Model Checking

CS560: Reasoning About Programs

Roopsha Samanta

Based on slides by Georg Weissenbacher
Roadmap

Previously
- Bounded model checking for programs

Today
- Model checking for Kripke structures
Clarke & Emerson, *Design and Synthesis of Synchronization Skeletons using Branching-Time Temporal Logic*, 1981

Algorithmic framework for exhaustive exploration of finite-state transition systems to check temporal properties
Knaster-Tarski Theorem: Definitions

Given a state space $S$, a function/predicate transformer $f : 2^S \rightarrow 2^S$ is monotone if $\forall X, Y \in S$: $X \subseteq Y \Rightarrow f(X) \subseteq f(Y)$

$X$ is a **fixpoint** of function $f$ if $f(X) = X$.

$X$, is a **least fixpoint** of function $f$ if for any fixpoint $Y$, $X \subseteq Y$.

$X$ is a **greatest fixpoint** of function $f$ if for any fixpoint $Y$, $X \supseteq Y$.

$\mu(Y). f(Y)$

$\nu(Y). f(Y)$
Knaster-Tarski Theorem

Let $S$ be a set of states and $f: 2^S \mapsto 2^S$ be a monotone predicate transformer. Then:

1. $\mu(Y). f(Y) = \cap \{Y: f(Y) = Y\} = \cup_i f^i(\text{false})$
2. $\nu(Y). f(Y) = \cup \{Y: f(Y) = Y\} = \cap_i f^i(\text{true})$

- Monotone functions always have a least and a greatest fix point!
- These fixpoints can be easily computed
- The meanings of CTL operators can be expressed as fixpoints of monotone functions on $2^S$, enabling efficient model checking
Knaster-Tarski Theorem

Let $S$ be a set of states and $f: 2^S \mapsto 2^S$ be a monotone predicate transformer. Then:

1. $\mu(Y).f(Y) = \cap \{Y: f(Y) = Y\} = \cap \{Y: f(Y) \subseteq Y\} = \bigcup_i f^i(false)$
2. $\nu(Y)f(Y) = \cup \{Y: f(Y) = Y\} = \cup \{Y: f(Y) \supseteq Y\} = \bigcap_i f^i(true)$

- Monotone functions always have a least and a greatest fix point!
- These fixpoints can be easily computed
- The meanings of CTL operators can be expressed as fixpoints of monotone functions on $2^S$, enabling efficient model checking
Let $S$ be a set of states and $f : 2^S \mapsto 2^S$ be a monotone predicate transformer. Then:

1. $\mu(Y). f(Y) = \cap \{Y : f(Y) = Y\} = \cap \{Y : f(Y) \subseteq Y\} = \cup_i f^i(false)$
2. $\nu(Y). f(Y) = \cup \{Y : f(Y) = Y\} = \cup \{Y : f(Y) \supseteq Y\} = \cap_i f^i(true)$

Let $S = \{s_0, s_1, \ldots, s_n\}$ be a set of states and $f : 2^S \mapsto 2^S$ be a monotone predicate transformer. Then:

1. the least fixpoint exists and equals $f^{n+1}(\emptyset)$ and
2. the greatest fixpoint exists and equals $f^{n+1}(S)$. 

Simpler version when $S$ is finite
Knaster-Tarski Theorem: Definitions

A complete lattice is a partially ordered set \((L, \leq)\) where every subset of \(L\) has a glb and an lub. A function \(f\) over a lattice \((L, \leq)\) is monotonic if for all \(x, y \in L\): \(x \leq y \Rightarrow f(x) \leq f(y)\). Point \(x\) is a fixpoint of function \(f\) if \(f(x) = x\), a prefixed point if \(f(x) \leq x\) and a postfixed point if \(f(x) \geq x\).

Given a state space \(S\), the power set \(2^S\) is a complete lattice where \(\leq\) is the subset relation. Consider a monotonic predicate transformer \(f: 2^S \mapsto 2^S\) over this lattice.

