Temporal Logic Model Checking

Thomas Wahl

Computing Laboratory, Oxford University
Temporal Logics

LTL Model Checking

CTL Model Checking
Temporal Logics: Purpose

Temporal Logic: decidable logic to reason over program behavior along infinite timelines.

Is statement $x$ executed infinitely often?

Is every request followed by a grant eventually?

Is there an execution along which always $x \geq 0$?
Temporal Logics: Ingredients

All temporal logics have **temporal operators**, which quantify over states along a program execution path:

“in all future states”
“in some future state”
“in the next state”

Some also have **branching operators**, which quantify over execution paths of a program:

“for all executions”
“for some execution”
Temporal Logics: Representatives

\textbf{LTL:} only temporal operators.
cannot distinguish different program executions

\textbf{CTL:} temporal + branching operators.
can explicitly specify behavior along different paths

\textbf{CTL*:} $\Downarrow LTL \cup CTL$

$\mu$-calculus: $\Downarrow$ CTL*

[\text{LTL: Pnueli ’77, CTL: Emerson/Clarke ’82}]
Temporal Logics: Which One to Use?

Infinite debate about which logic is “best”…

**Criteria:** easy of use, expressiveness, efficiency.

**Allen Emerson:** “Modalities for Model Checking: Branching Time Logic Strikes Back.” (1987)

**Moshe Vardi:** “Branching vs. Linear Time: Final Showdown.” (2001)
Outline

Temporal Logics

LTL Model Checking

CTL Model Checking
Linear Temporal Logic (LTL): Syntax

LTL formulas are built out of:

- atomic propositions: $p$, $q$, $r$
- Boolean operators: $\land$, $\neg$, $\ldots$
- temporal operators:
  - $X p$: “next time $p$”
  - $F q$: “eventually $q$”
  - $G r$: “always $r$”
  - $p U q$: “$p$ until $q$”

Again: these apply to a fixed execution path
Linear Temporal Logic: Semantics

LTL formulas evaluated over a path \( \pi = \pi_0\pi_1\pi_2 \ldots \)

Suffix of \( \pi \) starting at the \( i \)th state: \( \pi^i \).

Validity is defined as follows:

\[
\begin{align*}
\pi \models p & \quad \text{iff} \quad p \in L(\pi_0) \\
\pi \models \neg f & \quad \text{iff} \quad \pi \not\models f \\
\pi \models g \land h & \quad \text{iff} \quad \pi \models g \text{ and } \pi \models h \\
\pi \models X f & \quad \text{iff} \quad \pi^1 \models f \\
\pi \models G f & \quad \text{iff} \quad \forall i : \pi^i \models f \\
\pi \models F f & \quad \text{iff} \quad \exists i : \pi^i \models f \\
\pi \models g U h & \quad \text{iff} \quad \exists i : (\pi^i \models h \text{ and } \forall j < i : \pi^j \models g)
\end{align*}
\]
Given Kripke structure \( M := (S, R, L, s_0) \) and LTL formula \( f \):
do all paths \( \pi \) through \( M \) satisfy \( f \)?

\[
M \models f := \forall \pi : \pi_0 = s_0 \land \pi \text{ path in } M : \pi \models f
\]

**Approach:**
- represent \( M \) and \( f \) in *same* data structure
- should be efficiently manipulatable

**Solution:** finite-state automata
Finite-State Automata (FSA)

... quite similar to a transition system like a Kripke structure:

\[ A = (\Sigma, Q, \delta, Q_0, F) \]

- \( \Sigma, Q \) finite alphabet, finite state set
- Transition relation: \( \delta \subseteq Q \times \Sigma \times Q \)
- \( Q_0 \) initial states, \( F \) accepting states.

Automaton \( A \) accepts certain “words”, which form its language \( \mathcal{L}(A) \).
Kripke Structure and FSA

**Goal:** given structure $M$, define automaton $A_M$ such that

$$L(A_M) = \{ \pi : \pi \text{ is a path in } M \}.$$

Pretty straightforward.

