Consider the Needham-Schröder Authentication Protocol

\[ \{A, N_A\}_{K_B} \]
\[ A \xrightarrow{} B \]
\[ \{N_A, N_B\}_{K_A} \]
\[ A \xleftarrow{} B \]
\[ \{N_B\}_{K_B} \]
\[ A \xrightarrow{} B \]

How do you know the protocol is “correct?”
- Prove it by hand
  - sometimes it is very tedious and thus error-prone
- Verify it by machine
  - Let’s do it!
• Here is the buggy trace found by OMocha

\[
\begin{align*}
A & \quad \xrightarrow{\{A,N_A\}_{K_E}} \quad E \\
A & \quad \xleftarrow{\{N_A,N_B\}_{K_A}} \quad E \\
A & \quad \xrightarrow{\{N_B\}_{K_E}} \quad E
\end{align*}
\]

\[
\begin{align*}
E & \quad \xrightarrow{\{A,N_A\}_{K_B}} \quad B \\
E & \quad \xleftarrow{\{N_A,N_B\}_{K_A}} \quad B \\
E & \quad \xrightarrow{\{N_B\}_{K_B}} \quad B
\end{align*}
\]
Generally, it is undecidable to verify an arbitrary property on an arbitrary algorithm.

Model checking hence focuses on verifying limited classes of properties on restricted computation models. These include:
- linear temporal logic
- computational tree logic
- $\mu$-calculus
- finite-state automata
- pushdown automata
- Petri nets

In this lecture, we will discuss automatic verification of linear temporal logic and computational tree logic on finite-state automata.

Moreover, you will have a chance to use tools to verify some protocols automatically.
Relation to Other Topics

- from logic to temporal logic
- “realistic” functional programming
- applying type system
1 Temporal Logic
   - Linear Temporal Logic
   - Computation Tree Logic

2 Automata Theory
   - Finite-State Automata for Finite Strings
   - Finite-State Automata for Infinite Strings

3 LTL Model Checking
   - From LTL to Büchi Automata
   - Language Containment

4 CTL Model Checking
   - Explicit-State Model Checking
   - Symbolic Model Checking
   - Bounded Model Checking
   - Induction
• Let $\text{AP}$ be the set of atomic propositions. A Kripke structure $K = (Q, Q_0, \delta, L)$ is a triple where
  • $Q$ is a set of states;
  • $Q_0 \subseteq Q$ is the set of initial states;
  • $\delta \subseteq Q \times Q$ is a (total) transition relation;
  • $L : Q \to 2^{\text{AP}}$

• As usual, we write $q \longrightarrow q'$ for $(q, q') \in \delta$

• A computation path from $q$ is an infinite sequence of states $\pi = p_0 p_1 \cdots p_n \cdots$ with $p_0 = q$ and $p_i \to p_{i+1}$ for $0 \leq i$

• Define $\pi(i) = p_i p_{i+1} \cdots$
Linear Temporal Logic – Syntax

- An atomic proposition is an LTL formula
- If $f$ and $g$ are LTL formulae
  - $\neg f$ and $f \lor g$ are LTL formulae
  - $fUg$ is an LTL formula
Let $K = (Q, Q_0, \delta, L)$ be a Kripke structure.

Given a computation path $\pi = p_0 p_1 \cdots p_n \cdots$ and an LTL formula $f$, define the satisfaction relation $K, \pi \models f$ by

- $K, \pi \models ap$ if $ap \in L(p_0)$
- $K, \pi \models \neg f$ if not $K, \pi \models f$
- $K, \pi \models f \lor g$ if $K, \pi \models f$ or $K, \pi \models g$
- $K, \pi \models Xf$ if $K, \pi(1) \models f$
- $K, \pi \models fu g$ if there is a $k \geq 0$ such that $K, \pi(k) \models g$ and $K, \pi(j) \models f$ for $0 \leq j < k$

We will use the following abbreviation:

$$f \land g \equiv \neg(\neg f \lor \neg g) \quad \text{F}f \equiv \text{trueU}f$$

$$Gf \equiv \neg\text{F}\neg f$$
• $K, \pi \models ap$: $ap$ holds initially
• $K, \pi \models Xf$: $f$ holds at next position
• $K, \pi \models fUg$: $f$ holds until $g$ holds
• $K, \pi \models Ff$: $f$ holds eventually
• $K, \pi \models Gf$: $f$ always holds
Computational Tree Logic – Syntax

- An atomic proposition is a CTL formula
- If $f$ and $g$ are CTL formulae
  - $\neg f$ and $f \lor g$ are CTL formulae
  - $\mathbf{A}(fUg)$ and $\mathbf{E}(fUg)$ are CTL formulae
Let $K = (Q, Q_0, \delta, L)$ be a Kripke structure. Given a state $q \in Q$ and a CTL formula $f$, define the satisfaction relation $K, q \models f$ as follows:

- $K, q \models ap$ if $ap \in L(q)$
- $K, q \models \neg f$ if not $K, q \models f$
- $K, q \models f \lor g$ if $K, q \models f$ or $K, q \models g$
- $K, q \models \text{EX} f$ if $K, q' \models f$ for some $q'$ with $q \rightarrow q'$
- $K, q \models \text{A}(f \text{U} g)$ if $K, \pi \models f \text{U} g$ for all computation path $\pi$ from $q$
- $K, q \models \text{E}(f \text{U} g)$ if $K, \pi \models f \text{U} g$ for some computation path $\pi$ from $q$

We will use the following abbreviation:

\[
\begin{align*}
\text{AX} f & \equiv \neg \text{EX} \neg f \\
\text{AF} f & \equiv \text{A}(\text{true U} f) \\
\text{EF} f & \equiv \text{E}(\text{true U} f) \\
\text{AG} f & \equiv \neg \text{EF} \neg f \\
\text{EG} f & \equiv \neg \text{AF} \neg f
\end{align*}
\]
K, q \models ap: \ ap \ holds \ at \ q

