Universität Bremen

Systeme hoher Sicherheit und Qualität

WS 2019/2020

Lecture 11:

Foundations of Model Checking

Christoph Lüth, Dieter Hutter, Jan Peleska

Where are we?

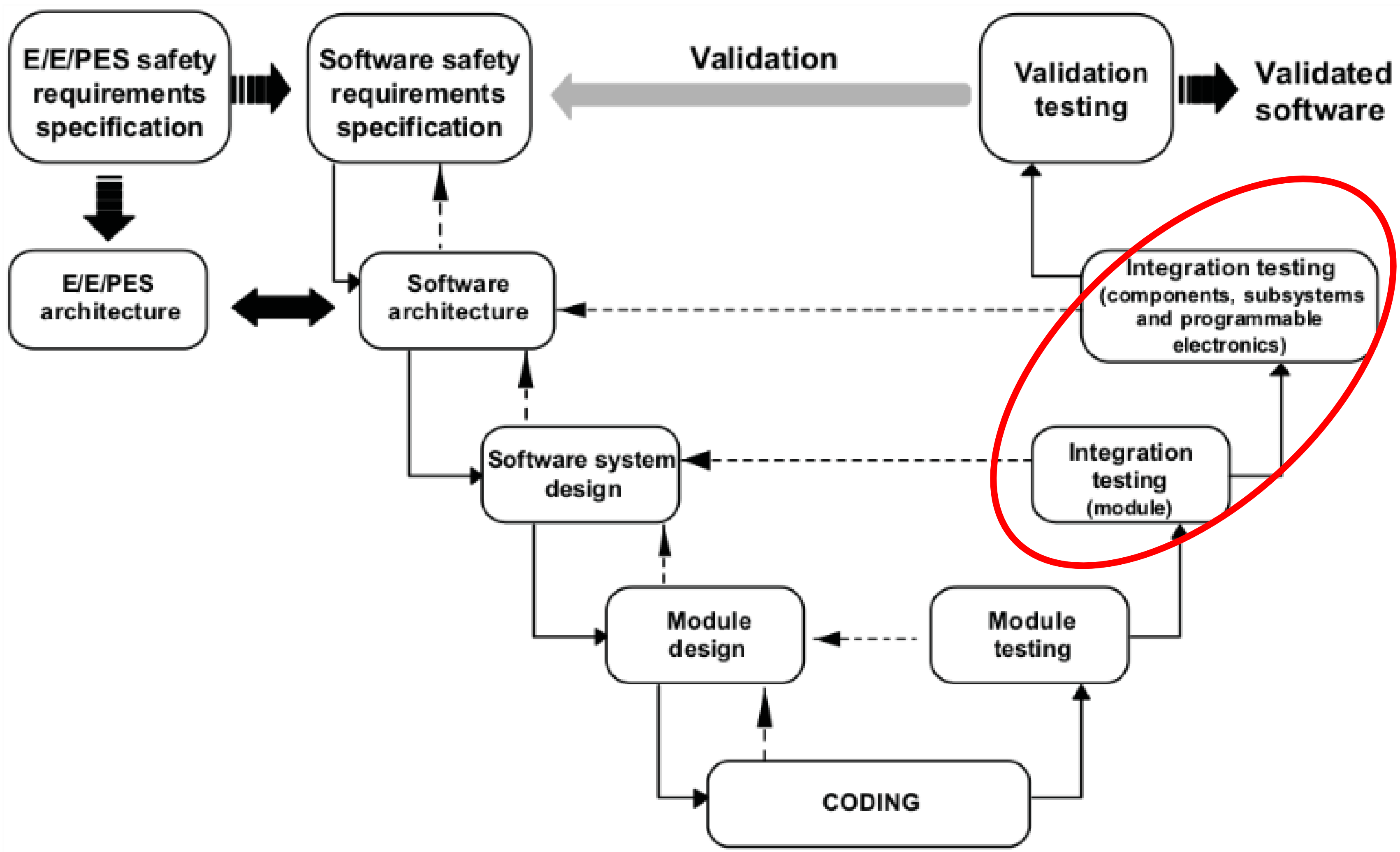
Introduction

- In the last lectures, we were verifying program properties with the Floyd-Hoare calculus (or verification condition generation). Program verification translates the question of program correctness into a **proof** in program logic (the Floyd-Hoare logic), turning it into a **deductive problem**.