**Technicality:**
- FSAs have edge labels (“inputs”),
- $M$ has state labels (atomic propositions).

But that can be fixed . . .
Kripke Structure and FSA

\[ M: \]
\[ M_A: \]

\[ s_0 \rightarrow \{ p, q \} \]
\[ s_0 \rightarrow \{ q \} \]
\[ s_1 \rightarrow \{ p \} \]
\[ s_2 \rightarrow \{ q \} \]

Every state of \( M_A \) is accepting.
LTL Formula and FSA

Goal: given LTL formula $f$, define automaton $A_f$ such that

$$\mathcal{L}(A_f) = \{\pi : \pi \models f\}.$$ 

Achieve this using Büchi acceptance condition:

**Infinite** path is accepted by a FSA if some accepting state is visited infinitely often.

Means: path goes through a **cycle** that contains an accepting state.
LTL Formula and FSA

(These FSAs represent which LTL formulas?)

“Tableau Construction” [GPVW 1985-95]
Finally: LTL Model Checking, by Language Containment:

\[ M \models f \quad \text{iff} \quad \mathcal{L}(A_M) \subseteq \mathcal{L}(A_f) \]

iff \[ \mathcal{L}(A_M) \cap \neg \mathcal{L}(A_f) = \emptyset \]

iff \[ \mathcal{L}(A_M) \cap \mathcal{L}(A_{\neg f}) = \emptyset \]

iff \[ \mathcal{L}(A_M \cap A_{\neg f}) = \emptyset. \]

Terminology in MCer SPIN: \( \neg f \) is a “never claim”

Our algorithm looks for violations of \( f \).
LTL Model Checking

We had:

\[ M \models f \iff \mathcal{L}(A_M \cap A_{\neg f}) = \emptyset. \]

Intersection of two FSAs:
by lock-step execution (standard constructions)

Emptiness of a FSA \( A \) with Büchi acceptance condition:

\[ \mathcal{L}(A) \neq \emptyset \iff \text{there is a reachable cycle through } A \text{ that contains an accepting state}. \]
Final remarks.

**Counter examples:** If $\mathcal{L}(A_M \cap A_{\neg f}) \neq \emptyset$, then $M \not\models f$.

The path to and through the accepting cycle is a counter example.

**Complexity:** $O(|M| \cdot 2^{|f|})$.

Exponential complexity in $|f|$ is not as big a problem in practice as it may seem:

Usually, $|f| \ll |M|$.
Outline

Temporal Logics

LTL Model Checking

CTL Model Checking
Computation Tree Logic (CTL): Syntax

Combines branching operators

A: “for all executions” (“futures”, “paths”)
E: “for some execution”

and temporal operators X, F, G, U, but only in specific ways:

\[
\begin{bmatrix}
  A \\
  E
\end{bmatrix}
\begin{bmatrix}
  X \\
  F \\
  G \\
  U
\end{bmatrix}

\]

and arbitrarily nested:

AG EF reset \hspace{1cm} E(req U ack) \hspace{1cm} EX AX false

Not: A E p (makes no sense), A FG p (allowed in CTL*, not CTL)
Temporal Logics

LTL Model Checking

CTL Model Checking

CTL: Intuitive Semantics

\[\mathit{AX}p\]

"p imminent"

\[\mathit{AF}p\]

"p inevitable"

\[\mathit{EF}p\]

"p reachable"

\[\mathit{AG}p\]

"p invariant"
CTL Model Checking

**Input:** \( M := (S, R, L, s_0) \) and CTL formula \( f \)

**Goal:** determine whether \( M, s_0 \models f \).

We will do this by

1. computing \( S_f = \{ s \in S : M, s \models f \} \), and then
2. checking whether \( s_0 \in S_f \), possibly on the fly.
The Tarski-Knaster Theorem

View CTL formula as set of states satisfying it. Then observe:

\[ EG \, p \, = \, p \land EX \, p \land EX \, EX \, p \land \ldots \]
\[ = \, p \land EX(p \land EX(p \land \ldots)) \]
\[ = \, p \land EX(EG \, p) . \]
The Tarski-Knaster Theorem

View CTL formula as set of states satisfying it. Then observe:

\[ \text{EG} \, p = p \land \text{EX} \, p \land \text{EX} \, \text{EX} \, p \land \ldots \]
\[ = p \land \text{EX}(p \land \text{EX}(p \land \ldots)) \]
\[ = p \land \text{EX}(\text{EG} \, p) \].

