Model Checking

Lecture #10: CTL Model Checking

[Baier & Katoen, Chapter 6.4]

Joost-Pieter Katoen

Software Modeling and Verification Group

Model Checking Course, RWTH Aachen, WiSe 2019/2020

Overview

Topic

The CTL model-checking problem:
Given:

- A finite transition system TS
- CTL state-formula Φ

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:

$$\Phi ::= \text{true} \;\big|\; a \;\big|\; \Phi_1 \wedge \Phi_2 \;\big|\; \neg\Phi \;\big|\; \exists\varphi \;\big|\; \forall\varphi$$

- and φ is a path-formula formed by the grammar:

$$\varphi ::= \bigcirc\Phi \;\big|\; \Phi_1 \,\mathsf{U}\, \Phi_2.$$

Examples

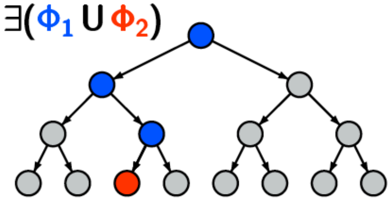
$\forall\Box\exists\bigcirc\, a$ and $\exists(\forall\Box a)\,\mathsf{U}\, b$ are CTL formulas.

Intuition

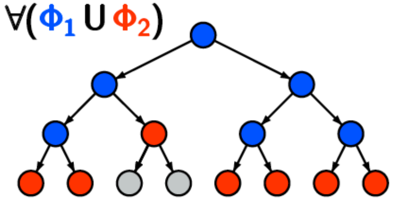
- $s \models \forall\varphi$ if all paths starting in s fulfill φ
- $s \models \exists\varphi$ if some path starting in s fulfill φ

Intuitive CTL Semantics

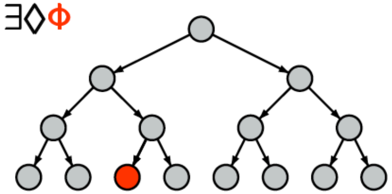
$\exists(\Phi_1 \, \mathsf{U} \, \Phi_2)$



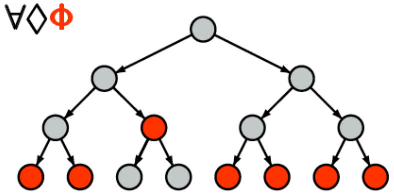
$\forall(\Phi_1 \, \mathsf{U} \, \Phi_2)$



$\exists\Diamond\Phi$

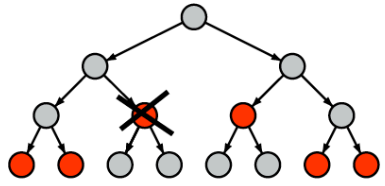


$\forall\Diamond\Phi$

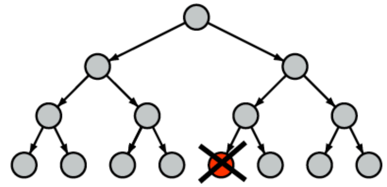


Intuitive CTL Semantics

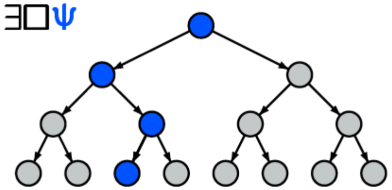
$\neg\forall\Diamond\Phi$



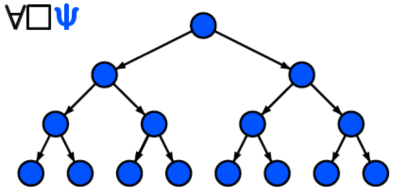
$\neg\exists\Diamond\Phi$



$\exists\Box\Psi$



$\forall\Box\Psi$



Overview

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 Φ 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:

$$
\begin{array}{lll}
s \models a & \text{iff} & a \in L(s) \\
s \models \neg \Phi & \text{iff} & \text{not } (s \models \Phi) \\
s \models \Phi \wedge \Psi & \text{iff} & (s \models \Phi) \text{ and } (s \models \Psi) \\
s \models \exists \varphi & \text{iff} & \text{there exists } \pi \in Paths(s).\ \pi \models \varphi \\
s \models \forall \varphi & \text{iff} & \text{for all } \pi \in Paths(s).\ \pi \models \varphi
\end{array}
$$