Let \((L, \leq)\) be a complete lattice and \(f: L \mapsto L\) be a monotone function. Then:
1. the least fixpoint exists and equals the least prefixed point,
2. the greatest fixpoint exists and equals the greatest postfixed point, and
3 the fixpoints form a complete lattice.
Kanster-Tarski Theorem

Knaster, *Un théorème sur les fonctions d’ensembles*, 1927

Model Checking CTL

- For each CTL formula \( \varphi \), we will compute
  \[
  \{ s \mid \mathcal{M}, s \models \varphi \}
  \]

- CTL can be expressed in terms of \( \neg, \lor, EX, EU \), and \( EG \)

- Will define these operators by induction:
  - \( EX \varphi \) is defined as \( \{ s_0 \mid \exists s_1. \ T(s_0, s_1) \land \mathcal{M}, s_1 \models \varphi \} \)
For each CTL formula $\varphi$, we will compute
\[ \{ s \mid M, s \models \varphi \} \]

CTL can be expressed in terms of $\neg$, $\lor$, $EX$, $EU$, and $EG$

Will define these operators by induction:
- $EX\varphi \overset{\text{def}}{=} \{ s_0 \mid \exists s_1. T(s_0, s_1) \land M, s_1 \models \varphi \}$
  - Note: This is the pre-image of $T$ with respect to $\varphi$
Model Checking CTL

- It remains to be shown that \( EG\varphi \) and \( EU\varphi \) can be computed
- We claim:
  - \( EG\varphi \equiv \nu Z.\varphi \land EX Z \)
    - i.e., \( EG\varphi \) is greatest fixed point of \( \tau(Z) = \varphi \land EX Z \)
  - \( E(\varphi_1 U \varphi_2); \equiv \mu Z.\varphi_2 \lor (\varphi_1 \land EX Z) \)
    - i.e., \( E(\varphi_1 U \varphi_2) \) is least fixed point of \( \tau(Z) = \varphi_2 \lor (\varphi_1 \land EX Z) \)
    - Recall least fixed point of strongest post condition
Model Checking CTL

- \( E(\varphi_1 U \varphi_2) \equiv \mu Z. \varphi_2 \lor (\varphi_1 \land EX Z) \)
  - Remember: \( EX \) is “pre-image”

- \( E(\varphi_1 U \varphi_2) \) holds in \( \varphi_2 \)
Model Checking CTL

- \( E(\varphi_1 U \varphi_2); \equiv \mu Z. \varphi_2 \lor (\varphi_1 \land EX Z) \)
  - Remember: \( EX \) is “pre-image”

- \( E(\varphi_1 U \varphi_2) \) holds in \( \varphi_2 \)

- And in predecessor states of \( \varphi_2 \) in which \( \varphi_1 \) holds
Model Checking CTL

- $E(\varphi_1 U \varphi_2); \equiv \mu Z. \varphi_2 \lor (\varphi_1 \land EX Z)$
  - Remember: $EX$ is “pre-image”

- $E(\varphi_1 U \varphi_2)$ holds in $\varphi_2$
- And in predecessor states of $\varphi_2$ in which $\varphi_1$ holds
- Fixed point: Transitive closure of all such predecessor states
Model Checking CTL

- $EG\varphi \equiv \nu Z. \varphi \land EX Z$
  - Remember: $EX$ is “pre-image”

- Start with all states in which $\varphi$ holds
Model Checking CTL

- $EG\varphi \equiv \nu Z . \varphi \land EX Z$
  - Remember: $EX$ is “pre-image”

- Start with all states in which $\varphi$ holds
- Shrink to states in $\varphi$ such that $\varphi$ still holds after 1 step
Model Checking CTL

- $EG\varphi \equiv \nu Z. \varphi \land EX Z$
  - Remember: $EX$ is “pre-image”

- Start with all states in which $\varphi$ holds
- Shrink to states in $\varphi$ such that $\varphi$ still holds after 1 step
- Keep shrinking until fixed point reached
Model Checking CTL

\[ E(\mathcal{T} \ U \ \mathcal{T}) \]

\[ \mu Z. (V (\perp \land EX \perp) = \{ s_2 \}) \]

\[ 1. V (\perp \land EX \perp) = \{ s_2 \} \]
Model Checking CTL

\[ E(\mu Z. \forall x (\exists y (y = x \land EX x) \land EX y)) \]

1. \[ \forall x (\exists y (y = x \land EX \bot)) = \{s_2\} \]

2. \[ \forall x (\exists y (y = x \land EX \{s_2\})) = \]
Model Checking CTL

\[ E(\mathbb{U}) \]

1. \( V (\top \land \neg Z) = \{s_2\} \)

2. \( V (\top \land \{s_1\}) = \)

\[ \mu Z. \forall V (\top \land EX Z) \]
Model Checking CTL

\[ E(\overline{G} U \overline{F}) \]

\[ \begin{align*}
\mu Z. &\quad V (\overline{G} \land EX Z) \\
1. &\quad V (\overline{G} \land EX \bot) = \{s_2\} \\
2. &\quad V (\overline{G} \land \{s_1\}) = \{s_1, s_2\}
\end{align*} \]

\[ E(\text{guy}) \]

\[ E'(\text{guy}) \]
Model Checking CTL

\[
E(\text{ } U \text{ } )
\]

