### Topic

The CTL model-checking problem:
Given:

- A finite transition system $TS$
- CTL state-formula $\Phi$

Decide whether $TS \models \Phi$, and if $TS \not\models \Phi$ provide a counterexample.\(^1\)

---

\(^1\)CTL counterexamples are outside the scope of this course.

---

### CTL Syntax

**Definition: Syntax Computation Tree Logic**

- CTL state-formulas with $a \in AP$ obey the grammar:

\[
\Phi ::= \text{true} \mid a \mid \Phi_1 \land \Phi_2 \mid \neg \Phi \mid \exists \varphi \mid \forall \varphi
\]

and $\varphi$ is a path-formula formed by the grammar:

\[
\varphi ::= \Diamond \Phi \mid \Phi_1 \mathsf{U} \Phi_2.
\]

**Examples**

- $\forall \Diamond \exists a$ and $\exists (\forall a) \mathsf{U} b$ are CTL formulas.

**Intuition**

- $s \not\models \forall \varphi$ if all paths starting in $s$ fulfill $\varphi$
- $s \models \exists \varphi$ if some path starting in $s$ fulfill $\varphi$
Intuitive CTL Semantics

Overview

- CTL Semantics
- Existential Normal Form
- Basic CTL Model-Checking Algorithm
- Model Checking EU and $\exists \Box$
- Complexity Considerations
- Summary

CTL Semantics

Define a satisfaction relation for CTL-formulas over $AP$ for a given transition system $TS$ without terminal states.

Two parts:

- Interpretation of state-formulas over states of $TS$
- Interpretation of path-formulas over paths of $TS$
CTL Semantics (1)

**Notation**

$TS, s \models \Phi$ if and only if state-formula $\Phi$ holds in state $s$ of transition system $TS$. As $TS$ is known from the context we simply write $s \models \Phi$.

**Definition: Satisfaction relation for CTL state-formulas**

The satisfaction relation $\models$ is defined for CTL state-formulas by:

- $s \models a$ iff $a \in L(s)$
- $s \models \neg \Phi$ iff not ($s \models \Phi$)
- $s \models \Phi \land \Psi$ iff ($s \models \Phi$) and ($s \models \Psi$)
- $s \models \exists \varphi$ iff there exists $\pi \in \text{Paths}(s)$. $\pi \models \varphi$
- $s \models \forall \varphi$ iff for all $\pi \in \text{Paths}(s)$. $\pi \models \varphi$

where the semantics of CTL path-formulas is defined on the next slide.

Transition System Semantics

- For CTL-state-formula $\Phi$, the satisfaction set $\text{Sat}(\Phi)$ is defined by:
  \[
  \text{Sat}(\Phi) = \{ s \in S \mid s \models \Phi \}
  \]

- $TS$ satisfies CTL-formula $\Phi$ iff $\Phi$ holds in all its initial states:
  \[
  TS \models \Phi \text{ if and only if } \forall s_0 \in I. s_0 \models \Phi
  \]

- Point of attention: $TS \not\models \Phi$ is not equivalent to $TS \not\models \neg \Phi$
  because of several initial states, e.g., $s_0 \models \exists \Box \varphi$ and $s'_0 \not\models \exists \Box \varphi$.

CTL Semantics (2)

**Definition: Satisfaction relation for CTL path-formulas**

Given path $\pi$ and CTL path-formula $\varphi$, the satisfaction relation $\models$ where $\pi \models \varphi$ if and only if path $\pi$ satisfies $\varphi$ is defined as follows:

\[
\begin{align*}
\pi \models \Box \Phi & \iff \pi[1] \models \Phi \\
\pi \models \Phi \lor \Psi & \iff (\exists j \geq 0. \pi[j] \models \Psi \text{ and } (\forall 0 \leq i < j. \pi[i] \not\models \Phi))
\end{align*}
\]

where $\pi[j]$ denotes the state $s_i$ in the path $\pi = s_0 s_1 s_2 \ldots$.

