Model Checking
Nontraditional use of nontraditional logic!

Checking whether a formula is satisfied in a finite domain.

Model: finite-state transition system

Logic: Propositional Temporal Logic.

Verification Procedure: exhaustively search of the state space to determine the truth of specification.
Why Model checking?
Why Model checking?
Why Model checking?

- Doesn’t aim too high!
Why Model checking?

- Doesn’t aim too high!

- Originally restricted to finite-state systems.
Why Model checking?

- Doesn’t aim too high!
- Originally restricted to finite-state systems.
- Applicable to systems with “short” descriptions.
Why Model checking?

- Doesn’t aim too high!
  - Originally restricted to finite-state systems.
  - Applicable to systems with “short” descriptions.
  - Control-oriented systems such as hardware, protocols, ...Can we come up with them automatically?
Why Model checking?

- Doesn’t aim too high!
  - Originally restricted to finite-state systems.
  - Applicable to systems with “short” descriptions.
  - Control-oriented systems such as hardware, protocols, ...Can we come up with them automatically?
- Fully automatic with low computational complexity.
Why Model checking?

- Doesn’t aim too high!
  - Originally restricted to finite-state systems.
  - applicable to systems with “short” descriptions.
  - control-oriented systems such as hardware, protocols, ...Can we come up with them automatically?
- Fully automatic with low computational complexity.
- Can be viewed as an elaborate debugging tool: counterexamples.
First Step:
We need a formal model!
A transition system \( TS \) is a tuple \((S, \text{Act}, \rightarrow, I, \text{AP}, L)\) where

- \( S \) is a set of states,
- \( \text{Act} \) is a set of actions,
- \( \rightarrow \subseteq S \times \text{Act} \times S \) is a transition relation,
- \( I \subseteq S \) is a set of initial states,
- \( \text{AP} \) is a set of atomic propositions, and
- \( L : S \rightarrow 2^{\text{AP}} \) is a labeling function.

\( TS \) is called finite if \( S, \text{Act}, \) and \( \text{AP} \) are finite.
Example

The labeling function $L$ relates a set $L(s) \subseteq 2^{AP}$ of atomic propositions to any state $s$. Intuitively, $L(s)$ stands for exactly those propositions $a \in AP$ which are satisfied by state $s$. Given that $\Phi$ is a propositional logic formula, then $s$ satisfies the formula $\Phi$ if the evaluation induced by $L(s)$ makes the formula $\Phi$ true; that is:

$$s \models \Phi \text{ if } L(s) \models \Phi.$$ (Basic principles of propositional logic are explained in Appendix A.3, see page 915 ff.)

Example 2.2. Beverage Vending Machine

We consider an (somewhat foolish) example, which has been established as standard in the field of process calculi. The transition system in Figure 2.1 models a preliminary design of a beverage vending machine. The machine can either deliver beer or soda. States are represented by ovals and transitions by labeled edges. State names are depicted inside the ovals. Initial states are indicated by having an incoming arrow without source.

The state space is $S = \{\text{pay}, \text{select}, \text{soda}, \text{beer}\}$. The set of initial states consists of only one state, i.e., $I = \{\text{pay}\}$. The (user) action $\text{insert\_coin}$ denotes the insertion of a coin, while the (machine) actions $\text{get\_soda}$ and $\text{get\_beer}$ denote the delivery of soda and beer, respectively. Transitions of which the action label is not of further interest here, e.g., as it denotes some internal activity of the beverage machine, are all denoted by the distinguished action symbol $\tau$. We have:

$$\text{Act} = \{\text{insert\_coin}, \text{get\_soda}, \text{get\_beer}, \tau\}.$$ Some example transitions are:

- $\text{pay} \xrightarrow{\text{insert\_coin}} \text{select}$
- $\text{get\_beer} \xrightarrow{\tau} \text{pay}$
Example

Figure 2.3: Transition system modeling the extended beverage vending machine.
Second Step: We need a formal Specification!
There are 5 philosophers at a table sharing 5 chopsticks for eating.

Each philosopher needs two chopsticks to eat.

At each point in time at most one of two neighbouring philosophers can eat.
There are 5 philosophers at a table sharing 5 chopsticks for eating. Each philosopher needs two chopsticks to eat. At each point in time at most one of two neighbouring philosophers can eat. Classic deadlock scenario example!
Reachability

Problem: given an TS, and a target set T, is T reachable from Q₀.

Solution?
Reachability

Problem: given an TS, and a target set T, is T reachable from Q₀.

Solution? Depth First Search, in O(n+m) time.
Reachability

Problem: given an TS, and a target set T, is T reachable from Q_0.

Solution? Depth First Search, in $O(n+m)$ time.

DFS(q)

Add q to visited_states;
for each q’ such that q -a-> q’
   if q’ in T
      print "YES!"; halt;
   else if q’ not in visited_states
      DFS(q’)

Reachability

Problem: given an TS, and a target set T, is T reachable from Q₀.

Solution? Depth First Search, in O(n+m) time.
Reachability

Problem: given an TS, and a target set T, is T reachable from Q₀.