1. \( V (\text{ } \land \text{ } EX \bot ) = \{s_2\} \)
2. \( V (\text{ } \land \{s_1\} ) = \{s_1, s_2\} \)
3. \( V (\text{ } \land EX \{s_1, s_2\} ) = \)

\[
\mu Z. \ V (\text{ } \land EX Z )
\]
Model Checking CTL

\[
E(\begin{array}{c} \text{\textcolor{red}{R}} \\ \text{\textcolor{red}{G}} \\ \text{\textcolor{red}{B}} \end{array} U \begin{array}{c} \text{\textcolor{green}{G}} \\ \text{\textcolor{green}{Y}} \end{array})
\]

\[
\mu Z. V (\begin{array}{c} \text{\textcolor{green}{G}} \\ \text{\textcolor{green}{Y}} \end{array} \land EX \bot)
\]

1. \[
V (\begin{array}{c} \text{\textcolor{green}{G}} \\ \text{\textcolor{green}{Y}} \end{array} \land EX \bot) = \{ s_2 \}
\]

2. \[
V (\begin{array}{c} \text{\textcolor{green}{G}} \\ \text{\textcolor{green}{Y}} \end{array} \land \{ s_1 \}) = \{ s_1, s_2 \}
\]

3. \[
V (\begin{array}{c} \text{\textcolor{green}{G}} \\ \text{\textcolor{green}{Y}} \end{array} \land \{ s_0, s_1, s_3 \}) = \]
Model Checking CTL

\[ E( \neg U \neg ) \]

1. \[ V ( \neg \neg \neg \neg ) = \{ s_2 \} \]
2. \[ V ( \neg \neg \neg \{ s_1 \} ) = \{ s_1, s_2 \} \]
3. \[ V ( \{ s_1 \} ) = \]

\[ \mu Z . V ( \neg \neg \neg \neg \neg \neg ) \]
Model Checking CTL

\[ E(\quad U \quad ) \]

\[ \mu Z. V (\quad \land \; EX \; \bot) \]

1. \[ V (\quad \land \; EX \; \bot) = \{ s_2 \} \]

2. \[ V (\quad \land \; \{ s_1 \} ) = \{ s_1, \; s_2 \} \]

3. \[ V (\; \{ s_1 \} ) = \{ s_1, \; s_2 \} \]
Model Checking CTL

$$E( \begin{array}{c} \text{绿} \text{黄} \text{红} \\ \text{绿} \text{黄} \text{红} \end{array} )$$

$$\begin{array}{c} s_0 \\ s_1 \\ s_2 \\ s_3 \end{array}$$

$$\mu Z. \bigvee V ( \bigwedge EX \bot )$$

1. $$V ( \bigwedge EX \bot ) = \{ s_2 \}$$
2. $$V ( \bigwedge \{ s_1 \} ) = \{ s_1, s_2 \}$$
3. $$V ( \{ s_1 \} ) = \{ s_1, s_2 \}$$

4. Fixed Point!
   - $$\mathcal{M}, s_1 \models E( \begin{array}{c} \text{绿} \text{黄} \text{红} \\ \text{绿} \text{黄} \text{红} \end{array} )$$
   - $$\mathcal{M}, s_2 \models E( \begin{array}{c} \text{绿} \text{黄} \text{红} \\ \text{绿} \text{黄} \text{红} \end{array} )$$
Model Checking CTL

- More complex formulas?
  - Start with innermost sub-formulas!
  - Compute nested fixed point

- Remember:
  \[ E(\text{ } U \text{ } ) = \{ s_1, s_2 \} \]

- So if we want to compute
  \[ EG(E(\text{ } U \text{ } )) \]

- We compute greatest fixed point
  \[ \nu Z.\{ s_1, s_2 \} \land EX Z \]
Let’s compute the greatest fixed point

\[ \nu Z.\{ s_1, s_2 \} \land EX Z \]

1. \[ \{ s_1, s_2 \} \land EX T \]
Let’s compute the greatest fixed point

\[ \nu Z. \{ s_1, s_2 \} \land EX Z \]

1. \{ s_1, s_2 \} \land T
Let’s compute the greatest fixed point

$$\nu Z . \{ s_1, s_2 \} \land EX Z$$

$$1. \{ s_1, s_2 \} \land T = \{ s_1, s_2 \}$$
Model Checking CTL

- Let’s compute the greatest fixed point

$$\nu Z.\{s_1, s_2\} \land EX Z$$

1. $$\{s_1, s_2\} \land T = \{s_1, s_2\}$$

2. $$\{s_1, s_2\} \land EX \{s_1, s_2\}$$
Let’s compute the greatest fixed point

\[ \nu Z. \{s_1, s_2\} \land EX Z \]