K, q \models AXf: \ f \ holds \ at \ the \ next \ position \ in \ all \ paths

K, q \models EXf: \ f \ holds \ at \ the \ next \ position \ in \ some \ paths

K, q \models AFf: \ f \ holds \ for \ all \ paths \ from \ q \ eventually

K, q \models EFf: \ f \ holds \ for \ some \ path \ from \ q \ eventually

K, q \models AGf: \ f \ always \ holds \ for \ all \ paths \ from \ q

K, q \models EGf: \ f \ always \ holds \ for \ some \ path \ from \ q

K, q \models A(fUg): \ f \ holds \ until \ g \ holds \ for \ all \ paths \ from \ q

K, q \models E(fUg): \ f \ holds \ until \ g \ holds \ for \ some \ path \ from \ q
LTL and CTL are not comparable
- There is a property on some Kripke structure, which is expressible by LTL but not CTL and vice versa

\[ \text{FG} a \] holds for all computation paths from \( q_0 \)
- but \( \text{AFAG} a \) does not hold on \( q_0 \)
Automata Theory

- Finite State Automata and Regular Languages
  - a simple model accepts finite strings
- \(\omega\)-Automata and \(\omega\)-Regular Languages
  - an extension of simple model accepts infinite strings
Strings and Languages

- Consider a set of *alphabets* $\Sigma$
- A *string* $\alpha$ is a finite sequence of symbols $a_1a_2\cdots a_n$
  - $a$, $abc$, verification, …
- The *length* of a string $\alpha = a_1a_2\cdots a_n$ is $n$
- The string of length 0 is called *empty string* and denoted by $\varepsilon$
- The set of all strings over $\Sigma$ is denoted by $\Sigma^*$
- A subset of $\Sigma^*$ is called a *language*
A finite state automaton (or finite state machine) is a tuple $M = (\Sigma, Q, Q_0, \delta, F)$ where

- $\Sigma$ is a finite set of alphabets;
- $Q$ is a finite set of states;
- $Q_0 \subseteq Q$ is the set of initial states;
- $\delta \subseteq Q \times \Sigma \times Q$ is a (total) transition relation; and
- $F \subseteq Q$ is a set of final states.

We will write $q \xrightarrow{a} q'$ for $(q, a, q') \in \delta$

If $|Q_0| = 1$ and $\delta$ is in fact a function from $Q \times \Sigma$ to $Q$, we say the automaton is deterministic.
Let $M = (\Sigma, Q, Q_0, \delta, F)$ be a finite state automaton and $
alpha = a_1 a_2 \cdots a_n$

A run for $\alpha$ on $M$ is a sequence of states $p_0 p_1 \cdots p_n$ such that

- $p_0 \in Q_0$;
- $p_i \xrightarrow{a_i} p_{i+1}$ for $0 \leq i < n$

The set of runs for $\alpha$ on $M$ is denoted by $\text{Run}_M(\alpha)$

The string $\alpha$ is accepted by $M$ if there is a run $p_0 p_1 \cdots p_n \in \text{Run}_M(\alpha)$ such that $p_n \in F$

The language accepted by $M$ is denoted by $L(M)$
Example

\[ M = (\{a, b\}, \{q_0, q_1\}, \{q_0\}, \delta, \{q_0\}) \] where

- \( q_0 \xrightarrow{a} q_1 \); \( q_0 \xrightarrow{b} q_0 \); \( q_1 \xrightarrow{a} q_0 \); \( q_1 \xrightarrow{b} q_1 \)

\[ L(M) = \{ \alpha : a \text{ occurs even number of times in } \alpha \} \]
Basic Properties

- Nondeterminism does not increase expressiveness
  - A language is accepted by a deterministic finite state automaton if and only if it is accepted by a nondeterministic finite state automaton
- The class of languages accepted by finite state automaton is the class of regular languages
  - $RR'$, $R + R'$, $R^*$, $\overline{R}$
An \( \omega \)-string \( \sigma \) is an infinite sequence of symbols 
\[ s_1 s_2 \cdots s_n \cdots \]
- \( aa \cdots , 010011011 \cdots , \cdots \)

The set of all \( \omega \)-strings over \( \Sigma \) is denoted by \( \Sigma^\omega \)

A subset of \( \Sigma^\omega \) is called an \( \omega \)-language
Automata to $\omega$-Automata

• How to make finite state automata accept $\omega$-strings?
• $M = (\Sigma, Q, Q_0, \delta, F)$: a finite state automaton
• A run for $\sigma = s_1s_2\cdots s_n\cdots$ on $M$ is an infinite sequence of states $p_1p_2\cdots p_n\cdots$ such that
  • $p_0 \in Q_0$
  • $p_i \xrightarrow{s_i} p_{i+1}$ for $0 \leq i$
• How to define acceptance?
  • There is no “last” state in an infinite run
Acceptance for $\omega$-Strings

- Let $\sigma = s_1s_2 \cdots s_n \cdots$ be an $\omega$-string and $B = (\Sigma, Q, Q_0, \delta, F)$ a finite state automaton.
- A run $r$ for $\sigma$ on $B$ is an infinite sequence of states $p_1p_2 \cdots p_n \cdots$ such that
  - $p_0 \in Q_0$
  - $p_i \xrightarrow{s_i} p_{i+1}$ for $0 \leq i$
- Define $\text{Inf}_B(r)$ to be the set of states which occur infinitely many times in $r$.
- The Büchi acceptance condition for a run $r$ requires $\text{Inf}_B(r) \cap F \neq \emptyset$.
- An $\omega$-string $\sigma$ is accepted by $B$ if there is an $r \in \text{Run}_B(\sigma)$ satisfying the Büchi acceptance condition.
- A finite state automaton using the Büchi acceptance condition is called a Büchi automaton.
Example

- $bb \cdots, aabb \cdots, ababb \cdots, \cdots$
- $abb \cdots, babb \cdots, \cdots$
- $L(M) = \{ \sigma :$
  
  there are infinitely many or even number of $a$ in $\sigma$ \}
Deterministic Büchi automata is strictly less expressive than nondeterministic ones

- The language \( \{ \sigma : \sigma \text{ contains finitely many } a's \} \) is accepted by a nondeterministic Büchi automaton but not by any deterministic Büchi automaton.
Proof.

