Lecture 6
Explicit Model Checking for CTL
Model Checking \cite{CE81,QS82}

An efficient procedure that receives:
- A \textit{finite-state model} describing a system
- A \textit{temporal logic formula} describing a property

It returns
- \texttt{yes}, if the system has the property
- \texttt{no} + \texttt{Counterexample}, otherwise
**CTL Model Checking** \( M \models f \)

- **Goal:** For each \( s \), computes \( \text{label}(s) \), which is the set of subformulas of \( f \), true in \( s \)

- The Model Checking algorithm works **iteratively** on subformulas of \( f \), from **simpler** subformulas to more **complex** ones

- For checking \( \text{AG}(\text{request} \Rightarrow \text{AF grant}) \)
  - Check grant, request
  - Then check \( \text{AF grant} \)
  - Next check \( \text{request} \Rightarrow \text{AF grant} \)
  - Finally check \( \text{AG}(\text{request} \Rightarrow \text{AF grant}) \)
Model Checking $M \models f$ (cont.)

- We check subformula $g$ of $f$ only after all subformulas of $g$ have already been checked.

- For subformula $g$, the algorithm adds $g$ to $\text{label}(s)$ for every state $s$ that satisfies $g$.

- When we finish checking $g$, the following holds:
  - $g \in \text{label}(s) \iff M,s \models g$
Model Checking $M \models f$ (cont.)

Alternative description
Denote $S_g = \{ s \mid M, s \models g \}$

- The goal of model checking is to compute $S_g$ for each subformula $g$ of $f$

- In particular, $S_f$
Model Checking $M \models f$ (cont.)

- $M \models f$ if and only if $f \in \text{labels}(s)$ for all initial states $s$ of $M$

- $M \models f$ if and only if $S_0 \subseteq S_f$

- The algorithm has time complexity: $O(|M| \times |f|)$
Model Checking Atomic Propositions

• For atomic proposition \( p \in \mathcal{AP} \): 
  \( p \in \text{label}(s) \iff p \in L(s) \)

  Held by alg          Defined by M

How do we handle more complex formulas?

Observation:
• Sufficient to handle \( \neg, \lor, \text{EX}, \text{EU}, \text{EG} \)
Model Checking \( \neg, \lor \) formulas

\( \neg f_1 \): add to label(s) if and only if \( f_1 \not\in \text{labels(s)} \)

\( f_1 \lor f_2 \): add to label(s) if and only if

\( f_1 \in \text{labels(s)} \) or \( f_2 \in \text{labels(s)} \)
Model Checking $g = \text{EX } f_1$

add $g$ to label(s) if and only if $s$ has a successor $t$ such that $f_1 \in \text{labels}(t)$

procedure $\text{CheckEX } (f_1)$

\begin{align*}
T & := \{ t \mid f_1 \in \text{label}(t) \} \\
\text{while } T \neq \emptyset \text{ do} \\
\quad \text{choose } t \in T; \ T := T \setminus \{t\}; \\
\quad \text{for all } s \text{ s.t. } R(s,t) \text{ do} \\
\quad \quad \text{if EX } f_1 \notin \text{label}(s) \text{ then} \\
\quad \quad \quad \text{label}(s) := \text{label}(s) \cup \{ \text{EX } f_1\}; \\
\quad \text{end for all} \\
\text{end while}
\end{align*}
Model Checking \( g = E(f_1 \cup f_2) \)

procedure **CheckEU** \((f_1, f_2)\)

\[
T := \{ s \mid f_2 \in \text{label}(s) \}
\]

For all \( s \in T \) do 
\[
\text{label}(s) := \text{label}(s) \cup \{ E(f_1 \cup f_2) \}
\]

while \( T \neq \emptyset \) do

\[
\text{choose } s \in T ; \; T := T \setminus \{s\} ;
\]

for all \( t \) s.t. \( R(t,s) \) do

\[
\text{if } E(f_1 \cup f_2) \notin \text{label}(t) \text{ and } f_1 \in \text{label}(t) \text{ then}
\]

\[
\text{label}(t) := \text{label}(t) \cup \{ E(f_1 \cup f_2) \};
\]

\[
T := T \cup \{t\}
\]

end for all

end while

Do not add a state to \( T \) more than once
Example $g = E(f_1 \cup f_2)$
• How shall we handle \( g = EF f_1 \) ?