1. \( \{s_1, s_2\} \land T = \{s_1, s_2\} \)

2. \( \{s_1, s_2\} \land \{s_0, s_1, s_3\} \)
Model Checking CTL

Let’s compute the greatest fixed point

\[ \nu Z . \{ s_1, s_2 \} \land EX Z \]

1. \[ \{ s_1, s_2 \} \land T = \{ s_1, s_2 \} \]

2. \[ \{ s_1, s_2 \} \land \{ s_0, s_1, s_3 \} = \{ s_1 \} \]
Model Checking CTL

Let’s compute the greatest fixed point

\[ \nu Z . \{ s_1, s_2 \} \land \mathbf{EX} Z \]

1. \( \{ s_1, s_2 \} \land T = \{ s_1, s_2 \} \)
2. \( \{ s_1, s_2 \} \land \{ s_0, s_1, s_3 \} = \{ s_1 \} \)
3. \( \{ s_1, s_2 \} \land \mathbf{EX} \{ s_1 \} \)
Model Checking CTL

- Let’s compute the greatest fixed point

\[ \nu Z . \{ s_1, s_2 \} \land EX Z \]

1. \( \{ s_1, s_2 \} \land T = \{ s_1, s_2 \} \)
2. \( \{ s_1, s_2 \} \land \{ s_0, s_1, s_3 \} = \{ s_1 \} \)
3. \( \{ s_1, s_2 \} \land \{ s_0, s_1, s_3 \} \)
Let’s compute the greatest fixed point

\[ vZ . \{ s_1, s_2 \} \land EX Z \]

1. \( \{ s_1, s_2 \} \land T = \{ s_1, s_2 \} \)

2. \( \{ s_1, s_2 \} \land \{ s_0, s_1, s_3 \} = \{ s_1 \} \)

3. \( \{ s_1, s_2 \} \land \{ s_0, s_1, s_3 \} = \{ s_1 \} \)
Let’s compute the greatest fixed point

\[ \nu Z . \{ s_1, s_2 \} \land \mathbf{EX} Z \]

1. \( \{ s_1, s_2 \} \land T = \{ s_1, s_2 \} \)

2. \( \{ s_1, s_2 \} \land \{ s_0, s_1, s_3 \} = \{ s_1 \} \)

3. \( \{ s_1, s_2 \} \land \{ s_0, s_1, s_3 \} = \{ s_1 \} \)

4. Fixed Point!

\[ \mathcal{M}, s_1 \models \mathbf{EG}(E( \text{ } U \text{ } )) \]
Model Checking CTL

- Worst case complexity of this algorithm?
- Checking CTL-Formula $\varphi$ for $\langle S, T, I, L \rangle$ is $O(|\varphi| \cdot (|S| + |T|))$
- Why?
  - Each fixed point is $O(|S| + |T|)$
  - We have to compute $O(|\varphi|)$ fixed points
LTL Model Checking

- We’ll look at the problem from a new angle:
  Model Checking using Automata Theory

- Remember: a finite automaton accepts a finite input if a final state is reached

(This automaton accepts (a|b)*b – all words ending with b)
Automata Theory

Definition (Finite Automaton)

A finite automaton $A$ is a tuple $<\Sigma, Q, \delta, Q_0, F>$

- $\Sigma$ is the input alphabet
- $Q$ is a finite set of states
- $\delta : Q \times \Sigma \times Q$ is the transition relation
- $Q_0 \subseteq Q$ is the set of initial states
- $F \subseteq Q$ is the set of final states

$\Sigma = \{a, b\}, \quad Q = \{p_0, p_1\}, \quad Q_0 = \{p_0\}, \quad F = \{p_1\}$
Let $\mathcal{A}$ be a finite automaton over an input alphabet $\Sigma$
Then $\mathcal{L}(\mathcal{A})$ denotes the language:
\[
\{ w \in \Sigma^* \mid \mathcal{A} \text{ accepts } w \}
\]
i.e., $\mathcal{L}(\mathcal{A})$ consists of all finite words accepted by $\mathcal{A}$

(This automaton accepts $(a|b)^*b$ – all words ending with $b$)
Maybe we can define an automaton accepting “good” execution traces?
Maybe we can define an automaton accepting “good” execution traces?

Problem: An execution trace is an infinite sequence of states
Maybe we can define an automaton accepting “good” execution traces?