Suppose there is a deterministic Büchi automaton $B = (\Sigma, Q, \{q_0\}, \delta, F)$ accepting the same language. Then there is an $n_0$ such that $q_0 \xrightarrow{b^{n_0}} q$ for some $q \in F$. Otherwise, $B$ would not accept $b^{\omega}$, a contradiction. Similarly, there must be an $n_1$ such that $q_0 \xrightarrow{b^{n_1}a^{n_1}} q$ for some $q \in F$. Hence there are $n_0, n_1, \ldots, n_m, \ldots$ such that

$$q_0 \xrightarrow{b^{n_0}a^{n_1} \cdots a^{n_m}} q$$

for some $q \in F$. But the $\omega$-string $b^{n_0}a^{n_1} \cdots a^{n_m} \cdots$ contains infinitely many $a$’s. □
A generalized Büchi Automaton $B = (\Sigma, Q, Q_0, \delta, \mathcal{F})$ consists of

- $\Sigma$, a finite set of alphabets
- $Q$, a finite set of states
- $Q_0 \subseteq Q$, the set of initial states
- $\delta \subseteq Q \times \Sigma \times Q$, the transition relation
- $\mathcal{F} \subseteq 2^Q$, a finite class of accepting sets

The generalized Büchi acceptance condition for a run $r$ requires $\text{Inf}_B(r) \cap F \neq \emptyset$ for all $F \in \mathcal{F}$

An $\omega$-string $\sigma$ is accepted by $B$ if there is an $r \in \text{Run}_B(\sigma)$ satisfying the generalized Büchi acceptance condition.

A finite state automaton using the generalized Büchi acceptance condition is called a generalized Büchi automaton.
Let $B = (\Sigma, Q, Q_0, \delta, F)$ be a Büchi automaton. It is easy to see that $G_B = (\Sigma, Q, Q_0, \delta, \{F\})$ is a generalized Büchi automaton accepting the same language.

Conversely, let $G = (\Sigma, Q, Q_0, \delta, \mathcal{F})$ be a generalized Büchi automaton with $\mathcal{F} = \{F_0, F_1, \ldots, F_{n-1}\}$.

Construct $B_G = (\Sigma, Q \times \mathbb{N}, Q_0 \times \{0\}, \delta', Q \times \{0\})$ as follows:

- $((q, m), s, (q', m')) \in \delta'$ if and only if
  - $(q, s, q') \in \delta$
  - $m' = \begin{cases} m & \text{if } q' \not\in F_m \\ m + 1 \mod n & \text{if } q' \in F_m \end{cases}$

Intuitively, we iterate through all accepting sets by a counter.

A run visits all accepting sets infinitely many times if and only if the counter resets to 0 infinitely many times.

Büchi automata have the same expressive power as generalized Büchi automata.
Let $K = (Q, Q_0, \delta, L)$ be a Kripke structure and $f$ an LTL formula. We write $K \models f$ if $K, \pi \models f$ for all computation paths $\pi$ from a state in $Q_0$.

Given a Kripke structure $K = (Q, Q_0, \delta, L)$ and an LTL formula $f$, the **LTL model checking problem** is to decide whether $K \models f$. 
The idea is to reduce the LTL model checking problem to the language containment problem in automata theory.

Intuitively, we will:
- translate any Kripke structure to an automaton;
- translate any LTL formula to another automaton;
- check whether the language accepted by the former automaton is contained in the language accepted by the latter.
Consider any Kripke structure $K = (Q, Q_0, \delta, L)$

Let $\Sigma_{AP} = 2^{AP}$

We will construct a Büchi automaton accepting a $\omega$-language in $\Sigma_{AP}^\omega$

Define $B_K = (\Sigma_{AP}, Q \cup \{\iota\}, \{\iota\}, \delta', Q)$

- $\iota$ is a new state not in $Q$
- $(q, s, q') \in \delta'$ if $s = L(q')$ and $(q, s, q') \in \delta$
- $(\iota, s_0, q_0) \in \delta'$ if $s_0 = L(q_0)$ and $q_0 \in Q_0$

The alphabets are in fact the set of atomic propositions satisfied in the target state

Any computation path in $K$ corresponds to an $\omega$-string over $\Sigma_{AP}$ accepted by $B_K$ and vice versa. Precisely,

$L(B_K) = \{ L(\pi) : \pi \text{ is a computation path from } q_0 \in Q_0 \}$. 

For any LTL formula $f$, we would like to construct a Büchi automata $B_f$ over $\Sigma_{AP}$ accepting all $\omega$-strings satisfying $f$

Hence to check whether $K, \pi \models f$ for all $\pi$ from some state in $Q_0$ is equivalent to checking $L(B_K) \subseteq L(B_f)$
Let $f$ be an LTL formula. The *Fischer-Ladner closure* $C(f)$ is defined as follows (we identify $\neg\neg f'$ with $f'$).

$$C(f) = \{ f', \neg f' : f' \text{ is a subformula of } f \}$$

For example,

$$C(a\mathbf{U}b) = \{ a\mathbf{U}b, \neg(a\mathbf{U}b), a, \neg a, b, \neg b \}$$
Let $f$ be an LTL formula. A subset $D$ of $C(f)$ is *healthy* if it satisfies the following conditions:

- for all $f' \in C(f)$, either $f' \in D$ or $\neg f' \in D$;
- if $f'_0 \lor f'_1 \in C(f)$, then $f'_0 \lor f'_1 \in D$ iff $f'_0 \in D$ or $f'_1 \in D$;
- if $f'Ug' \in D$, then $g' \in D$ or $f' \in D$;
- if $f'Ug' \in C(f) \not\in D$, then $g' \not\in D$.  