- Model-checking takes a different approach: instead of directly working with the (source code) of the program, we work with an **abstraction** of the system (the system **model**). Because we build an abstraction, this approach is also applicable at higher verification levels. (It is also complimentary to deductive verification.)

- The **key questions** are: how do these models look like? What properties do we want to express, and how do we express and prove them?

Model Checking in the Development Cycle



Introduction

- Model checking operates on (abstract) state machines
  - Does an abstract system satisfy some behavioral property
    e.g. liveness (deadlock) or safety properties
    - consider traffic lights in Requirement Engineering
    - Example: “green must always follow red”

- Automatic analysis if state machine is finite
    - Push-button technology
    - User does not need to know logic (at least not for the proof)

- Basis is satisfiability of boolean formula in a finite domain (SAT). However, finiteness does not imply efficiency – all interesting problems are at least NP-complete, and SAT is no exception (Cook’s theorem).

The Model-Checking Problem

The **Basic Question**:

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

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

- What is $\mathcal{M}$?
  - A finite-state machine or Kripke structure.
- What is ϕ?
  - Temporal logic
- How to prove it?
  - By enumerating the states and thus construct a model (hence the term model checking)
  - The basic problem: state explosion

Finite State Machine (FSM)

Definition: Finite State Machine (FSM)

A FSM is given by $\mathcal{M} = \langle \Sigma, I, \rightarrow \rangle$ where

- Σ is a finite set of **states**,
- $I \subseteq \Sigma$ is a set of **initial** states, and
- $\rightarrow \subseteq \Sigma \times \Sigma$ is a **transition relation**, s.t. $\rightarrow$ is left-total:

$$\forall s \in \Sigma. \exists s' \in \Sigma. s \rightarrow s'$$

- Variations of this definition exists, e.g. no initial states.
- Note there is no final state, and no input or output (this is the key difference to **automata**).
- If $\rightarrow$ is a function, the FSM is deterministic, otherwise it is non-deterministic.

First Example: A Simple Drink Dispenser

1) Insert a coin.
2) Press button: tea or coffee
3) Tea or coffee dispensed
4) Back to 1)

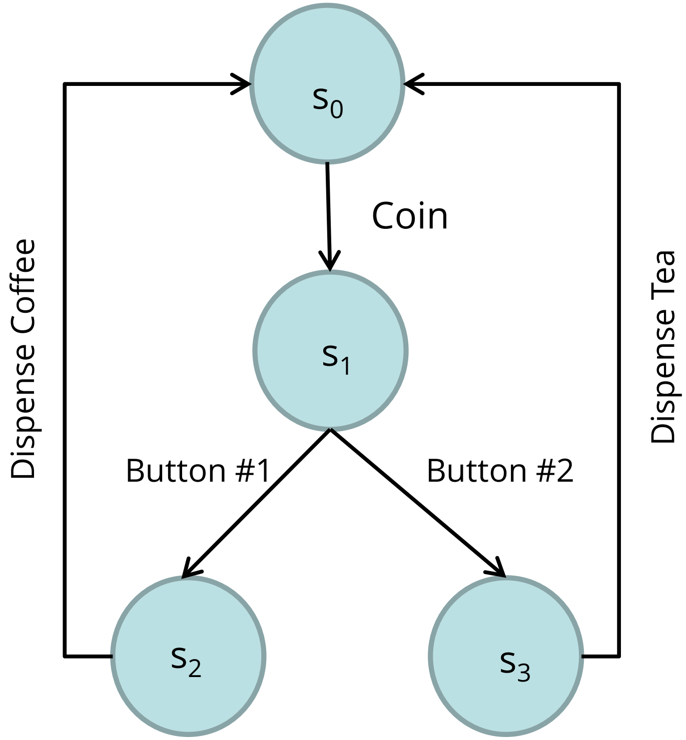
FSM:

