Model Checking II
Temporal Logic Model Checking

Edmund M. Clarke, Jr.
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
Temporal Logic Model Checking

**Specification Language:** A propositional temporal logic.

**Verification Procedure:** Exhaustive search of the state space of the system to determine if the specification is true or not.


Why Model Checking?

Advantages:
- No proofs!
- Fast
- Counter-examples
- No problem with partial specifications
- Logics can easily express many concurrency properties

Main Disadvantage: *State Explosion Problem*
- Too many processes
- Data Paths

Much progress has been made on this problem recently!
Finite-state systems are modeled by labeled state-transition graphs, called *Kripke Structures*.
If some state is designated as the *initial state*, the structure can be unwound into an infinite tree with that state as the root.

We will refer to this infinite tree as the *computation tree* of the system.

Paths in the tree represent possible computations or behaviors of the program.
Formally, a *Kripke structure* is a triple $M = \langle S, R, L \rangle$, where
- $S$ is the set of states,
- $R \subseteq S \times S$ is the transition relation, and
- $L : S \rightarrow \mathcal{P}(AP)$ gives the set of atomic propositions true in each state.

We assume that every state has at least one possible successor (i.e., for all states $s \in S$ there exists a state $s' \in S$ such that $(s, s') \in R$).

A *path in $M$* is an infinite sequence of states, $\pi = s_0, s_1, \ldots$ such that for $i \geq 0$, $(s_i, s_{i+1}) \in R$.

We write $\pi^i$ to denote the *suffix* of $\pi$ starting at $s_i$.

Unless otherwise stated, we assume *finite* Kripke structures.
Temporal logics may differ according to how they handle branching in the underlying computation tree.

In a linear temporal logic, operators are provided for describing events along a single computation path.

In a branching-time logic the temporal operators quantify over the paths that are possible from a given state.
The computation tree logic CTL* (pronounced “CTL star”) combines both branching-time and linear-time operators.

In this logic a path quantifier can prefix an assertion composed of arbitrary combinations of the usual linear-time operators.

1. Path quantifiers:
   - $\mathbf{A}$ — “for every path”
   - $\mathbf{E}$ — “there exists a path”

2. Linear-time operators:
   - $\mathbf{X} p$ — $p$ holds true next time.
   - $\mathbf{F} p$ — $p$ holds true sometime in the future
   - $\mathbf{G} p$ — $p$ holds true globally in the future
   - $p \mathbf{U} q$ — $p$ holds true until $q$ holds true

For a path $\pi = (s_0, s_1, \ldots)$, state $s_0$ is considered to be at the present time.
The syntax of state formulas is given by the following rules:

- If \( p \) is an atomic proposition, then \( p \) is a state formula.
- If \( f \) and \( g \) are state formulas, then \( \neg f \) and \( f \lor g \) are state formulas.
- If \( f \) is a path formula, then \( E(f) \) and \( A(f) \) are state formulas.

Two additional rules are needed to specify the syntax of path formulas:

- If \( f \) is a state formula, then \( f \) is also a path formula. (A state formula \( f \) is true for a path \( \pi \) if the \( f \) is true in the initial state of the path \( \pi \).)
- If \( f \) and \( g \) are path formulas, then \( \neg f \), \( f \lor g \), \( Xf \), \( Ff \), \( Gf \), and \( f \cup g \) are path formulas.
If \( f \) is a state formula, the notation \( M, s \models f \) means that \( f \) holds at state \( s \) in the Kripke structure \( M \).

Assume \( f_1 \) and \( f_2 \) are state formulas and \( g \) is a path formula. The relation \( M, s \models f \) is defined inductively as follows:

1. \( s \models p \iff \) atomic proposition \( p \) is true in \( s \).
2. \( s \models \neg f_1 \iff s \not\models f_1 \).
3. \( s \models f_1 \lor f_2 \iff s \models f_1 \) or \( s \models f_2 \).
4. \( s \models E(g) \iff g \) holds true for some path \( \pi \) starting with \( s \).
4. \( s \models A(g) \iff g \) holds true for every path \( \pi \) starting with \( s \).
If \( f \) is a path formula, the notation \( M, \pi \models f \) means that \( f \) holds true for path \( \pi \) in Kripke structure \( M \).