**Remark:**

We transform a logical question of \( M,s \models f \) to a graph traversal algorithm
Model Checking \( g = EG f_1 \)

\[ s \models EG f_1 \]

iff

There is a path \( \pi \), starting at \( s \), such that \( \pi \models G f_1 \)

iff

There is a path from \( s \) to a strongly connected component, where all states satisfy \( f_1 \)
Model Checking $g = \text{EG } f_1$

- A **Strongly Connected Component (SCC)** in a graph is a subgraph $C$ s.t. **every node** in $C$ is reachable from **any other node** in $C$ via **nodes** in $C$

- An SCC $C$ is **maximal (MSCC)** if it is not contained in **any other SCC** in the graph
- $C$ is **nontrivial** if it contains at least one edge. **Otherwise**, it is **trivial**

**Tarjan** has a linear algorithm in $O(|S|+|R|)$ for finding all MSCCs in a graph, including the trivial SCCs.
Model Checking $g = \text{EG } f_1$

Why using maximal SCCs?

Complexity concerns:

There are up to $2^{|S|}$ non-maximal SCCs in $M$

Number of maximal SCCs is at most $|S|$

- Disjoint
- Overall number of states is $|S|$
Model Checking $g = EG f_1$

Reduced structure for $M$ and $f_1$:
Remove from $M$ all states s.t. $f_1 \notin \text{label}(s)$

Resulting model: $M' = (S', R', L')$
- $S' = \{ s \mid M, s \models f_1 \}$
- $R' = (S' \times S') \cap R$
- $L'(s') = L(s')$ for every $s' \in S'$

Theorem: $M, s \models EG f_1$ iff
1. $s \in S'$ and
2. There is a path in $M'$ from $s$ to some state in a nontrivial strongly connected component of $M'$

$R'$ might no longer be total
Model Checking $g = \text{EG } f_1$

procedure \textbf{CheckEG} ($f_1$)

\begin{align*}
S' & := \{ s \mid f_1 \in \text{label}(s) \} \\
\text{SCC} & := \{ C \mid C \text{ is a nontrivial SCC of } M' \} \\
T & := \bigcup_{C \in \text{SCC}} \{ s \mid s \in C \}
\end{align*}

For all $s \in T$ do $\text{label}(s) := \text{label}(s) \cup \{ \text{EG } f_1 \}$

while $T \neq \emptyset$ do

\begin{align*}
& \text{choose } s \in T; \ T := T \setminus \{s\}; \\
& \text{for all } t \in S' \text{ s.t. } R(t,s) \text{ do} \\
& \hspace{1em} \text{if } \text{EG } f_1 \notin \text{label}(t) \text{ then} \\
& \hspace{2em} \text{label}(t) := \text{label}(t) \cup \{ \text{EG } f_1 \}; \\
& \hspace{2em} T := T \cup \{t\}
\end{align*}

end for all

end while
Complexity for EG $f_1$

- Computing $M'$: $O(|S| + |R|)$
- Computing SCCs using Tarjan’s algorithm: $O(|S'| + |R'|)$
- Labeling all states in SCCs: $O(|S'|)$
- Backward traversal: $O(|S'| + |R'|)$

Overall: $O(|S| + |R|) = O(M)$
Theorem: $M, s \models EG f_1$ iff

1. $s \in S'$ and
2. There is a path in $M'$ from $s$ to some state in a nontrivial strongly connected component of $M'$

Proof:
Model Checking Complexity

- Each subformula requires $O(|M|)$
- Number of subformulas: $O(|f|)$
- Total: $O(|M| \times |f|)$
Microwave Example

Diagram of microwave states:
- **Start Error**
- **Close**
- **Close Heat**

Transitions:
1. Start → Close
2. Close → Start Error
3. Close → Open
4. Close Heat → Cook
5. Start Error → Close
6. Start Error → Close
7. Close Heat → Start Close Heat