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

CTL Semantics (2)

Definition: satisfaction relation for CTL path-formulas

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

$$
\begin{array}{ll}
\pi \models \bigcirc \Phi & \text{iff } \pi[1] \models \Phi \\
\pi \models \Phi \,\mathsf{U}\, \Psi & \text{iff } (\exists j \geq 0.\ \pi[j] \models \Psi \text{ and } (\forall\, 0 \leq i < j.\ \pi[i] \models \Phi))
\end{array}
$$

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

Transition System Semantics

- For CTL-state-formula Φ, the satisfaction set $Sat(\Phi)$ is defined by:

$$Sat(\Phi) \;=\; \{\, s \in S \mid s \models \Phi \,\}$$

- TS satisfies CTL-formula Φ iff Φ holds in all its initial states:

$$TS \models \Phi \quad \text{if and only if} \quad \forall s_0 \in I.\ s_0 \models \Phi$$

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

Overview

Existential Normal Form

Definition: existential normal form

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

$$\Phi \; ::= \; \text{true} \;\mid\; a \;\mid\; \Phi_1 \wedge \Phi_2 \;\mid\; \neg\Phi \;\mid\; \exists\bigcirc\Phi \;\mid\; \exists(\Phi_1 \, \mathsf{U} \, \Phi_2) \;\mid\; \exists\Box\Phi$$

Only existentially quantified temporal modalities $\bigcirc$, U and $\Box$.

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

Proof.

Universally quantified temporal modalities can be transformed as follows:

$$\forall\bigcirc\Phi \equiv \neg\exists\bigcirc\neg\Phi$$

$$\forall(\Phi \, \mathsf{U} \, \Psi) \equiv \neg\exists(\neg\Psi \, \mathsf{U} \, (\neg\Phi \wedge \neg\Psi)) \wedge \neg\exists\Box\neg\Psi$$

Overview

Basic Idea

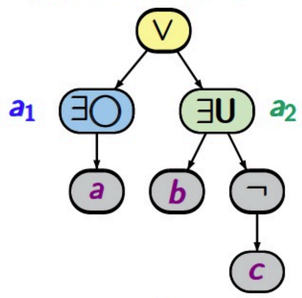
- How to check whether TS satisfies CTL formula Ψ?
  - convert the formula Ψ into the equivalent Φ in ENF
  - compute recursively the set $Sat(\Phi) = \{ s \in S \mid s \models \Phi \}$
  - $TS \models \Phi$ if and only if each initial state of TS belongs to $Sat(\Phi)$
- Recursive bottom-up computation of $Sat(\Phi)$:
  - consider the parse tree of Φ
  - start to compute $Sat(a_i)$, for all leafs in the parse tree
  - then go one level up in the tree and determine $Sat(\cdot)$ for these nodes

    e.g.,: $Sat(\underbrace{\Psi_1 \wedge \Psi_2}_{\text{node at level } i}) = Sat(\underbrace{\Psi_1}_{\text{node at level } i+1}) \cap Sat(\underbrace{\Psi_2}_{\text{node at level } i+1})$
  - then go one level up and determine $Sat(\cdot)$ of these nodes
  - and so on....... until the root is treated, i.e., $Sat(\Phi)$ is computed
- Check whether $I \subseteq Sat(\Phi)$.