Assume \( g_1 \) and \( g_2 \) are path formulas and \( f \) is a state formula. The relation \( M, \pi \models f \) is defined inductively as follows:

1. \( \pi \models f \) if and only if \( s \) is the first state of \( \pi \) and \( s \models f \).
2. \( \pi \models \neg g_1 \) if and only if \( \pi \not\models g_1 \).
3. \( \pi \models g_1 \lor g_2 \) if and only if \( \pi \models g_1 \) or \( \pi \models g_2 \).
4. \( \pi \models X g_1 \) if and only if \( \pi^1 \models g_1 \).
5. \( \pi \models F g_1 \) if and only if there exists some \( k \geq 0 \) such that \( \pi^k \models g_1 \).
6. \( \pi \models G g_1 \) if and only if for every \( k \geq 0 \), \( \pi^k \models g_1 \).
7. \( \pi \models g_1 U g_2 \) if and only if there exists a \( k \geq 0 \) such that \( \pi^k \models g_2 \) and \( \pi^j \models g_1 \) for \( 0 \leq j < k \).

Recall: For \( \pi = (s_0, s_1, \ldots) \), we write \( \pi^i \) to denote the suffix starting with \( s_i \).
Notice that $Fp$, $FFp$, $FFFFp$, etc., hold true for a path $\pi$ even if $p$ holds true at only the initial state in the path $\pi$.

- In CTL*, the ‘future’ includes the present state. (States have temporal duration, so if we’re presently in state $s$ at time $t$, then we’ll still be in state $s$ in the future at time $t + dt$ where $dt$ is an infinitesimally small period of time.)
Note the following:

- $\mathbf{A}(f) \equiv \neg \mathbf{E}(\neg f)$
- $\mathbf{F} f \equiv (\text{true } \mathbf{U} f)$
  
  (Recall: $\pi \models g_1 \mathbf{U} g_2 \iff$ there exists a $k \geq 0$ such that $\pi^k \models g_2$ and $\pi^j \models g_1$ for $0 \leq j < k$.)
- $\mathbf{G} f \equiv \neg \mathbf{F} \neg f$

So, given any CTL* formula, we can rewrite it without using the operators $\mathbf{A}$, $\mathbf{F}$, or $\mathbf{G}$. 
The Logic CTL

CTL is a restricted subset of CTL* that permits only branching-time operators—each of the linear-time operators G, F, X, and U must be immediately preceded by a path quantifier.

More precisely, CTL is the subset of CTL* that is obtained if the following two rules are used to specify the syntax of path formulas.

- If $f$ and $g$ are state formulas, then $Xf$ and $fUg$ are path formulas.
- If $f$ is a path formula, then so is $\neg f$.

Example: $AG(EFp)$
Linear temporal logic (LTL), on the other hand, consists of formulas that have the form $\mathbf{A} f$ where $f$ is a path formula in which the only state subformulas permitted are atomic propositions.

More precisely, a path formula is either:

- If $p \in AP$, then $p$ is a path formula.
- If $f$ and $g$ are path formulas, then $\neg f$, $f \lor g$, $\mathbf{X} f$, and $f \mathbf{U} g$ are path formulas.

Example: $\mathbf{A}(\mathbf{FG} p)$
It can be shown that the three logics discussed in this section have different expressive powers.

For example, there is no CTL formula that is equivalent to the LTL formula $A(FG\,p)$.

Likewise, there is no LTL formula that is equivalent to the CTL formula $AG(EF\,p)$.

The disjunction $A(FG\,p) \lor AG(EF\,p)$ is a CTL* formula that is not expressible in either CTL or LTL.
Consider the following Kripke structure:

Are there any paths starting with $S_1$ for which $G p$ is true?
Starting with $S_2$?
Starting with $S_3$?