Overview

- CTL Semantics
- Existential Normal Form
- Basic CTL Model-Checking Algorithm
- Model Checking EU and $\exists \Box$
- Complexity Considerations
- Summary
Existential Normal Form

Definition: existential normal form

A CTL formula is in existential normal form (ENF) if it is of the form:

\[ \Phi : a \mid \Phi_1 \land \Phi_2 \mid \neg \Phi \mid \exists \Phi_1 U \Phi_2 \mid \exists \Phi \]

Only existentially quantified temporal modalities \( \Box, U \) and \( \square \).

For each CTL formula, there exists an equivalent CTL formula in ENF.

Proof.

Universally quantified temporal modalities can be transformed as follows:

\[
\forall \Box \Phi \equiv \neg \exists \neg \Box \neg \Phi \\
\forall (\Phi U \Psi) \equiv \neg \exists (\neg \Psi U (\neg \Phi \land \neg \Psi)) \land \neg \exists \neg \Box \neg \Psi
\]

Basic Idea

- How to check whether TS satisfies CTL formula \( \Psi \)?
  - convert the formula \( \Psi \) into the equivalent \( \Phi \) in ENF
  - compute recursively the set \( \text{Sat}(\Phi) = \{ s \in S \mid s \vDash \Phi \} \)
  - \( TS \vDash \Phi \) if and only if each initial state of TS belongs to \( \text{Sat}(\Phi) \)

- Recursive bottom-up computation of \( \text{Sat}(\Phi) \):
  - consider the parse tree of \( \Phi \)
  - start to compute \( \text{Sat}(a_i) \), for all leafs in the parse tree
  - then go one level up in the tree and determine \( \text{Sat}(\cdot) \) for these nodes
    - e.g., \( \text{Sat}(\Phi_1 U \Phi_2) = \text{Sat}(\Phi_1) \cap \text{Sat}(\Phi_2) \)
  - then go one level up and determine \( \text{Sat}(\cdot) \) of these nodes
  - and so on....... until the root is treated, i.e., \( \text{Sat}(\Phi) \) is computed

- Check whether \( I \subseteq \text{Sat}(\Phi) \).

Basic Algorithm
Basic Algorithm

\[
\begin{align*}
\text{Sat}(&\text{true}) = S \\
\text{Sat}(a) & = \{ s \in S \mid a \in L(s) \} \\
\text{Sat}(\neg \Phi) & = S \setminus \text{Sat}(\Phi) \\
\text{Sat}(\Phi \land \Psi) & = \text{Sat}(\Phi) \cap \text{Sat}(\Psi) \\
\text{Sat}(\exists \Diamond \Phi) & = \{ s \in S \mid \text{Post}(s) \cap \text{Sat}(\Phi) \neq \emptyset \} \\
\text{Sat}(\exists \Box \Phi) & = \ldots \\
\text{Sat}(\exists (\Phi U \Psi)) & = \ldots \\
\end{align*}
\]

Treatment of \(\exists \Diamond \Phi\) and \(\exists (\Phi U \Psi)\): via a fixed-point computation

Characterization of \(\text{Sat}\) for EU

Expansion law:

\[
\exists (\Phi U \Psi) \equiv \Psi \lor (\Phi \land \exists \Diamond (\Phi U \Psi))
\]

In fact, \(\exists (\Phi U \Psi)\) is the smallest solution of this recursive equation

\(\text{Sat}(\exists (\Phi U \Psi))\) is the smallest subset \(T\) of \(S\), such that:

1. \(\text{Sat}(\Psi) \subseteq T\) and 2. \(s \in \text{Sat}(\Phi) \land \text{Post}(s) \cap T \neq \emptyset \) \(\Rightarrow s \in T\).