Automaton $B_f = (\Sigma_{AP}, Q, Q_0, \delta, F)$

- $Q = \{ D : D \text{ is healthy in } C(f) \}$
- $Q_0 = \{ D_0 \in Q : f \in D_0 \}$
- $(D, s, D') \in \delta$ if
  - $s = D \cap AP$;
  - if $Xf' \in D$, then $f' \in D'$;
  - if $Xf' \in C(f) \notin D$, then $f' \notin D'$;
  - if $f'Ug' \in D$ and $g' \notin D$, then $f'Ug' \in D'$; and
  - if $f'Ug' \in C(f) \notin D$ and $f' \in D$, then $f'Ug' \notin D'$.
- $F = \{ F_0, F_1, \ldots, F_n \}$ where
  $F_i = \{ D : f'_iUg'_i \notin D \text{ or } g'_i \in D \}$ and $f'_0Ug'_0$, $f'_1Ug'_1$, ...
  $f'_nUg'_n$ are all subformulae of this form in $C(f)$
$B_{a \cup b}$

- $C(a \cup b) = \{ a \cup b, \neg(a \cup b), a, \neg a, b, \neg b \}$
- All subsets are $\emptyset$, $\{ a \}$, $\{ b \}$, $\{ a \cup b \}$, $\{ a, b \}$, $\{ a, a \cup b \}$, $\{ b, a \cup b \}$, and $\{ a, b, a \cup b \}$. All subsets are $\emptyset$, $\{ a \}$, $\{ b \}$, $\{ a \cup b \}$, $\{ a, b \}$, $\{ a, a \cup b \}$, $\{ b, a \cup b \}$, and $\{ a, b, a \cup b \}$
\( B(a \cup b) \lor (\neg a \cup b) \)

\[ C((a \cup b) \lor (\neg a \cup b)) = \{(a \cup b) \lor (\neg a \cup b), \neg((a \cup b) \lor (\neg a \cup b)), a \cup b, \neg(a \cup b), \neg a \cup b, \neg(\neg a \cup b), a, \neg a, b, \neg b\} \]

- Healthy subsets are \( \{\}, \{a\}, \{(a \cup b) \lor (\neg a \cup b), a \cup b, a\}, \{(a \cup b) \lor (\neg a \cup b), \neg a \cup b\}, \{(a \cup b) \lor (\neg a \cup b), a \cup b, \neg a \cup b, b\}, \{(a \cup b) \lor (\neg a \cup b), a \cup b, \neg a \cup b, a, b\} \)

- Why is \( \{(a \cup b) \lor (\neg a \cup b), a \cup b, b\} \) not healthy?
Checking Language Containment

- For any LTL formula $f$, $L(B_f)$ contains all $\omega$-strings over $\Sigma_{AP}$ satisfying $f$
- For any Kripke structure $K$, $L(B_K)$ contains all $\omega$-strings over $\Sigma_{AP}$ corresponding some computation path in $K$ from an initial state
- It remains to check whether $L(B_K) \subseteq L(B_f)$
- Observe that $L(B_K) \subseteq L(B_f)$ if and only if $L(B_K) \cap \overline{L(B_f)} = \emptyset$
How to check $L(B_K) \cap \overline{L(B_f)} = \emptyset$?

- How to compute $\overline{L(B_f)}$?
- How to check $L(B_K) \cap \overline{L(B_f)} = \emptyset$?
Let $M$ be a finite state automaton. Its complement automaton, $\overline{M}$, is a finite state automaton such that $L(\overline{M}) = L(M)$
- determinize $M$ and change the accepting states

Can we do it for Büchi automata?
- Not directly. Deterministic Büchi automata is strictly less expressive than Büchi automata
  - A more general deterministic $\omega$-automata is required
  - Alas, it is rather complicated
Fortunately, there is an easy way out

Observe that a computation path $\pi$ satisfies $f$ if and only if it does not satisfy $\neg f$.

Hence, $L(B_f) = L(B_{\neg f})$.

Complementation of Büchi automata is not needed!
Checking $L(B_0) \cap L(B_1) = \emptyset$

- Let $M^0$ and $M^1$ be finite state automata
- How to check $L(M^0) \cap L(M^1) = \emptyset$?
  - construct product automaton $M^0 \times M^1$ and check if $L(M^0 \times M^1) = \emptyset$
- Can we do it for Büchi automata?
- Yes!
Product Automata $B^0 \times B^1$

Let $B^0 = (\Sigma, Q^0, Q^0_0, \delta^0, F^0)$ and $B^1 = (\Sigma, Q^1, Q^1_0, \delta^1, F^1)$ be Büchi automata.

Define $B^0 \times B^1$ as follows.