Basic Algorithm

$$\Phi = \underbrace{\exists\bigcirc a}_{\Phi_1} \vee \underbrace{\exists(b \, \mathsf{U} \, \neg c)}_{\Phi_2} \;\rightsquigarrow\; a_1 \vee a_2$$

syntax tree for Φ



processed in bottom-up fashion

compute $Sat(a), Sat(b), Sat(c)$

$Sat(\Phi_1) = \ldots = Sat(a_1)$

$Sat(\neg c) = S \setminus Sat(c)$

$Sat(\Phi_2) = \ldots = Sat(a_2)$

replace Φ_1 with a_1
replace Φ_2 with a_2

$Sat(\Phi) = Sat(a_1) \cup Sat(a_2)$

Basic Algorithm

$$\begin{array}{rcl} Sat(\text{true}) & = & S \\ Sat(a) & = & \{ s \in S \mid a \in L(s) \} \\ Sat(\neg\Phi) & = & S \setminus Sat(\Phi) \\ Sat(\Phi \wedge \Psi) & = & Sat(\Phi) \cap Sat(\Psi) \\ Sat(\exists\bigcirc \Phi) & = & \{ s \in S \mid Post(s) \cap Sat(\Phi) \neq \varnothing \} \\ Sat(\exists\Box\Phi) & = & \ldots\ldots \\ Sat(\exists(\Phi \,\mathsf{U}\, \Psi)) & = & \ldots\ldots \end{array}$$

Treatment of $\exists\Box\Phi$ and $\exists(\Phi \,\mathsf{U}\, \Psi)$: via a fixed-point computation

Overview

Characterization of *Sat* for EU

Expansion law:

$$\exists(\Phi \,\mathsf{U}\, \Psi) \equiv \Psi \vee \left(\Phi \wedge \exists\bigcirc\exists(\Phi \,\mathsf{U}\, \Psi)\right)$$

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

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

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

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

$$\Omega(T) = Sat(\Psi) \cap \{ s \in Sat(\Phi) \mid Post(s) \cap T \neq \varnothing \}$$

Proof

Characterization of Sat for $\exists\Box$

Expansion law:

$$\exists\Box\Phi \equiv \Phi \wedge \exists\bigcirc\exists\Box\Phi$$

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

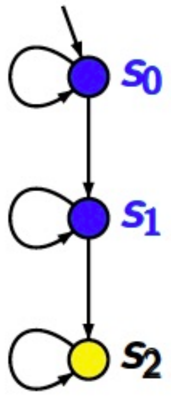
$Sat(\exists\Box\Phi)$ is the largest subset V of S, such that:

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

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

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

Example



$V = \{ s_0 \}$ satisfies the condition

$$V \subseteq \{ s \in Sat(\Phi) \mid Post(s) \cap V \neq \varnothing \}$$

but $V \subset Sat(\exists\Box a) = \{ s_0, s_1 \}$

Universally Quantified Formulas

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

  $$T \subseteq \{ s \in Sat(\Phi) \mid Post(s) \subseteq T \}$$

- $Sat(\forall(\Phi \mathsf{U} \Psi))$ is the smallest set T of states such that:

  $$Sat(\Psi) \cup \{ s \in Sat(\Phi) \mid Post(s) \subseteq T \} \subseteq T$$

Model Checking EU

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

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

- This suggests to compute $Sat(\exists(\Phi \mathsf{U} \Psi))$ iteratively:

  $$T_0 = Sat(\Psi) \quad \text{and} \quad T_{i+1} = T_i \cup \{ s \in Sat(\Phi) \mid Post(s) \cap T_i \neq \varnothing \}$$

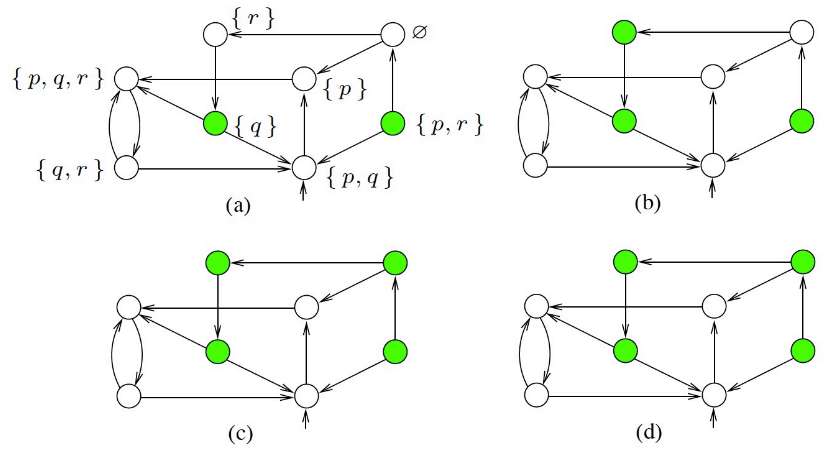
- T_i = states that can reach a Ψ-state in at most i steps via Φ states
- By induction it follows:

  $$T_0 \subseteq T_1 \subseteq \ldots \subseteq T_j \subseteq T_{j+1} \subseteq \ldots \subseteq Sat(\exists(\Phi \mathsf{U} \Psi))$$

- As TS is finite, we have $T_{k+1} = T_k = Sat(\exists(\Phi \mathsf{U} \Psi))$ for some k.

Model Checking EU in Pictures

Example



Computing $\exists\Diamond\left((p{=}r) \wedge (p \neq q)\right)$

Algorithm

$T := Sat(\Phi_2)$ ⟵ $\boxed{\text{collects all states } s \models \exists(\Phi_1 \,\mathsf{U}\, \Phi_2)}$

$E := Sat(\Phi_2)$ ⟵ $\boxed{\text{set of states still to be expanded}}$

WHILE $E \neq \varnothing$ DO
  select a state $s' \in E$ and remove s' from E
  FOR ALL $s \in Pre(s')$ DO
    IF $s \in Sat(\Phi_1) \setminus T$ THEN add s to T and E FI
  OD
OD

return T

Compute $Sat(\exists\Phi \,\mathsf{U}\, \Psi)$ by a linear-time enumerative backward search

Model Checking $\exists\Box$

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

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

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

Algorithm

$T := Sat(\Phi) \longleftarrow$ organizes the candidates for $s \models \exists\Box\Phi$

$E := S \setminus T \longleftarrow$ set of states to be expanded

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

naïve implementation:
quadratic time complexity

Compute $Sat(\exists\Box\Phi)$ by an enumerative backward search

Linear-Time Algorithm

$T := Sat(\Phi) \longleftarrow$ organizes the candidates for $s \models \exists\Box\Phi$

$E := S \setminus T \longleftarrow$ set of states to be expanded

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

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

Compute $Sat(\exists\Box\Phi)$ by a linear-time enumerative backward search

Linear-Time Algorithm Using Counters

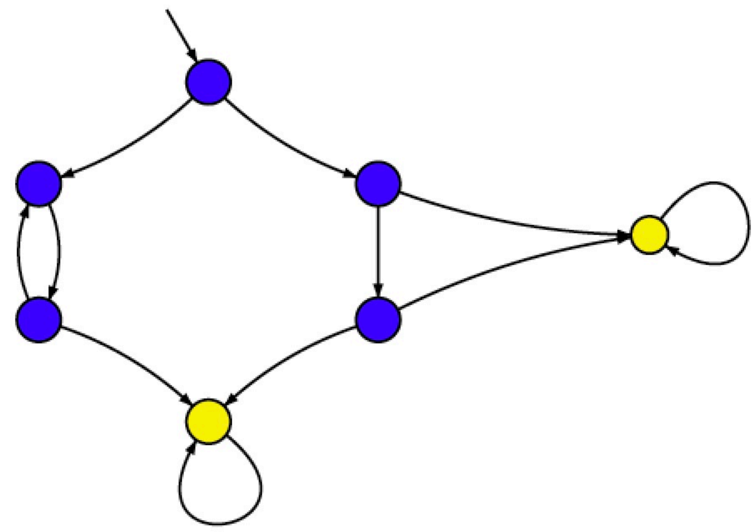
$T := Sat(\Phi)$; $E := S \setminus T$
FOR ALL $s \in Sat(\Phi)$ DO $c[s] := |Post(s)|$ OD

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

WHILE $E \neq \varnothing$ DO
  pick a state $s' \in E$ and remove s' from E
  FOR ALL $s \in Pre(s')$ 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
   FI
OD