That is, \(T = \text{Sat}(\exists (\Phi U \Psi))\) is the smallest fixed point of the (higher-order) function \(\Omega : 2^S \to 2^S\) given by:

\[
\Omega(T) = \text{Sat}(\Psi) \cap \{ s \in \text{Sat}(\Phi) \mid \text{Post}(s) \cap T \neq \emptyset \}.
\]
Characterization of $\text{Sat}$ for $\exists \Box$

Expansion law:

$$\exists \Box \Phi \equiv \Phi \land \exists \Box \exists \Box \Phi$$

In fact, $\exists \Box \Phi$ is the largest solution of this recursive equation

$\text{Sat}(\exists \Box \Phi)$ is the largest subset $V$ of $S$, such that:

1. $V \subseteq \text{Sat}(\Phi)$ and 2. $s \in V$ implies $\text{Post}(s) \cap V \neq \emptyset$.

That is, $V = \text{Sat}(\exists \Box \Phi)$ is the largest fixed point of the (higher-order) function $\Omega : 2^S \rightarrow 2^S$ given by:

$$\Omega(V) = \{ s \in \text{Sat}(\Phi) \mid \text{Post}(s) \cap V \neq \emptyset \}$$

Universally Quantified Formulas

- $\text{Sat}(\forall \Box \Phi) = \{ s \in S \mid \text{Post}(s) \subseteq \text{Sat}(\Phi) \}$
- $\text{Sat}(\forall \Box \Phi)$ equals the largest set $T$ of states such that:
  $$T \subseteq \{ s \in \text{Sat}(\Phi) \mid \text{Post}(s) \subseteq T \}$$
- $\text{Sat}(\forall (\Phi \cup \Psi))$ is the smallest set $T$ of states such that:
  $$\text{Sat}(\Psi) \cup \{ s \in \text{Sat}(\Phi) \mid \text{Post}(s) \subseteq T \} \subseteq T$$

Example

- $V = \{ s_0 \}$ satisfies the condition
- $V \subseteq \{ s \in \text{Sat}(\Phi) \mid \text{Post}(s) \cap V \neq \emptyset \}$
- but $V \subset \text{Sat}(\exists \Box a) = \{ s_0, s_1 \}$

Model Checking $\text{EU}$

$\text{Sat}(\exists (\Phi \cup \Psi))$ is the smallest subset $T$ of $S$, such that:

1. $\text{Sat}(\Psi) \subseteq T$ and 2. $s \in \text{Sat}(\Phi)$ and $\text{Post}(s) \cap T \neq \emptyset$ $\Rightarrow$ $s \in T$.

- This suggests to compute $\text{Sat}(\exists (\Phi \cup \Psi))$ iteratively:
  $$T_0 = \text{Sat}(\Psi) \text{ and } T_{i+1} = T_i \cup \{ s \in \text{Sat}(\Phi) \mid \text{Post}(s) \cap T_i \neq \emptyset \}$$
- $T_i$ = states that can reach a $\Psi$-state in at most $i$ steps via $\Phi$ states
- By induction it follows:
  $$T_0 \subseteq T_1 \subseteq \ldots \subseteq T_j \subseteq T_{j+1} \subseteq \ldots \subseteq \text{Sat}(\exists (\Phi \cup \Psi))$$
- As $TS$ is finite, we have $T_{k+1} = T_k = \text{Sat}(\exists (\Phi \cup \Psi))$ for some $k$. 

ixo Pieter Katoen  Lecture#11 21/44  Joost-Pieter Katoen  Lecture#11 22/44
Model Checking EU in Pictures

Example

Algorithm

\[
\begin{align*}
T & := \text{Sat}(\Phi_2) \quad \text{collects all states } s = \exists(\Phi_1 \cup \Phi_2) \\
E & := \text{Sat}(\Phi_2) \quad \text{set of states still to be expanded} \\
\text{WHILE } E \neq \emptyset \text{ DO} \\
& \quad \text{select a state } s' \in E \text{ and remove } s' \text{ from } E \\
& \quad \text{FOR ALL } s \in \text{Pre}(s') \text{ DO} \\
& \quad \quad \text{IF } s \in \text{Sat}(\Phi_1) \setminus T \text{ THEN add } s \text{ to } T \text{ and } E \text{ FI} \\
& \quad \text{OD} \\
\text{return } T
\end{align*}
\]

Compute Sat(∃Φ U Ψ) by a linear-time enumerative backward search

Model Checking ∃□

Sat(∃□Φ) is the largest subset \( V \) of \( S \), such that:

1. \( V \subseteq \text{Sat}(\Phi) \) and
2. \( s \in V \) implies \( \text{Post}(s) \cap V \neq \emptyset \).