At which states does $AG p$ hold true?
At which states does $AF AG p$ hold true?
Does $FG p$ hold true for all paths starting with $S_1$?
There are eight basic CTL operators:

- $AX$ and $EX$,
- $AG$ and $EG$,
- $AF$ and $EF$,
- $AU$ and $EU$

Each of these can be expressed in terms of $EX$, $EG$, and $EU$:

- $AX\ f = \neg EX(\neg f)$
- $AG\ f = \neg EF(\neg f)$
- $AF\ f = \neg EG(\neg f)$
- $EF\ f = E[true \ U \ f]$
- $A[f \ U \ g] \equiv \neg E[\neg g \ U \ \neg f \ \land \ \neg g] \ \land \ \neg EG \ \neg g$
The four most widely used CTL operators are illustrated here. Each computation tree has the state $s_0$ as its root.

$$M, s_0 \models \text{EF } g$$

$$M, s_0 \models \text{AF } g$$

$$M, s_0 \models \text{EG } g$$

$$M, s_0 \models \text{AG } g$$
Typical CTL* formulas

- **EF**(\textit{Started} \land \neg \textit{Ready}): It is possible to get to a state where \textit{Started} holds but \textit{Ready} does not hold.

- **AG**(\textit{Req} \rightarrow \textbf{AF} \textit{Ack}): If a request occurs, then it will be eventually acknowledged.

- **AG**(\textbf{AF} \textit{DeviceEnabled}): The proposition \textit{DeviceEnabled} holds infinitely often on every computation path.

- **AG**(\textbf{EF} \textit{Restart}): From any state it is possible to get to the \textit{Restart} state.

- **A**(\textbf{GF} \textit{enabled} \Rightarrow \textbf{GF} \textit{executed}): if a process is infinitely-often \textit{enabled}, then it is infinitely-often \textit{executed}.

Note that the first four formulas are CTL formulas.
Model Checking Problem

Let $M$ be the state–transition graph obtained from the concurrent system.

Let $f$ be the specification expressed in temporal logic.

Find all states $s$ of $M$ such that

$$M, s \models f.$$ 

There exist very efficient model checking algorithms for the logic CTL.

The EMC Verification System

Preprocessor

Model Checker (EMC)

State Transition Graph

$10^4$ to $10^5$ states

 CTL formulas

True or Counterexample
Basic Model Checking Algorithm

- $M, s_0 \models EG a \land AF b$?
- $M, s_0 \models \neg AF \neg a \land AF b$?
Basic Model Checking Algorithm

- \( M, s_0 \models \text{EG } a \land \text{AF } b? \)
- \( M, s_0 \models \neg \text{AF } \neg a \land \text{AF } b? \)
Basic Model Checking Algorithm

- $M, s_0 \models EG a \land AF b$?
- $M, s_0 \models \neg AF \neg a \land AF b$?
Basic Model Checking Algorithm

- $M, s_0 \models \textbf{EG} \ a \land \textbf{AF} \ b$?
- $M, s_0 \models \neg \textbf{AF} \ \neg a \land \textbf{AF} \ b$?
Basic Model Checking Algorithm

- $M, s_0 \models EG \ a \land AF \ b$?
- $M, s_0 \models \neg AF \ \neg a \land AF \ b$?
Mutual Exclusion Example
Mutual Exclusion Example

c1
n2
t1 -> AF c1

t1 n2
c1 n2
AF c1

t1 t2

n1 t2
t1 -> AF c1

t1 t2
t1 t2

n1 c2
t1 -> AF c1

t1 c2
Mutual Exclusion Example
Mutual Exclusion Example

Graph showing mutual exclusion with nodes and edges indicating transitions and conditions.
Vectorized version of *EMC* algorithm on Fujitsu FACOM VP400E using an explicit representation of the state–transition graph.

State Machine size:
- 131,072 states
- 67,108,864 transitions
- 512 transitions from each state on the average.

CTL formula:
- 113 different subformulas.

Time for model checking:
- 225 seconds!!