- $\Sigma$ is its alphabets
- $Q^0 \times Q^1 \times \{0, 1, 2\}$ are its states
- $Q^0_0 \times Q^1_0 \times \{0\}$ are the initial states
- $Q^0 \times Q^1 \times \{2\}$ are the accepting states

Moreover, $\langle p^0, r^1, x \rangle \overset{a}{\longrightarrow} \langle q^0, s^1, y \rangle$ if:

- $p^0 \overset{a}{\rightarrow}^0 q^0$ in $B^0$;
- $r^1 \overset{a}{\rightarrow}^1 s^1$ in $B^1$; and
- $y = \begin{cases} 
1 & \text{if } x = 0 \text{ and } q^0 \in F^0 \\
2 & \text{if } x = 1 \text{ and } s^1 \in F^1 \\
0 & \text{if } x = 2 \\
x & \text{otherwise}
\end{cases}$
Example of Product Automata

Product Büchi Automaton
Automata-Theoretic LTL Model Checking Algorithm

Input: a Kripke structure $K$ and an LTL formula $f$
Output: whether $K \models f$

1. Construct $B_K$ and $B_{\neg f}$ for $K$ and $\neg f$ respectively
2. Check whether $L(B_K \times B_{\neg f}) = \emptyset$
   - if so, return \textit{PASS}
   - otherwise, return \textit{FAIL}
Let $K = (Q, Q_0, \delta, L)$ be a Kripke structure and $f$ a CTL formula. We write $K \models f$ if $K, q_0 \models f$ for all $q_0 \in Q_0$.

Given a Kripke structure $K = (Q, Q_0, \delta, L)$ and a CTL formula $f$, the *CTL model checking problem* is to decide whether $K \models f$. 
Explicit-State CTL Model Checking

- Let $K = (Q, Q_0, \delta, L)$ be a Kripke structure and $f$ a CTL formula.
- Let $Q' \subseteq Q$. Define

\[
Pre_K(Q') = \{ q : \text{there is a } q' \text{ such that } q \to q', q' \in Q' \} \\
PRE_K(Q') = \{ q : \text{for all } q' \text{ such that } q \to q', q' \in Q' \}
\]

- Define the function $\lbrack f \rbrack_K$ as follows.

\[
\lbrack ap \rbrack_K = \{ q : ap \in L(q) \} \\
\lbrack \neg f \rbrack_K = Q \setminus \lbrack f \rbrack_K \\
\lbrack f \lor g \rbrack_K = \lbrack f \rbrack_K \cup \lbrack g \rbrack_K \\
\lbrack \text{EX} f \rbrack_K = Pre_K(\lbrack f \rbrack_K) \\
\lbrack A(f U g) \rbrack_K = \lbrack g \rbrack_K \cup (\lbrack f \rbrack_K \cap PRE_K(\lbrack A(f U g) \rbrack_K)) \\
\lbrack E(f U g) \rbrack_K = \lbrack g \rbrack_K \cup (\lbrack f \rbrack_K \cap Pre_K(\lbrack E(f U g) \rbrack_K))
\]
Solving $X = G(X)$

- A function $G : 2^Q \rightarrow 2^Q$ is monotonic if $A \subseteq B$ implies $G(A) \subseteq G(B)$
- Define $G_i \subseteq Q$ as follows

  $$G_0 = \emptyset \text{ and } G_{i+1} = G(G_i)$$

- Facts
  - $G_i \subseteq G_{i+1}$ for all $i$
  - $G_i = G_{i+1}$ implies $G_j = G_i$ for all $j \geq i$
- Let $f \in \mathbb{N}$ be that $G_f = G_{f+1}$. Then $G_f = G_{f+1} = G(G_f)$. $G_f$ is a fixed point of $G$
- Let $H \subseteq Q$ be that $H = G(H)$. Then $G_i \subseteq H$ for all $i$. Hence $G_f \subseteq H$. $G_f$ is the least fixed point of $G$
Compute $\mathbb{A}(fUg)_K$

- Define

$$G(X) = \left[ g \right]_K \cup \left( \left[ f \right]_K \cap \text{PRE}_K(X) \right)$$

- $G : 2^Q \rightarrow 2^Q$ is monotonic
- If $Q$ is finite, there is an $f \in \mathbb{N}$ such that $G_f = G_{f+1}$
- Then $G_f = \mathbb{A}(fUg)_K$
- $\mathbb{E}(fUg)_K$ can be computed similarly
CTL Model Checking Algorithm

Input: a Kripke structure $K$ and a CTL formula $f$
Output: whether $K \models f$

1. Compute $[f]_K$
2. Check $Q_0 \subseteq [f]_K$
   - if so, return $PASS$
   - otherwise, return $FAIL$

However, computing $[f]_K$ is not easy when $|Q|$ is very large

Can we compute $[f]_K$ efficiently (in practice)?
Let $\mathbb{B} = \{\text{false, true}\}$ be the Boolean domain.

An $n$-ary binary function is a function from $\mathbb{B}^n$ to $\mathbb{B}$.

Decision diagrams are representations for binary functions.

\[
f(x, y) = x \lor y
\]
Binary decision diagrams are obtained by

- merging identical nodes
- removing redundant nodes

\[ f(x, y, z) = x \land (y \lor z) \]
Variable Order in BDD’s

- For any fixed variable ordering, the BDD representation is canonical
  - $f = g$ if and only if $BDD(f) = BDD(g)$
- The size of BDD’s depends on the order of variables
  - Finding optimal order is NP-hard

(y \land u) \lor ((x \land z) \land (y \lor u))
Let $f$ and $g$ be two $n$-ary binary functions. The following BDD operations are available:

- **negation.** not $BDD(f) = BDD(\neg f)$
- **conjunction.** $BDD(f)$ and $BDD(g) = BDD(f \land g)$
- **disjunction.** $BDD(f)$ or $BDD(g) = BDD(f \lor g)$
- **universal quantification.** $\forall x_i BDD(f) = BDD(\forall x_i f)$
- **existential quantification.** $\exists x_i BDD(f) = BDD(\exists x_i f)$
- **renaming.** $BDD(f)[\bar{y}/\bar{x}] = BDD(f[\bar{x} := \bar{y}])$
Given a Qualified Boolean Formula (QBF) $f$, one can evaluate $f$ as follows:

- expand all qualifiers as mentioned
- construct BDD for the expanded formula
- return the resultant BDD (either false or true)

BDD’s can in fact determine the satisfiability or validity of any propositional logic formula.
Problems in the definition of $[f]_K$ in explicit-state CTL model checking

- Set operations
  - $[f \lor g]_K = [f]_K \cup [g]_K$
- $Pre_K(Q')$ and $PRE_K(Q')$
  - functions over sets of states
- Fixed points
  - $[E(f \mathcal{U} g)]_K = [g]_K \cup ([f]_K \cap Pre_K([E(f \mathcal{U} g)]_K))$

Can we solve them by BDD’s?
For simplicity, we only consider sets over binary vectors of size $n$

- $\{000, 010, 100, 110\}$

**Characteristic function** $\chi_H$ for set $H$ is an $n$-ary binary function such that $\chi_H(x_n, x_{n-1}, \ldots, x_1) = \text{true}$ if and only if $x_nx_{n-1}\cdots x_1 \in H$

- $\chi(x_3, x_2, x_1) = \neg x_1$

Set operations correspond to logical operations

- $\chi_{\overline{H_0}} = \neg \chi_{H_0}$
- $\chi_{H_0 \cup H_1} = \chi_{H_0} \lor \chi_{H_1}$
- $\chi_{H_0 \cap H_1} = \chi_{H_0} \land \chi_{H_1}$
Let $K = (Q, Q_0, \delta, L)$ be a Kripke structure. For simplicity, assume $|Q| = 2^m$ for some $m$.

Each state $q \in Q$ can be represented by a binary vector of size $m$.

Each transition $q \rightarrow q'$ is represented by a pair of binary vectors.

Hence the transition relation $\delta$ is represented by a set of binary vectors of size $2m$. 
Consider $K = (\{q_0, q_1, \ldots, q_7\}, \{q_0, q_2, q_4, q_6\}, \delta, L)$ with $q_i \rightarrow q_{i+1} \mod 8$

- $q_0 = 000, q_1 = 001, \ldots, q_7 = 111$
- $\chi_{Q_0}(x_3, x_2, x_1) = \neg x_1$
- $\chi_{\delta}(x_3, x_2, x_1, x'_3, x'_2, x'_1) =$

\[
\begin{pmatrix}
- x_3 \land \neg x_2 \land \neg x_1 \land \neg x'_3 \land \neg x'_2 \land x'_1 \\
- x_3 \land \neg x_2 \land x_1 \land \neg x'_3 \land x'_2 \land \neg x'_1 \\
- x_3 \land x_2 \land \neg x_1 \land \neg x'_3 \land x'_2 \land x'_1 \\
- x_3 \land x_2 \land x_1 \land x'_3 \land \neg x'_2 \land \neg x'_1 \\
x_3 \land \neg x_2 \land \neg x_1 \land x'_3 \land \neg x'_2 \land x'_1 \\
x_3 \land \neg x_2 \land x_1 \land x'_3 \land x'_2 \land \neg x'_1 \\
x_3 \land x_2 \land \neg x_1 \land x'_3 \land x'_2 \land x'_1 \\
x_3 \land x_2 \land x_1 \land \neg x'_3 \land \neg x'_2 \land \neg x'_1 
\end{pmatrix}
\]

- $\chi_{\delta}(b_3, b_2, b_1, b'_3, b'_2, b'_1) =$ true if and only if $q(b_3b_2b_1)_2 \rightarrow q(b'_3b'_2b'_1)_2$ where $(i)_2$ denotes the number represented by $i$ in binary
Computing $Pre_K(Q')$ and $PRE_K(Q')$

- $Q'$ is a set of binary vectors.
- Recall $Pre_K(Q') = \{ q : \text{there is a } q' \text{ such that } q \rightarrow q', q' \in Q' \}$
- Let $\chi'_Q$ be the characteristic function obtained by renaming each $x$ to $x'$ in $\chi_Q$.
  - Say, $\chi_Q'(x_3, x_2, x_1) = \neg x_1$. Then $\chi_Q'(x_3', x_2', x_1') = \neg x_1'$
- By assumption, $|Q| = 2^m$. Hence

\[
\chi_{Pre_K(Q')}(\overline{x}) = \exists \overline{x}'. \chi_\delta(\overline{x}, \overline{x}') \land \chi'_Q
\]

\[
\chi_{PRE_K(Q')}(\overline{x}) = \exists \overline{x}'. \neg (\chi_\delta(\overline{x}, \overline{x}') \land \neg \chi'_Q)
\]
Solving \( X = G(X) \) in BDD’s

- \( X \) is a set of binary vectors and \( G \) is a set function over state sets
- \( \chi_X \) can be represented by a BDD
- \( G \) can be computed by BDD operations
- We simply compute \( G_i \) iteratively

\[
\begin{align*}
1 & \quad i = 0 \\
2 & \quad G_i = \text{BDD}(\chi_0) \\
3 & \quad \text{do} \\
4 & \quad \quad i = i + 1 \\
5 & \quad \quad G_i = G(G_{i-1}) \\
6 & \quad \text{while } G_i \neq G_{i-1} \\
7 & \quad \text{return } G_i
\end{align*}
\]
Symbolic CTL Model Checking

- Given $K = (Q, Q_0, \delta, L)$
- Encode $\delta$ and $L$ in BDD's
  - $\chi_a(\overline{x}) = 1$ if and only if $a \in L(\overline{x})$
- Compute $\text{BDD}(\chi_{[f]} K)$
- Check if $\text{BDD}(\chi_{Q_0} \land \neg \chi_{[f]} K) = \text{BDD}(\chi_{\emptyset})$
  - if so, return $\text{PASS}$
  - otherwise, return $\text{FAIL}$
Limitation of BDD’s