Actions:
- Start
- Close
- Open
- Reset
- Done
- Warmup
- Cook
Property

- \( AG (\text{Start} \rightarrow \text{AF Heat}) \)
- \( \neg EF (\text{Start} \land \text{EG} \neg \text{Heat}) \)
- \( \neg E (\text{true} U (\text{Start} \land \text{EG} \neg \text{Heat})) \)

Instead of writing the formulas in label(s) for each \( s \), Use \( S(f) \) to denote the set of states s.t. \( f \in \text{label(s)} \)
\[ \neg E \ (\text{true} \ U \ (\text{Start} \land \ EG \ \neg \text{Heat})) \]

\[ S(\text{Start}) : \{2, 5, 6, 7\} \]
\[ S(\neg \text{Heat}) : \{1, 2, 3, 5, 6\} \]
\[ S(\neg \text{EG \neg Heat}) : \{1, 2, 3, 5\} \]
\[ \neg E (\text{true } U (\text{Start } \land EG \neg \text{Heat})) \]

- \( S(\text{Start}) : \{2,5,6,7\} \)
- \( S(\neg \text{Heat}): \{1,2,3,5,6\} \)
- \( S(EG \neg \text{Heat}): \{1,2,3,5\} \)

Graph:

- Start
  - \( \text{Start Error} \)
    - \( \text{Start} \)
    - \( \text{Close} \)
    - \( \text{Error} \)
- Close
  - Open
  - Start
- Close Heat
  - Start
  - Close
  - Heat
  - Cook
- Start Error
  - Close
  - Open

Transitions:

- Start to Close: Open
- Close to Start Heat: Start
- Start Error to Close: Close
- Close to Error: Close
- Error to Start: Start
- Start Heat to Close Heat: Cook
- Close Heat to Start Error: Start
- Cook to Close Heat: Warmup

Sets:

- \( S(\text{Start } \land EG \neg \text{Heat}): \{2,5\} \)
- \( S(\text{EU}): \{1,2,3,4,5,6,7\} \)
- \( S(\text{f}): \emptyset \)
Explicit Model Checking for Fair CTL
Motivation
Fair CTL ($\text{CTL}^F$)

- Same syntax as CTL
- Different semantics

$\text{CTL}^F$ formulas are interpreted over fair Kripke structures
Fair Kripke Structures

Fair Kripke structure $M = (S, S_0, R, L, F)$
- $S, S_0, R, L$ - as before
- $F \subseteq 2^S$ is a set of fairness constraints

- $F = \{ P_1, \ldots, P_k \}$ where
  - $P_i \subseteq S$
  - or
  - $P_i$ is a CTL formula
Fairness

Fair paths:

- \( \pi = s_0, s_1, s_2, \ldots \)
- \( \text{inf}(\pi) = \{ s | s = s_i \text{ for infinitely many } i \} \)

\( \pi \) is fair if for every \( P \in F \), \( \text{inf}(\pi) \cap P \neq \emptyset \)
Example

- $F = \{ \{1,4\}, \{1,6\}, \{3\} \}$

- A path $\pi$ is fair iff $\inf(\pi)$ includes one of the sets:
  - $\{1, 3\}$
  - $\{4, 6, 3\}$
  - Or their extensions
Semantics of Fair CTL

- $M,s \models_F \text{EX } \psi \iff$ there exists a fair path $\pi = s_0,s_1,\ldots$ from $s$ such that $M, s_1 \models_F \psi$

- $M,s \models_F \text{AX } \psi \iff$ for every fair path $\pi = s_0,s_1,\ldots$ from $s$, $M, s_1 \models_F \psi$

- Similarly for $\text{EG}, \text{AG}, \text{EU}, \text{AU}, \ldots$
Examples

Fairness constraints for hardware design, expressed as formulas:

- One input that should be 1 infinitely often
- K inputs, each should be 1 infinitely often
- K inputs that should be 1 together infinitely often