$$\Sigma = \{\, s_0, s_1, s_2, s_3 \,\}$$

$$I = \{\, s_0 \}$$

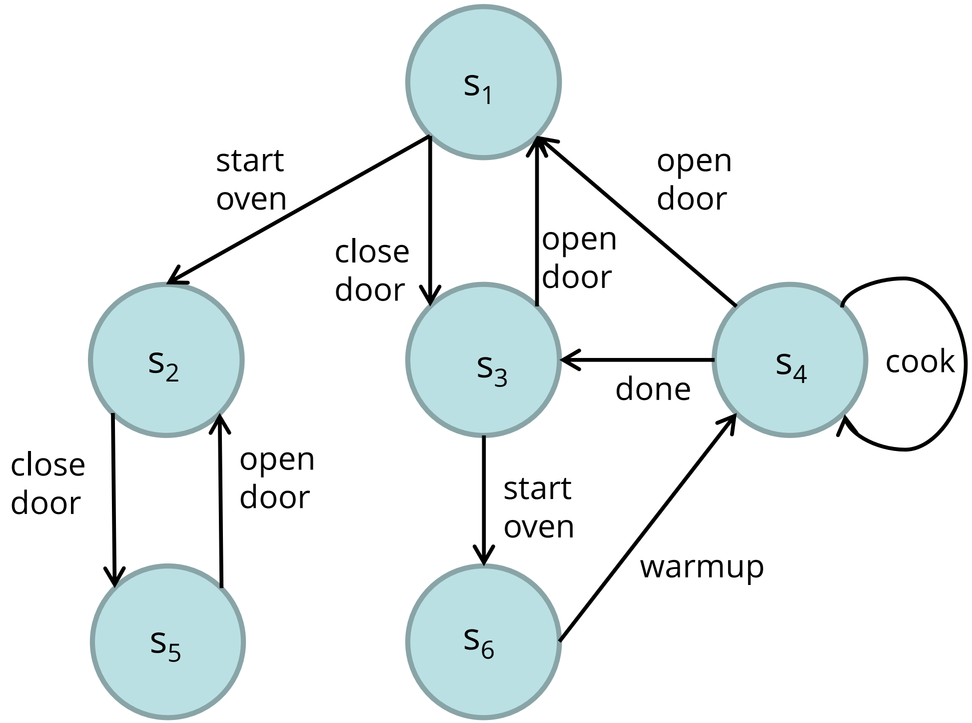
$$\rightarrow = \begin{array}{l} \{\, (s_0, s_1), (s_1, s_2), (s_2, s_3), \\ \quad (s_1, s_3), (s_2, s_0), (s_3, s_0) \,\} \end{array}$$

Note operation names are for decoration purposes only.



Example: A Simple Oven

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



Questions to ask

We want to answer **questions** about the system **behaviour** like

- Can the cooker heat 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 development of the system over time, i.e. possible **traces** of the system given by a succession of states.

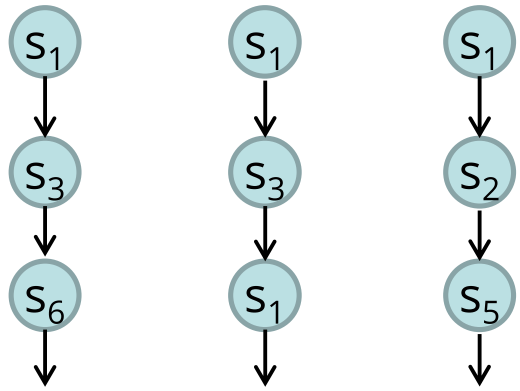
The tool to formalize and answer these questions is **temporal logic.**

Temporal Logic

Expresses properties of possible succession of states

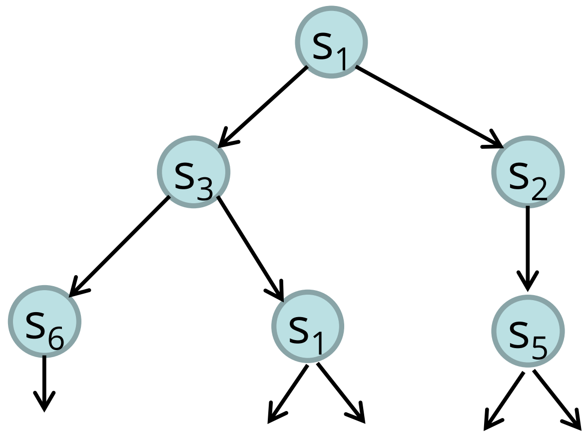
Linear Time

- Every moment in time has a unique successor
- Infinite sequences of moments
- Linear Temporal Logic LTL



Branching Time

- Every moment in time has several successors
- Infinite tree
- Computational Tree Logic CTL