Solution? Depth First Search, in O(n+m) time.

What if we are interested in more sophisticated properties?
Reachability

Problem: given an TS, and a target set T, is T reachable from Q₀.

Solution? Depth First Search, in O(n+m) time.

What if we are interested in more sophisticated properties?

Suggest a non-reachability property for philosophers!
Reachability

Problem: given an TS, and a target set T, is T reachable from Q₀.

Solution? Depth First Search, in $O(n+m)$ time.

What if we are interested in more sophisticated properties?

Suggest a non-reachability property for philosophers!
Reachability

Problem: given an TS, and a target set T, is T reachable from \( Q_0 \).

Solution? Depth First Search, in \( O(n+m) \) time.

What if we are interested in more sophisticated properties?

Suggest a non-reachability property for philosophers!

The light will always eventually turn green.
Option 1 for properties beyond reachability ...
One TS as a Spec for Another TS!

Given a TS $M$ for the model and a TS $S$ for the specification:
One TS as a Spec for Another TS!

Given a TS $M$ for the model and a TS $S$ for the specification:

Question: Is every behaviour of $M$ a behaviour of $S$?
One TS as a Spec for Another TS!

Given a TS $M$ for the model and a TS $S$ for the specification:

Question: Is every behaviour of $M$ a behaviour of $S$?

$$L(M) \subseteq L(S)$$
One TS as a Spec for Another TS!

Given a TS $M$ for the model and a TS $S$ for the specification:

Question: Is every behaviour of $M$ a behaviour of $S$?

\[ L(M) \subseteq L(S) \]

Solvable in PSpace: linear in $M$ and exponential in $S$. 
Best choice: new logic!
Alternative: Temporal Logic

- Language for describing properties of infinite sequences.
- Extension of propositional logic.
- Uses temporal operators to describe sequencing properties.
Linear Temporal Logic
LTL Syntax

ϕ ::= true | a | ϕ₁ ∧ ϕ₂ | ¬ϕ | ♦ϕ | ϕ₁ U ϕ₂
Figure 5.1 sketches the intuitive meaning of temporal modalities for the simple case in

We mostly abstain from explicitly indicating the set of propositions as this follows

<table>
<thead>
<tr>
<th>LTL Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varphi ::= \text{true} \mid a \mid \varphi_1 \land \varphi_2 \mid \neg \varphi \mid \Box \varphi \mid \varphi_1 U \varphi_2 )</td>
</tr>
</tbody>
</table>

\( a \in AP \)

\( \diamond \varphi \overset{\text{def}}{=} \text{true } U \varphi \)

\( \square \varphi \overset{\text{def}}{=} \neg \diamond \neg \varphi \)
LTL: Intuition
Before proceeding with the formal semantics of LTL, we present some examples.

**Example 5.2. Properties for the Mutual Exclusion Problem**

Consider the mutual exclusion problem for two concurrent processes $P_1$ and $P_2$, say. Process $P_i$ is modeled by three locations: (1) the noncritical section, (2) the waiting phase which is entered when the process intends to enter the critical section, and (3) the critical section. Let the propositions $\text{wait}_i$ and $\text{crit}_i$ denote that process $P_i$ is in its waiting phase and critical section, respectively.

The safety property stating that $P_1$ and $P_2$ never simultaneously have access to their critical sections can be described by the LTL-formula:

$$\square (\neg \text{crit}_1 \lor \neg \text{crit}_2).$$

This formula expresses that always $\square$ at least one of the two processes is not in its critical section $\neg \text{crit}_i$.

The liveness requirement stating that each process $P_i$ is infinitely often in its critical section can be described by the LTL-formula:

$$\diamond \text{crit}_i.$$
LTL Semantics

LTL is interpreted over paths.

These paths are (infinite) words labeled with subset of the atomic propositions (AP) that are true at each letter.
LTL Semantics

LTL is interpreted over paths.

These paths are (infinite) words labeled with subset of the atomic propositions (AP) that are true at each letter.

\[
\begin{align*}
\sigma & \models \text{true} \\
\sigma & \models a \quad \text{iff} \quad a \in A_0 \quad \text{(i.e., } A_0 \models a) \\
\sigma & \models \varphi_1 \land \varphi_2 \quad \text{iff} \quad \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \\
\sigma & \models \neg \varphi \quad \text{iff} \quad \sigma \not\models \varphi \\
\sigma & \models \bigcirc \varphi \quad \text{iff} \quad \sigma[1\ldots] = A_1A_2A_3\ldots \models \varphi \\
\sigma & \models \varphi_1 \mathsf{U} \varphi_2 \quad \text{iff} \quad \exists j \geq 0. \ \sigma[j\ldots] \models \varphi_2 \text{ and } \sigma[i\ldots] \models \varphi_1, \text{ for all } 0 \leq i < j
\end{align*}
\]
LTL Semantics

LTL is interpreted over paths.