- Symbolic CTL model checking does not solve all our problems
  - BDD’s are hard to predicate
    - the size is very sensitive to variable ordering
  - BDD’s cannot handle real systems
    - up to 300 binary variables
  - Oftentimes, BDD’s would blow up while building transition relations
    - no information at all when it doesn’t work
- Techniques that can be scaled up are always needed
• A CTL formula is in *negative normal form* if the negation appears only before atomic propositions
  • For instance, \( \text{AF} \neg p \) is in nnf but \( \neg \text{EG} p \) is not
• Write \( K, \pi \models f \text{R} g \) if for all \( j \geq 0 \), for all \( i < j \) \( K, \pi(i) \not\models f \) implies \( K, \pi(j) \models g \)
  • Observe that \( f \text{R} g \equiv \neg (\neg f \text{U} \neg g) \)
• All CTL formula can be transformed to its negative normal form
  • Use \( \neg \neg f \equiv f, \neg (f \lor g) \equiv \neg f \land \neg g, \neg (f \land g) \equiv \neg f \lor \neg g, \)
  \( \neg \text{AX} f \equiv \text{EX} \neg f, \neg \text{EX} f \equiv \text{AX} \neg f, \neg \text{E}(f \text{U} g) \equiv \text{A}(\neg f \text{R} \neg g), \)
  \( \neg \text{A}(f \text{U} g) \equiv \text{E}(\neg f \text{R} \neg g) \)
• \( \text{ACTL} \) is a subclass of CTL, where only universal path quantifier is allowed in negative normal form
  • \( \text{AG} p \) and \( \neg \text{E}(f \text{U} g) \) are in ACTL but \( \text{EG} p \) and \( \text{AGEF} p \) are not
Satisfiability and Validity

- Consider a propositional logic formula, say,
  \[ [p \to (q \lor r)] \land [q \lor \neg r] \]

- A *truth assignment* is a mapping from propositional variables \((p, q, r, \text{etc})\) to Boolean domain.

- A propositional logic formula is *satisfiable* if there is a truth assignment that makes the formula evaluate to true.
  - For instance, the above formula can be satisfied by setting \(p = \text{false}, q = \text{true}, \text{and } r = \text{true}\).

- A propositional logic formula is *valid* if for all truth assignment, the formula evaluates to true.
  - For instance, the above formula evaluates to false when \(p = \text{true}, q = \text{false}, \text{and } r = \text{false}\). It is not valid.

- For any propositional formula \(f\), \(f\) is not satisfiable if and only if \(\neg f\) is valid.
Boolean Satisfiability

- Given a propositional logic formula, determine whether there is a satisfying truth assignment
- First NP-complete problem
- Since mid 90’s, many practical SAT solvers are available
  - by “practical”, we mean SAT solvers that can handle thousands of binary variables!
  - widely used SAT solvers are MiniSAT, zchaff, grasp
- We will use SAT solvers to solve ACTL model checking within bounded steps
Bounded Model Checking

- The idea is to verify the Kripke structure up to a fixed number of steps.
- Equivalently, bounded model checking aims to find bugs within a fixed number of steps:
  - if bugs are found, bounded model checker reports them.
  - if bugs cannot be found in the first $n$ steps, it does not guarantee the correctness of the Kripke structure.
Consider the formula $\mathbf{AX}p$ on the Kripke structure $K$

What is a bug in $K$?

By definition $K \not\models \mathbf{AX}p$ if $K, q_0 \models \neg \mathbf{AX}p$ for some $q_0 \in Q_0$

Hence our goal is to find a $q_0 \in Q_0$ such that $K, q_0 \models \mathbf{EX}\neg p$

Can it be done by SAT solvers?

Yes! Checking the satisfiability of the following formula suffices.

- $\chi_{Q_0}(\overline{x}_0) \land \chi_{\delta}(\overline{x}_0, \overline{x}_1) \land \neg \chi_p(\overline{x}_1)$

What about verifying $\mathbf{EX}p$?

- Not directly. Checking the satisfiability of $[\chi_{Q_0}(\overline{x}_0) \land \chi_{\delta}(\overline{x}_0, \overline{x}_1)] \rightarrow \neg \chi_p(\overline{x}_1)$ does not work. Why?
Let $f$ be an ACTL formula

¬$f$ is equivalent to a CTL formula where only existential path quantifiers occur

- for instance, $\neg \text{AGAF} p \equiv \text{EFEG} \neg p$

It suffices to find a $q_0 \in Q_0$ such that $K, q_0 \models \neg f$

- if so, a bug is found and can be reported
- if not, we conclude there is no bug up to the bound
Bounded ACTL Model Checking – Example

- Let $K = (Q, Q_0, \delta, L)$ be a Kripke structure with $|Q| = 2^m$
- Consider verifying $K \models \text{AG}a$ up to the first 3 steps
- We hence try to find a $q_0 \in Q_0$ such that $K, q_0 \models \text{EF}\neg a$
- Consider the following propositional formula

\[
F_3(x_0, x_1, x_2, x_3)
= \chi_{Q_0}(x_0) \land \neg \chi_a(x_0) \lor \\
\chi_{Q_0}(x_0) \land \chi_\delta(x_0, x_1) \land \neg \chi_a(x_1) \lor \\
\chi_{Q_0}(x_0) \land \chi_\delta(x_0, x_1) \land \chi_\delta(x_1, x_2) \land \neg \chi_a(x_2) \lor \\
\chi_{Q_0}(x_0) \land \chi_\delta(x_0, x_1) \land \chi_\delta(x_1, x_2) \land \chi_\delta(x_2, x_3) \land \neg \chi_a(x_3)
\]

- Then $F_3(x_0, x_1, x_2, x_3)$ is satisfiable if and only if there is a state $q$ reachable from some $q_0 \in Q_0$ in three steps such that $a \not\in L(q)$. 
Notes about ACTL Bounded Model Checking

**Pros**
- Partial information. Even though we cannot verify the system, we do know it is correct up to a certain number of steps
- Scalability. Modern SAT solvers can handle thousands of binary variables. We can check larger systems

**Cons**
- A bit tricky to verify systems for sure. Extending bounded model checking to model checking is not straightforward
- Does not work well for general CTL formulae. Alternation of universal and existential path quantifiers causes problems
It is a bit tricky to verify ACTL by SAT solvers completely.