Kripke Structures

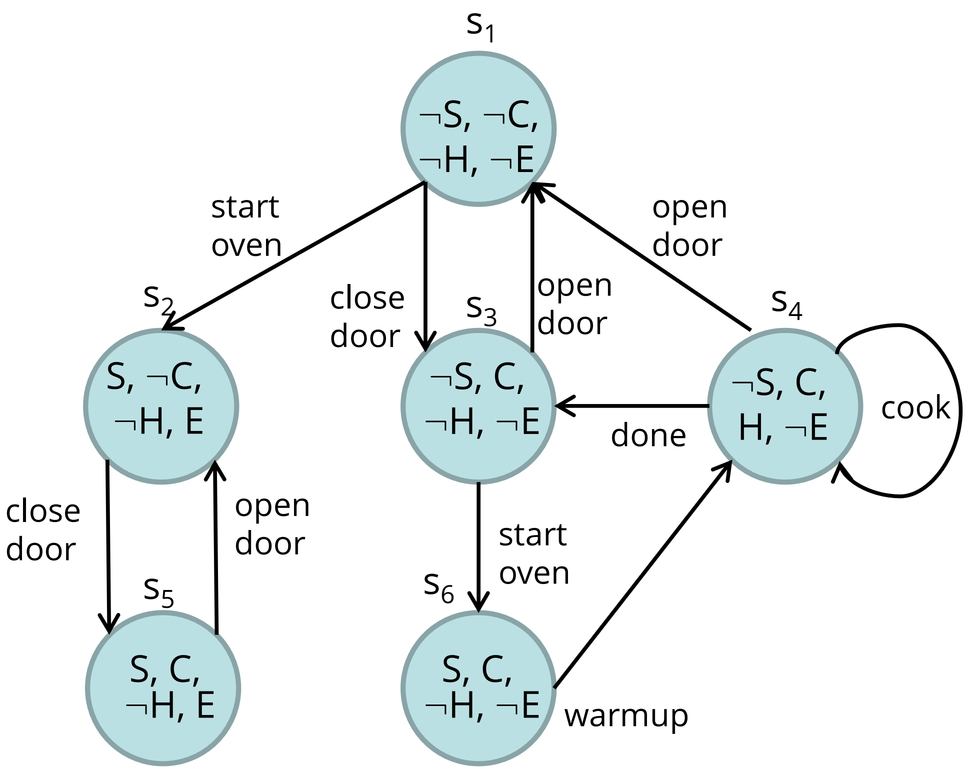
- In order to talk about propositions, we label the states of a FSM with propositions which hold there. This is called a **Kripke structure**.

Definition: Kripke structure
Given a set $Prop$ of **propositions**, then a Kripke structure is given by $\mathrm{K} = \langle \Sigma, I, \rightarrow, V \rangle$ where
- Σ is a finite set of states,
- $I \subseteq \Sigma$ is a set of initial states,
- $\rightarrow \subseteq \Sigma \times \Sigma$ is a left-total transition relation, and
- $V: Prop \rightarrow 2^{\Sigma}$ is a valuation function mapping propositions to the set of states in which they hold

- Equivalent formulation: for each state, set of propositions which hold in this state, i.e. $V': \Sigma \rightarrow 2^{Prop}$

Kripke Structure: Example

- Example: Cooker
- Propositions:
  - Cooker is starting: S
  - Door is closed: C
  - Cooker is hot: H
  - Error occurred: E



- Kripke structure:
  - $\Sigma = \{s_1, \ldots, s_6\}$
  - $I = \{s_1\}$
  - $\rightarrow = \{(s_1, s_2), (s_2, s_5), (s_5, s_2), (s_1, s_3)$ $(s_3, s_1), (s_3, s_6), (s_6, s_4), (s_4, s_4),$ $(s_4, s_3), (s_4, s_1)\}$
  - $\mathrm{V(S)} = \{s_2, s_5, s_6\}$, $\mathrm{V(C)} = \{s_3, s_4, s_5, s_6\}$, $\mathrm{V(H)} = \{s_4\}$, $\mathrm{V(E)} = \{s_2, s_5\}$

Semantics of Kripke Structures (Prop)