These paths are (infinite) words labeled with subset of the atomic propositions (AP) that are true at each letter.

\[ \sigma \models \text{true} \]

\[ \sigma \models a \quad \text{iff} \quad a \in A_0 \quad (\text{i.e., } A_0 \models a) \]

\[ \sigma \models \varphi_1 \land \varphi_2 \quad \text{iff} \quad \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \]

\[ \sigma \models \neg \varphi \quad \text{iff} \quad \sigma \not\models \varphi \]

\[ \sigma \models \bigcirc \varphi \quad \text{iff} \quad \sigma[1...] = A_1A_2A_3... \models \varphi \]

\[ \sigma \models \varphi_1 U \varphi_2 \quad \text{iff} \quad \exists j \geq 0. \ \sigma[j...] \models \varphi_2 \text{ and } \sigma[i...] \models \varphi_1, \text{ for all } 0 \leq i < j \]

LTL’s \models is the smallest relation satisfying the above rules.
The semantics of the combinations of
statement for
be regarded in order to be able to refer to the truth-value of the subformula
for the second statement is similar.

As a subsequent step, we determine the semantics of LTL-formulae with respect to a

As a subsequent step, we determine the semantics of LTL-formulae with respect to a

For the derived operators
and

means

In the

In the

In the

In the

...
The semantics of the combinations of statement for the second statement is similar. As a subsequent step, we determine the semantics of LTL-formulae with respect to a satisfaction relation.

\[
\begin{align*}
\sigma & \models \Diamond \varphi \iff \exists j \geq 0. \sigma[j...] \models \varphi \\
\sigma & \models \square \varphi \iff \forall j \geq 0. \sigma[j...] \models \varphi
\end{align*}
\]

\[
\begin{align*}
\sigma & \models \square \Diamond \varphi \iff \\
\sigma & \models \Diamond \square \varphi \iff
\end{align*}
\]
\( \sigma \models \lozenge \varphi \) iff \( \exists j \geq 0. \sigma[j\ldots] \models \varphi \)

\( \sigma \models \Box \varphi \) iff \( \forall j \geq 0. \sigma[j\ldots] \models \varphi \)

\( \sigma \models \lozenge \Box \varphi \) iff \( \exists j. \sigma[j\ldots] \models \varphi \)

\( \sigma \models \Box \lozenge \varphi \) iff \( \forall j. \sigma[j\ldots] \models \varphi \)
Examples

mutual exclusion:
Before proceeding with the formal semantics of LTL, we present some examples.

Example 5.2. Properties for the Mutual Exclusion Problem

Consider the mutual exclusion problem for two concurrent processes $P_1$ and $P_2$. Process $P_i$ is modeled by three locations: (1) the noncritical section, (2) the waiting phase which is entered when the process intends to enter the critical section, and (3) the critical section. Let the propositions $\text{wait}_i$ and $\text{crit}_i$ denote that process $P_i$ is in its waiting phase and critical section, respectively.

The safety property stating that $P_1$ and $P_2$ never simultaneously have access to their critical sections can be described by the LTL-formula:

$$\Box\left( \neg \text{crit}_1 \lor \neg \text{crit}_2 \right)$$

This formula expresses that always ($\Box$) at least one of the two processes is not in its critical section ($\neg \text{crit}_i$).
mutual exclusion:

\[ \Box (\neg \text{crit}_1 \lor \neg \text{crit}_2) \]

once red, the light cannot become green immediately:
Examples

mutual exclusion:

\( \Box ( \neg \text{crit}_1 \lor \neg \text{crit}_2 ) \)

once red, the light cannot become green immediately:

\( \Box (\text{red} \rightarrow \neg \Diamond \text{green}) \)
Examples

- mutual exclusion:
  \[ \Box (\neg crit_1 \lor \neg crit_2) \]

- once red, the light cannot become green immediately:
  \[ \Box (red \rightarrow \neg \Box green) \]

- every request will eventually lead to a response:
Examples

mutual exclusion:
\[ \Box (\neg \text{crit}_1 \lor \neg \text{crit}_2) \]

once red, the light cannot become green immediately:
\[ \Box (\text{red} \rightarrow \neg \Diamond \text{green}) \]

every request will eventually lead to a response:
\[ \Box (\text{request} \rightarrow \Diamond \text{response}) \]
Examples

mutual exclusion: \[\Box(\neg \text{crit}_1 \lor \neg \text{crit}_2)\]

once red, the light cannot become green immediately: \[\Box(\text{red} \rightarrow \neg \Diamond \text{green})\]

every request will eventually lead to a response: \[\Box(\text{request} \rightarrow \Diamond \text{response})\]

once red, the light always becomes green eventually after being yellow for some time:
Examples

mutual exclusion:

\[ \square (\neg \text{crit}_1 \lor \neg \text{crit}_2) \]

once red, the light cannot become green immediately:

\[ \square (\text{red} \rightarrow \neg \lozenge \text{green}) \]

every request will eventually lead to a response:

\[ \square (\text{request} \rightarrow \lozenge \text{response}) \]

once red, the light always becomes green eventually after being yellow for some time:

\[ \square (\text{red} \rightarrow \lozenge (\text{red} \lor \text{yellow} \land \lozenge (\text{yellow} \lor \text{green})))) \]