\( \text{EG} \, p \) is a fixpoint of the operator \( \tau(Z) = p \land \text{EX} \, Z \):

- in fact the greatest fixpoint of \( \tau \)
- computable by series of overapproximations:

\[
Z_0 = S \supseteq \quad Z_1 = p \land \text{EX} \, Z_0 \quad \supseteq \quad Z_2 = p \land \text{EX} \, Z_1 \quad \ldots \quad = \quad \nu Z. \, p \land \text{EX} \, Z
\]
Computing Fixpoints

Least fixpoint
\[ \mu Z. \tau(Z) :\]
1: \( Z := \emptyset \)
2: repeat
3: \( \tilde{Z} := Z \)
4: \( Z := \tau(Z) \)
5: until \( Z = \tilde{Z} \)
6: return \( Z \)

Greatest fixpoint
\[ \nu Z. \tau(Z) :\]
1: \( Z := S \)
2: repeat
3: \( \tilde{Z} := Z \)
4: \( Z := \tau(Z) \)
5: until \( Z = \tilde{Z} \)
6: return \( Z \)

Only difference: initial value of \( Z \)

18 Feb, 2009 Thomas Wahl, Oxford University
Fixpoint Characterization of CTL

\[
\begin{align*}
\text{EF } h & = \mu Z.(h \lor \text{EX } Z) \\
\text{EG } h & = \nu Z.(h \land \text{EX } Z) \\
\text{E}(g \text{ U } h) & = \mu Z.(h \lor (g \land \text{EX } Z)) \\
\text{AF } h & = \mu Z.(h \lor \text{AX } Z) \\
\text{AG } h & = \nu Z.(h \land \text{AX } Z) \\
\text{A}(g \text{ U } h) & = \mu Z.(h \lor (g \land \text{AX } Z))
\end{align*}
\]

- F, U: least fixpoint \( \mu \), uses “\( \lor \)”. Liveness properties
- G: greatest fixpoint \( \nu \), uses “\( \land \)”. Safety properties
CTL Model Checking: Complexity

**Example** $AG\ EF\ p$: “It is always possible to reach a $p$-state.”

$$AG\ EF\ p = AG(EF\ p) = \nu Z. (\mu Y.p \lor EX\ Y) \land AX\ Z.$$

**Procedure:**

1. $Y_0 := \mu Y.p \lor EX\ Y\ (O(|M|))$
2. $Z_0 := \nu Z.Y_0 \land AX\ Z\ (O(|M|))$
3. return $Z_0$

$\Rightarrow cost \approx 2 \times |M|.$
CTL Model Checking: Complexity

Each fixpoint costs $O(|M|)$ steps
(and each step only involves EX, AX, $\land$, $\lor$)

number of fixpoint computations =
number of EF, EG, AF, AG’s that appear in $f$

$\Rightarrow$ complexity $O(|M| \times |f|)$.

Compare: LTL model checking: $O(|M| \times 2^{|f|})$. 
All we need to do is implement the fixpoint routines:

Set of states satisfying $\text{EF} \ p$:

1. $Z := \emptyset$
2. repeat
3. $\tilde{Z} := Z$
4. $Z := p \lor \text{EX} \ Z$
5. until $Z = \tilde{Z}$
6. return $Z$

Need:
- disjunction $\lor$
- (pre-)image EX
- termination $Z = \tilde{Z}$

Termination easy with BDDs: canonicity!
Symbolic Transition Relations

set of states: constraint over the state variables
set of transitions: constraint over two copies of the state variables:

\[ \text{if } x \text{ then } y := true \]

as a Boolean formula over 4 (not 2) variables:

\[ R(x, x', y, y') = ((x \land y') \lor (\neg x \land y' = y)) \land x' = x \]
Symbolic Image Operations

Pre-image (predecessors) of a set of states $Z$:

$$\text{EX } Z = \{ s : \exists z \in Z : R(s, z) \} .$$

Given:

- BDD $Z$ over $x_1, \ldots, x_k$, and
- BDD $R$ over $x_1, \ldots, x_k, x'_1, \ldots, x'_k$.

Wanted: BDD for $\text{EX } Z$ over $x_1, \ldots, x_k$.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result over variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $Z' := Z$ with $x_i$ renamed to $x'_i$</td>
<td>$x'_1 \ldots x'_k$</td>
</tr>
<tr>
<td>2. $L := Z' \land R$</td>
<td>$x_1 \ldots x_k, x'_1 \ldots x'_k$</td>
</tr>
<tr>
<td>3. result := $\exists x'_1 \ldots x'_k : L$</td>
<td>$x_1 \ldots x_k$</td>
</tr>
</tbody>
</table>