- We now want to define a logic in which we can formalize temporal statements, i.e. statements about the behaviour of the system and its changes over time.
- The basis is **open propositional logic** (PL): negation, conjunction, disjunction, implication*.
- With that, we define how a PL-formula ϕ holds in a Kripke structure K at state s , written as $K, s \vDash \phi$.
- Let $K = \langle \Sigma, I, \rightarrow, V \rangle$ be a Kripke structure, $s \in \Sigma$, and ϕ a formula of propositional logic, then
  - $K, s \vDash p$ if $p \in Prop$ and $s \in V(p)$
  - $K, s \vDash \neg\phi$ if not $K, s \vDash \phi$
  - $K, s \vDash \phi_1 \wedge \phi_2$ if $K, s \vDash \phi_1$ and $K, s \vDash \phi_2$
  - $K, s \vDash \phi_1 \vee \phi_2$ if $K, s \vDash \phi_1$ or $K, s \vDash \phi_2$

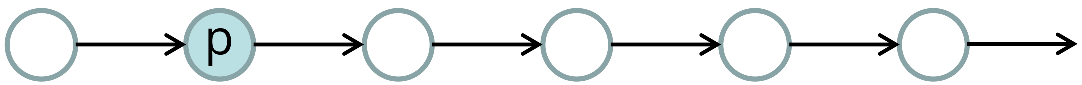
* Note implication is derived: $\phi_1 \rightarrow \phi_2 = \neg\phi_1 \vee \phi_2$

Linear Temporal Logic

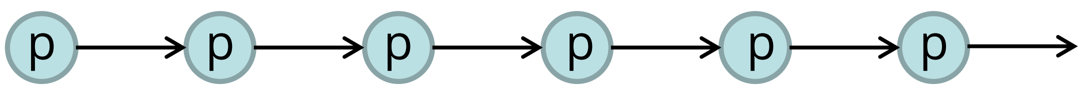
- The formulae of LTL are given as

$$\phi \ ::= \ p \mid \neg\, \phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \qquad \text{Propositional formulae}$$
$$\mid\ X\,\phi \mid G\,\phi \mid F\,\phi \mid \phi_1\, U\, \phi_2 \qquad \text{Temporal operators}$$

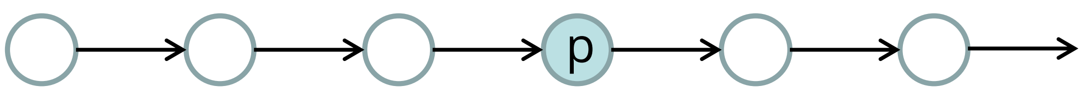
- X p: in the next moment p holds



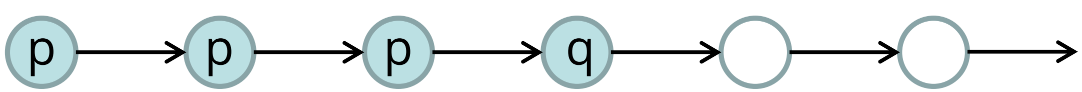
- G p: p holds in all moments



- F p: there is a moment in the future when p will hold



- p U q: p holds in all moments until q holds



Examples of LTL formulae

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

  $G(H \to C)$

- Is it always possible to recover from an error?

