EF}\,\neg C) \quad \times$$

- The oven will always eventually heat up

$$\mathsf{AG}(\mathsf{EF}(\neg H \wedge \mathsf{EX}\, H)) \quad \times$$

  - Only with reset transition.

Patterns of Specification

- Something bad (p) cannot happen: $\mathsf{AG}\,\neg p$
- p occurs infinitly often: $\mathsf{AG}(\mathsf{AF}\,p)$
- p occurs eventually: $\mathsf{AF}\,p$
- In the future, p will hold eventually forever: $\mathsf{AF}\,\mathsf{AG}\,p$
- Whenever p will hold in the future, q will hold eventually: $\mathsf{AG}(p \longrightarrow \mathsf{AF}\,q)$
- In all states, p is always possible: $\mathsf{AG}(\mathsf{EF}\,p)$

LTL and CTL

- We have seen that CTL is more expressive than LTL, but (surprisingly), there are properties which we can formalise in LTL but not in CTL!
- Example: all paths which have a p along them also have a q along them.
- LTL: $\Diamond p \longrightarrow \Diamond q$
- CTL: **Not** $\mathrm{AF}\, p \longrightarrow \mathrm{AF}\, q$ (would mean: if all paths have p, then all paths have q), neither $\mathrm{AG}(p \longrightarrow \mathrm{AF}\, q)$ (which means: if there is a p, it will be followed by a q).
- The logic CTL^* combines both LTL and CTL (but we will not consider it further here).

State Explosion and Complexity

- The basic problem of model checking is **state explosion**.
- State grows **exponentially**: n states have a state space of 2^n. Add one integer variable with $n = 2^{32}$ states, and this gets intractable.
- Theoretically, there is not much hope. The basic problem of deciding wether a particular formula holds is known as the satisfiability problem, and for the temporal logics we have seen, its complexity is as follows:
  - LTL without U is NP-complete.
  - LTL is $PSPACE$-complete.
  - CTL is $EXPTIME$-complete.
- The good news is that at least it is **decidable**. Practically, **state abstraction** is the key technique. E.g. instead of considering all possible integer values, consider only wether i is zero or larger than zero.

Summary

- Model-checking allows us to show to show properties of systems by enumerating the system's states, by modelling systems as **finite state machines**, and expressing properties in temporal logic.
- We considered Linear Temporal Logic (LTL) and Computational Tree Logic (CTL). LTL allows us to express properties of single paths, CTL allows quantifications over all possible paths of an FSM.
- The basic problem: the system state can quickly get **huge**, and the basic complexity of the problem is **horrendous**. Use of abstraction and state compression techniques make model-checking bearable.
- Next: what has software to do with all of this? Operational Semantics.