- This suggests to compute Sat(∃□Φ) iteratively:
  \[ V_0 = \text{Sat}(\Phi) \quad \text{and} \quad V_{i+1} = \{ s \in T_i \mid \text{Post}(s) \cap V_i \neq \emptyset \} \]
- \( V_i \) = states that have some \( \Phi \)-path of at least \( i \) transitions
- By induction it follows:
  \[ V_0 \supseteq V_1 \supseteq ... \supseteq V_j \supseteq ... \supseteq \text{Sat}(∃□Φ) \]
- As \( TS \) is finite, we have \( V_{k+1} = V_k = \text{Sat}(∃□Φ) \) for some \( k \).
Algorithm

\( T := \text{Sat}(\phi) \leftarrow \text{organizes the candidates for } s = 3\Box \phi \)
\( E := S \setminus T \leftarrow \text{set of states to be expanded} \)

WHILE \( E \neq \emptyset \) DO
pick a state \( s' \in E \) and remove \( s' \) from \( E \)
FOR ALL \( s \in \text{Pre}(s') \) DO
IF \( s \in T \) and \( \text{Post}(s) \cap T = \emptyset \) THEN
remove \( s \) from \( T \) and add \( s \) to \( E \)
FI
OD
return \( T \)

naive implementation: quadratic time complexity

Linear-Time Algorithm

\( T := \text{Sat}(\phi) \leftarrow \text{organizes the candidates for } s = 3\Box \phi \)
\( E := S \setminus T \leftarrow \text{set of states to be expanded} \)

WHILE \( E \neq \emptyset \) DO
pick a state \( s' \in E \) and remove \( s' \) from \( E \)
FOR ALL \( s \in \text{Pre}(s') \) DO
IF \( s \in T \) and \( \text{Post}(s) \cap (T \cup E) = \emptyset \) THEN
remove \( s \) from \( T \) and add \( s \) to \( E \)
FI
OD
return \( T \)

linear time implementation: uses counters \( c[s] \) for \( |\text{Post}(s) \cap (T \cup E)| \)

Linear-Time Algorithm Using Counters

\( T := \text{Sat}(\phi); E := S \setminus T \)
FOR ALL \( s \in \text{Sat}(\phi) \) DO \( c[s] := |\text{Post}(s)| \) OD

loop invariant: \( c[s] = |\text{Post}(s) \cap (T \cup E)| \) for \( s \in T \)

WHILE \( E \neq \emptyset \) DO
pick a state \( s' \in E \) and remove \( s' \) from \( E \)
FOR ALL \( s \in \text{Pre}(s') \) DO
IF \( s \in T \) THEN
\( c[s] := c[s] - 1 \)
IF \( c[s] = 0 \) THEN
remove \( s \) from \( T \) and add \( s \) to \( E \)
FI
OD

Example

Compute \( \text{Sat}(3\Box\phi) \) by a linear-time enumerative backward search
**An Alternative SCC-Based Algorithm**

An SCC-based algorithm for determining $\text{Sat}(\exists\Box \Phi)$:

1. Eliminate all states $s \notin \text{Sat}(\Phi)$:
   - determine $TS(\Phi) = (S', Act, \rightarrow', I', AP, L')$ with $S' = \text{Sat}(\Phi)$, $\rightarrow' = \rightarrow \cap (S' \times Act \times S')$, $I' = I \cap S'$, and $L'(s) = L(s)$ for $s \in S'$
   - Why? all removed states refute $\exists\Box \Phi$ and thus can be safely removed

2. Determine all non-trivial strongly connected components in $TS(\Phi)$
   - non-trivial SCC = maximal, connected sub-graph with $> 0$ transition
     ⇒ any state in such SCC satisfies $\exists\Box \Phi$