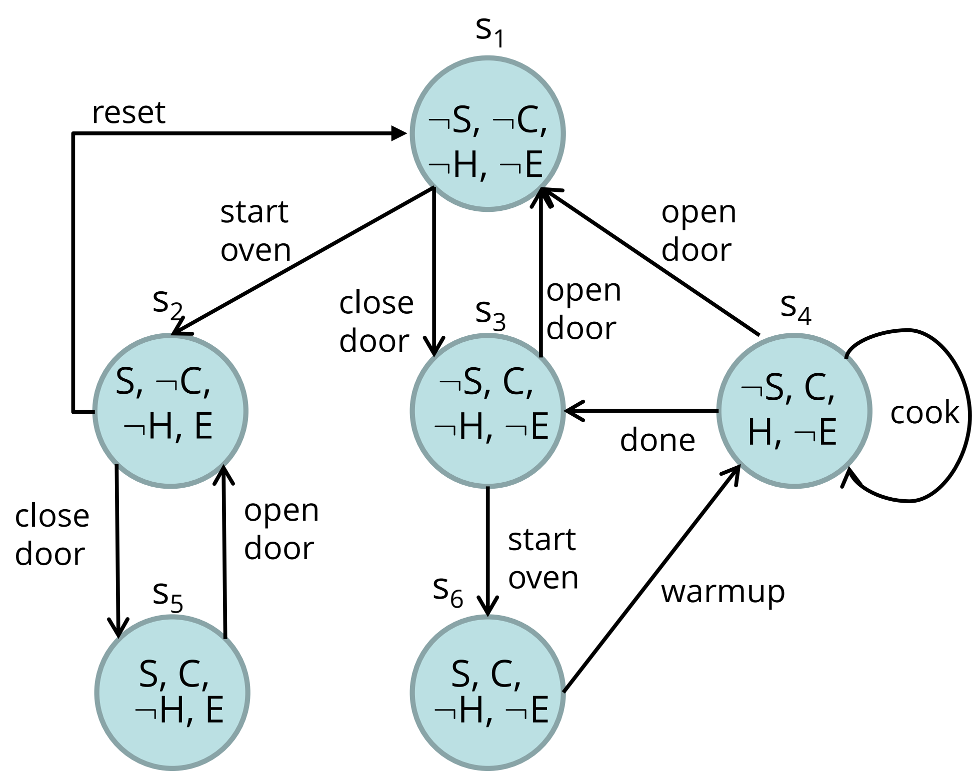
  $G\ (E \to F \neg E)$

  - Need to add a transition.

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

  $F\ (S\ \to X\ C)$

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



Paths in an FSM/Kripke Structure

- A **path** in an FSM (or Kripke structure) is a sequence of states starting in one of the initial states and connected by the transition relation (essentially, a **run** of the system).

- Formally: for an FSM $M = \langle \Sigma, I, \rightarrow \rangle$ or a Kripke structure $K = \langle \Sigma, I, \rightarrow, V \rangle$, a **path** is given by a sequence $s_1 s_2 s_3 \ldots \in \Sigma^*$ such that $s_1 \in I$ and $s_i \rightarrow s_{i+1}$.

- For a path $\mathrm{p} = s_1 s_2 s_3 \ldots$, we write
  - p_i for **selecting** the i-th element s_i and
  - p^i for the **suffix** starting at position i, $s_i s_{i+1} s_{i+2} \ldots$

Semantics of LTL in Kripke Structures

Let $K = \langle \Sigma, I, \rightarrow, V \rangle$ be a Kripke Structure and ϕ an LTL formula, then we say $K \vDash \phi$ (**ϕ holds in K**), if $K, s \vDash \phi$ for all paths $s = s_1 s_2 s_3 \ldots$ in K, where:

- $K, s \vDash p$ if $p \in Prop$, $s_1 \in V(p)$
- $K, s \vDash \neg\phi$ if not $K, s \vDash \phi$
- $K, s \vDash \phi_1 \wedge \phi_2$ if $K, s \vDash \phi_1$ and $K, s \vDash \phi_2$
- $K, s \vDash \phi_1 \vee \phi_2$ if $K, s \vDash \phi_1$ or $K, s \vDash \phi_2$

- $K, s \vDash X\,\phi$ if $K, s^2 \vDash \phi$
- $K, s \vDash G\,\phi$ if $K, s^n \vDash \phi$ for all $n > 0$
- $K, s \vDash F\,\phi$ if $K, s^n \vDash \phi$ for some $n > 0$
- $K, s \vDash \phi\, U\, \psi$ if $K, s^n \vDash \psi$ for some $n > 0$, and for all $i, 0 < i < n$, we have $K, s^i \vDash \phi$

More examples for the cooker

- Question: does the cooker work?
- Specifically, cooking means that first the door is open, then the oven heats up, cooks, then the door is open again, and all without an error.
  - $c = \neg C \wedge X(S \wedge X(H \wedge F \neg C)) \wedge G\, \neg E$ – not quite.
  - $c = (\neg C \wedge\ \neg E) \wedge\ X(S \wedge \neg E \wedge X(H \wedge \neg E \wedge F(\neg C \wedge\ \neg E)))$ – better

- So, does the cooker work?
  - There is at least one path s.t. c holds eventually.
  - This is **not** $\mathrm{G}\, F\, c$, which says that all paths must eventually cook (which might be too strong).
  - We cannot express this in LTL; this is a principal limitation.

Computational Tree Logic (CTL)

- LTL does not allow us the quantify over paths, e.g. assert the existence 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 whether $\neg\phi$ 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 another temporal logic 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 transition relation of the Kripke structure.