Compute $Sat(\exists\Box\Phi)$ by a linear-time enumerative backward search

Example



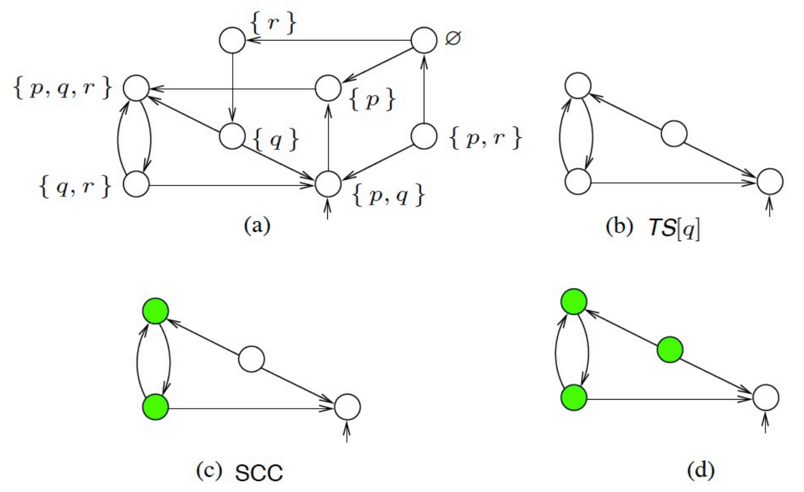
Computing $Sat(\exists\Box blue)$ search

An Alternative SCC-Based Algorithm

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

1. Eliminate all states $s \notin Sat(\Phi)$:
   - determine $TS[\Phi] = (S', Act, \rightarrow', I', AP, L')$ with $S' = Sat(\Phi)$, $\rightarrow' = \rightarrow \cap (S' \times Act \times S')$, $I' = I \cap S'$, 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
   - $\Rightarrow$ 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

Example



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

CTL Model-Checking Algorithm

$$
\begin{array}{rcl}
Sat(\text{true}) & = & S \\
Sat(a) & = & \{ s \in S \mid a \in L(s) \} \\
Sat(\neg\Phi) & = & S \setminus Sat(\Phi) \\
Sat(\Phi \wedge \Psi) & = & Sat(\Phi) \cap Sat(\Psi) \\
Sat(\exists\bigcirc \Phi) & = & \{ s \in S \mid Post(s) \cap Sat(\Phi) \neq \varnothing \} \\
Sat(\exists\Box\Phi) & = & \bigcap_{n \geq 0} V_n \text{ where} \\
 & & V_0 = Sat(\Phi) \\
 & & V_{n+1} = \{ s \in T_i \mid Post(s) \cap V_n \neq \varnothing \} \\
Sat(\exists(\Phi \,\mathsf{U}\, \Psi)) & = & \bigcup_{n \geq 0} T_n \text{ where} \\
 & & T_0 = Sat(\Psi) \\
 & & T_{n+1} = T_n \cup \{ s \in Sat(\Phi) \mid Post(s) \cap T_n \neq \varnothing \}
\end{array}
$$

Overview

Time Complexity

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

Proof.

1. The parse tree of Φ 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 $Sat(\exists \, \mathsf{U})$ and $Sat(\exists \square \ldots)$
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 Φ is equivalent to some LTL-formula φ then:

$\Phi \equiv \varphi$ where φ is obtained by removing all path quantifiers from Φ.
In particular, $|\varphi| \leq |\Phi|$.

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

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

Proof.

Take φ_n = the absence of a Hamiltonian path in a digraph on n vertices □

LTL Encoding the Hamiltonian Path Problem

CTL Encoding the Hamiltonian Path Problem

All $n!$ possibilities need to be explicitly enumerated

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

Overview

Summary

- CTL model checking determines $Sat(\Phi)$ by a recursive descent over Φ
- $Sat(\exists(\Phi \, \mathsf{U} \, \Psi))$ is approximated from below by a backward search from Ψ-states
- $\exists\Box\Phi$ is approximated from above by a backward search from Φ-states
- The CTL model-checking algorithm is linear in the size of TS and Φ
- The CTL model-checking problem is PTIME-complete