Problem: An execution trace is an infinite sequence of states

Solution: Define automata accepting infinite input words
  These automata are called $\omega$-Automata or Büchi-automata
ω-Automata

Definition (Büchi Automaton)

A Büchi automaton $B$ is a tuple $<\Sigma, Q, \delta, Q_0, F>$

- $\Sigma$ is the input alphabet
- $Q$ is a finite set of states
- $\delta : Q \times \Sigma \times Q$ is the transition relation
- $Q_0 \subseteq Q$ is the set of initial states
- $F \subseteq Q$ is the set of accepting states
ω-Automata

Definition (Büchi Automaton)

A Büchi automaton $B$ is a tuple $<\Sigma, Q, \delta, Q_0, F>$

- $\Sigma$ is the input alphabet
- $Q$ is a finite set of states
- $\delta : Q \times \Sigma \times Q$ is the transition relation
- $Q_0 \subseteq Q$ is the set of initial states
- $F \subseteq Q$ is the set of accepting states

Wait... isn’t that exactly the same definition as before?
ω-Automata

Definition (Büchi Automaton)

A Büchi automaton $B$ is a tuple $\langle \Sigma, Q, \delta, Q_0, F \rangle$:
- $\Sigma$ is the input alphabet
- $Q$ is a finite set of states
- $\delta : Q \times \Sigma \times Q$ is the transition relation
- $Q_0 \subseteq Q$ is the set of initial states
- $F \subseteq Q$ is the set of accepting states
- A Büchi automaton accepts an infinite word $w \in \Sigma^\omega$ if the corresponding run visits at least one state in $F$ infinitely often.
We use $\Sigma^\omega$ to denote all words of infinite length.

The following automaton accepts all words $w \in \Sigma^\omega$ with finitely many $a$'s.

- **$\omega$-Automata**

\[ \Sigma = \{a, b\}, \quad Q = \{p_0, p_1\}, \quad Q_0 = \{p_0\}, \quad F = \{p_1\} \]
ω-Automata

- Intuitively, an automaton $B$ defines a set of infinite behaviors
\(\omega\)-Automata

- Intuitively, an automaton \(B\) defines a set of infinite behaviors
- We can use Büchi automata to represent Kripke structures!
  - Make all states accepting states
  - Label incoming edges of \(s_i\) with \(L(s_i)\)
Model Checking with $\omega$-Automata

- We can encode any Kripke structure $\mathcal{M}$ as a Büchi automaton $B\mathcal{M}$
  - For a given run $s_0, s_1 \ldots$ of $\mathcal{M}$, $B\mathcal{M}$ accepts $L(s_0) L(s_1) \ldots$
- Now assume we have a second automaton, $B\varphi$ representing a “specification” $\varphi$
- Let $B\varphi$ be the automaton accepting “all good behaviors” according to $\varphi$
- Then $\overline{B\varphi}$ is the automaton accepting “all bad behaviors”
- Then $\mathcal{M}$ can’t behave badly if
  $$L(B\mathcal{M}) \cap L(\overline{B\varphi}) = \emptyset$$

Emptiness checking of Büchi automata
Model Checking with $\omega$-Automata

- We can encode any Kripke structure $\mathcal{M}$ as a Büchi automaton $B_{\mathcal{M}}$
  - For a given run $s_0, s_1 \ldots$ of $\mathcal{M}$, $B_{\mathcal{M}}$ accepts $L(s_0) L(s_1) \ldots$
- Now assume we have a second automaton, $B_{\varphi}$ representing a “specification” $\varphi$
- Let $B_{\varphi}$ be the automaton accepting “all good behaviors” according to $\varphi$
- Then $\overline{B_{\varphi}}$ is the automaton accepting “all bad behaviors”
- Then $\mathcal{M}$ can’t behave badly if
  \[ L(B_{\mathcal{M}}) \cap L(\overline{B_{\varphi}}) = \emptyset \]
  \[ L(B_{\mathcal{M}}) \cap L(\overline{B_{\varphi}}) \]
  Is accepted by the “intersection of the automata”
  $B_{\mathcal{M}}$ and $\overline{B_{\varphi}}$
Automaton-based Model Checking of LTL

Given $\mathcal{M} = \langle S, T, I, L \rangle$ and $A\varphi$

1. Construct $B_\mathcal{M}$
2. Put $\neg \varphi$ into negation normal form
3. Construct $B_{\neg \varphi}$ for $\neg \varphi$ in NNF
   Negating Büchi automata is hard – we want to avoid this step
4. Construct $B = B_\mathcal{M} \cap B_{\neg \varphi}$
5. Check $B$ for emptiness ($\mathcal{L}(B) \models \emptyset$)
Summary

Today

- CTL model checking using nested fixed points
- LTL model checking using automata (overview)

Next

- Reading days
- Program synthesis!