Computational Tree Logic (CTL)

- The formulae of **CTL** are given as

$$\phi \ ::= p \mid \neg \phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2$$

Propositional formulae

$$\mid\ AX\ \phi \mid EX\ \phi \mid AG\ \phi \mid EG\ \phi$$

Temporal operators

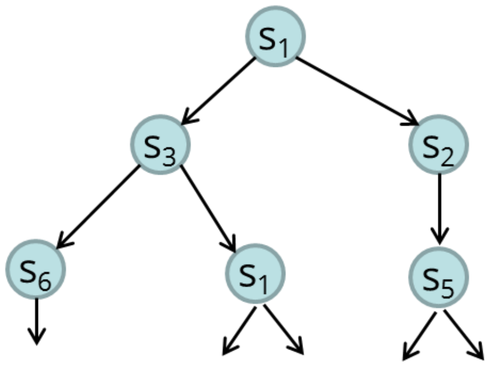
$$\mid\ AF\ \phi \mid EF\ \phi \mid \phi_1\ AU\ \phi_2 \mid \phi_1 EU\ \phi_2$$

- Note that CTL formulae can be considered to be a LTL formulae with a **modality** (*A* or *E*) added to each temporal operator.
  - Generally speaking, the *A* modality says the temporal operator holds for all paths, and the *E* modality says it only holds for all least one path.

- Hence, we do not define a **satisfaction** for a single path *p*, but with respect to a specific state in an FSM.

Computational Tree Logic (CTL)

- Specifying possible paths by combination
  - Branching behavior
    All paths: A, exists path: E
  - Succession of states in a path
    Temporal operators X, G, F, U



- For example:
  - AX p : in all paths the next state satisfies p
  - EX p : there is an path in which the next state satisfies p
  - p AU q : in all paths p holds as long as q does not hold
  - EF p : there is an path in which eventually p holds

Semantics of CTL in Kripke Structures

For a Kripke structure $K = \langle \Sigma, I, \rightarrow, V \rangle$ and a CTL-formula ϕ, we say $K \models \phi$ (**ϕ holds in K**) if $K, s \models \phi$ for all $s \in I$, where $K, s \models \phi$ is defined inductively as follows (omitting the clauses for propositional operators $p, \neg, \wedge, \vee$):

- $K, s \models AX\ \phi$ iff for all s' with $s \rightarrow s'$, we have $K, s' \models \phi$
- $K, s \models EX\ \phi$ iff for some s' with $s \rightarrow s'$, we have $K, s' \models \phi$
- $K, s \models AG\ \phi$ iff for all paths p with $p_1 = s$,
  we have $K, p_i \models \phi$ for all $i \geq 2$.
- $K, s \models EG\ \phi$ iff for some path p with $p_1 = s$,
  we have $K, p_i \models \phi$ for all $i \geq 2$.
- $K, s \models AF\ \phi$ iff for all paths p with $p_1 = s$,
  we have $K, p_i \models \phi$ for some i
- $K, s \models EF\ \phi$ iff for some path p with $p_1 = s$,
  we have $K, p_i \models \phi$ for some i
- $K, s \models \phi\ AU\ \psi$ iff for all paths p with $p_1 = s$,
  there is i with $K, p_i \models \psi$ and for all $j < i, K, p_j \models \phi$
- $K, s \models \phi\ EU\ \psi$ iff for some path p with $p_1 = s$,
  there is i with $K, p_i \models \psi$ and for all $j < i, K, p_j \models \phi$