3. $s \models \exists\Box \Phi$ is equivalent to "an SCC in $TS(\Phi)$ is reachable from $s$"
   - this search can be done in a backward manner in linear time

---

**CTL Model-Checking Algorithm**

- $\text{Sat}(\text{true}) = S$
- $\text{Sat}(a) = \{ s \in S | a \in L(s) \}$
- $\text{Sat}(\neg \Phi) = S \setminus \text{Sat}(\Phi)$
- $\text{Sat}(\Phi \land \psi) = \text{Sat}(\Phi) \cap \text{Sat}(\psi)$
- $\text{Sat}(\exists\Box \Phi) = \{ s \in S | \text{Post}(s) \cap \text{Sat}(\Phi) \neq \emptyset \}$
- $\text{Sat}(\exists\diamond \Phi) = \bigcap_{n \geq 0} V_n$ where
  - $V_0 = \text{Sat}(\Phi)$
  - $V_{n+1} = \{ s \in T_n | \text{Post}(s) \cap V_n \neq \emptyset \}$
- $\text{Sat}(\exists(\Phi \lor \psi)) = \bigcup_{n \geq 0} T_n$ where
  - $T_0 = \text{Sat}(\psi)$
  - $T_{n+1} = T_n \cup \{ s \in \text{Sat}(\Phi) | \text{Post}(s) \cap T_n \neq \emptyset \}$

---

**Example**

Determining $\text{Sat}(\exists\Box q)$ using the SCC-based algorithm

---

**Overview**

- CTL Semantics
- Existential Normal Form
- Basic CTL Model-Checking Algorithm
- Model Checking EU and $\exists\Box$
- Complexity Considerations
- Summary
**Complexity Considerations**

**Time Complexity**

The CTL model-checking problem can be solved in $O(|\Phi| \cdot |TS|)$.

**Proof.**

1. The parse tree of $\Phi$ has size $O(|\Phi|)$
2. The time complexity at a node of the parse tree is in $O(|TS|)$
3. This holds in particular for computing $\text{Sat}(\exists U)$ and $\text{Sat}(\exists \Box \ldots)$
4. The entire time complexity is thus in $O(|\Phi| \cdot |TS|)$

**Complexity of CTL Model-Checking Problem**

The CTL model-checking problem is PTIME-complete.

**Proof.**

**CTL vs. LTL Model Checking**

LTL model checking is PSPACE-complete

CTL model checking is PTIME-complete.

Take a property that can be expressed in both LTL and CTL

Is CTL model checking more efficient? No!

LTL-formulae can be exponentially shorter than their CTL-equivalent

**CTL Versus LTL**

If $\Phi$ is equivalent to some LTL-formula $\varphi$ then:

$\Phi \equiv \varphi$ where $\varphi$ is obtained by removing all path quantifiers from $\Phi$.

In particular, $|\varphi| \leq |\Phi|$.

If $P \neq NP$, then there is a sequence $\varphi_n$, $n \geq 0$ of LTL formulas such that:

- $|\varphi_n|$ is polynomial in $n$
- $\varphi_n$ has an equivalent CTL formula
- no CTL formula of polynomial length is equivalent to $\varphi_n$

**Proof.**

Take $\varphi_n =$ the absence of a Hamiltonian path in a digraph on $n$ vertices
Complexity Considerations

LTL Encoding the Hamiltonian Path Problem

All $n!$ possibilities need to be explicitly enumerated

Suppose there is a CTL-formula of polynomial length equivalent to $\varphi_n$. Then: as CTL model-checking is $\in P$, the Hamiltonian path problem $\in P$, and $P = NP$.

Summary

▶ CTL model checking determines $Sat(\phi)$ by a recursive descent over $\phi$

▶ $Sat(\exists(\phi U \psi))$ is approximated from below by a backward search from $\psi$-states

▶ $\exists\square\phi$ is approximated from above by a backward search from $\phi$-states

▶ The CTL model-checking algorithm is linear in the size of $TS$ and $\phi$

▶ The CTL model-checking problem is PTIME-complete