Examples of CTL propositions

- If the cooker is hot, then is the door closed

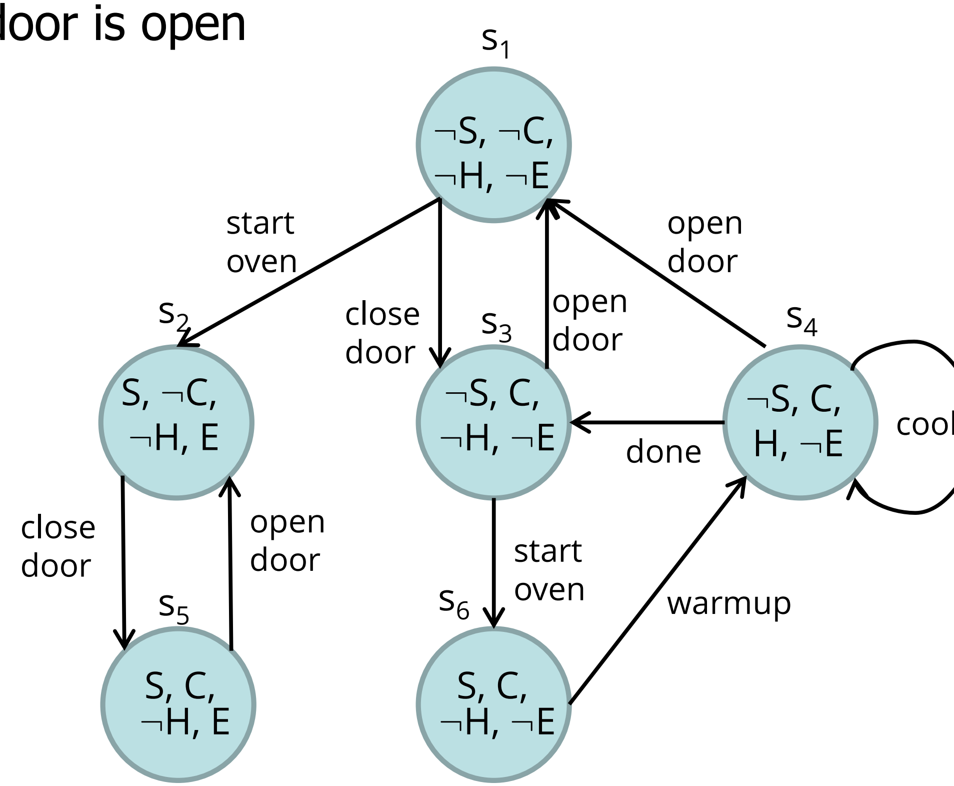
$$AG\ (H\ \rightarrow C)$$

- It is always possible to eventually cook (heat is on), and then eventually get the food (i.e. the door is open afterwards):

$$AF\ (H\ \rightarrow AF \neg\ C)$$

- It is always possible that the cooker will eventually warmup.

$$AG\ \big(EF\ (\neg H \wedge EX\ H)\big)$$



LTL, CTL and CTL*

- CTL is more expressive than LTL, but (surprisingly) there are also properties we can express in LTL but not in CTL:
  - The formula $(F\phi) \rightarrow F\psi$ cannot be expressed in CTL
    - "When ϕ occurs somewhere, then ψ also occurs somewhere."
    - Not: $(AF\phi) \rightarrow AF\psi$, nor $AG(\phi \rightarrow AF\ \psi)$
  - The formula $AG\ (EF\phi)$ cannot be expressed in LTL
    - "For all paths, it is always the case that there is some path on which ϕ is eventually true."
- CTL* - Allow for the use of temporal operators (X, G, F, U) without a directly preceding path quantifier (A, E)
  - e.g. AGF φ is allowed
- CTL* subsumes both LTL and CTL.

Complexity and State Explosion

- Even our small oven example has 6 states with 4 labels each. If we add one integer variable with 32 bits (e.g. for the heat), we get 2^{32} additional states.

- Theoretically, there is not much hope. The basic problem of deciding whether a formula holds (**satisfiability problem**) for the temporal logics we have seen has the following complexity:
  - LTL without U is NP-complete;
  - LTL is PSPACE-complete;
  - CTL (and CTL*) are EXPTIME-complete.

- This is known as **state explosion**.

- But at least it is **decidable**. Practically, state abstraction is the key technique, so e.g. for an integer variable i we identify all states with $i \leq 0$, and those with $0 < i$.

Safety and Liveness Properties

- Safety: nothing bad ever happens
  - E.g. "x is always not equal 0"
  - Safety properties are falsified by a bad (reachable) state
  - Safety properties can falsified by a finite prefix of an execution trace

- Liveness: something good will eventually happen
  - E.g. "system is always terminating"
  - Need to keep looking for the good thing forever
  - Liveness properties can be falsified by an infinite-suffix of an execution trace: e.g. finite list of states beginning with the initial state followed by a *cycle* showing you a loop that can cause you to get stuck and never reach the "good thing"

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**.
- Note difference to deductive verification (Floyd-Hoare logic): that uses the **source code** as the basis, here we need to construct a **model** of the system.
  - The model can be wrong – on the other hand we can construct the model and check properties before even building the system.
  - Model checking is **complementary** to deductive verification.
- 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, leading to so-called **state explosion.** But the use of abstraction and state compression techniques make model-checking bearable.
- Next week: tools for model checking.