We will introduce a complete SAT-based verification algorithm for invariant checking.

An invariant is an atomic proposition which is satisfied in all states reachable from initial states.

- $a$ is an invariant if and only if $\text{AG}a$ and $Ga$ hold.

We will apply inductive reasoning in invariant checking!
Induction

- Consider verifying AGa on $K = (Q, Q_0, \delta, L)$
- Suppose we know the following
  - $a \in L(q_0)$ for all $q_0 \in Q_0$
  - for all $q$ and $q'$ such that $q \rightarrow q'$, $a \in L(q)$ implies $a \in L(q')$
- Can we conclude $K \models AGa$?
  - Yes!

Proof.

If $K \not\models AGa$, there is a $q_0, q_1, \ldots, q_m \in Q$ such that
  - $q_0 \in Q_0$
  - $q_i \rightarrow q_{i+1}$ for $0 \leq i < m$
  - $a \in L(q_i)$ for $0 \leq i < m$ but $a \not\in L(q_m)$

Then $q_m \not\in Q_0$ by the basis. Moreover, $a \in L(q_m)$ for $a \in L(q_{m-1})$ and $q_{m-1} \rightarrow q_m$ by inductive step
The idea can be generalized to more than one step

- $a \in L(q_i)$ for all $q_i \in Q_i$ and $0 \leq i < k$ where

$$Q_i = \{q' : q_0 \rightarrow q_1 \rightarrow \cdots \rightarrow q_i \text{ for some } q_0 \in Q_0 \}$$

- $a \in L(q_i)$ for $0 \leq i < k$ implies $a \in L(q_k)$ where $q_i \rightarrow q_{i+1}$ for $0 \leq i < k$

- How can we perform $k$-induction by SAT solvers
Induction by SAT Solvers

- Consider the following two SAT problems
  
  - $\chi_{Q_0}(\overline{x}_0) \land \neg \chi_a(\overline{x}_0)$
  
  - $\chi_a(\overline{y}_0) \land \chi_\delta(\overline{y}_0, \overline{y}_1) \land \neg \chi_a(\overline{y}_1)$

- What do they mean if they are not satisfiable?
  
  - it’s impossible to have an initial state not satisfying $a$
  
    - all initial states satisfy $a$
  
  - it’s impossible to reach a state not satisfying $a$ from a state satisfying $a$
  
    - any state satisfying $a$ can only go to states satisfying $a$

- Hence, if these propositional logic formulae are unsatisfiable, we conclude $a$ is an invariant
The technique can be generalized to $k$-induction

Consider the following propositional logic formulae

- $\chi Q_0(\overline{x}_0) \land \neg \chi a(\overline{x}_0)$
- $\chi Q_0(\overline{x}_0) \land \chi \delta(\overline{x}_0, \overline{x}_1) \land \neg \chi a(\overline{x}_0)$
- $\ldots$
- $\chi Q_0(\overline{x}_0) \land \chi \delta(\overline{x}_0, \overline{x}_1) \land \cdots \chi \delta(\overline{x}_{k-2}, \overline{x}_{k-1}) \land \neg \chi a(\overline{x}_{k-1})$
- $\chi a(\overline{y}_0) \land \chi \delta(\overline{y}_0, \overline{y}_1) \land \chi a(\overline{y}_1) \land \chi \delta(\overline{y}_1, \overline{y}_2) \land \cdots \chi a(\overline{y}_{k-1}) \land \chi \delta(\overline{y}_{k-1}, \overline{y}_k) \land \neg \chi a(\overline{y}_k)$

If all of them are unsatisfiable, we conclude $a$ is an invariant

- what if some of them are satisfiable?
When \( k \)-induction fails, there are two possibilities

- some of basis formulae are satisfiable
  \[ \chi Q_0(\overline{x}_0) \land \neg \chi a(\overline{x}_0), \chi Q_0(\overline{x}_0) \land \chi \delta(\overline{x}_0, \overline{x}_1) \land \neg \chi a(\overline{x}_0), \ldots \]
  - a counterexample is found!

- the inductive formula is satisfiable
  \[ \chi a(\overline{y}_0) \land \chi \delta(\overline{y}_0, \overline{y}_1) \land \chi a(\overline{y}_1) \land \chi \delta(\overline{y}_1, \overline{y}_2) \land \cdots \chi a(\overline{y}_{k-1}) \land \chi \delta(\overline{y}_{k-1}, \overline{y}_k) \land \neg \chi a(\overline{y}_k) \]

If the inductive step is satisfiable, one increases \( k \) and performs \( k + 1 \)-induction

- if \( a \) is not an invariant, there is a \( k \) such that \( k \)-induction fails in the basis
- the basis will be satisfiable for some \( k \)
- what if \( a \) is indeed an invariant?
  - can we always establish invariance by induction? not necessarily!
If the basis formulae are not satisfiable but the inductive formula is satisfiable, when can we conclude the invariant checking passes?

Idea: the shortest counterexample cannot be longer than the diameter of reachability graph

- The reachability graph consists of states as nodes and transitions as edges

Proof.

Let $k$ be the diameter of reachability graph. Consider $q_0 \rightarrow q_1 \rightarrow \cdots \rightarrow q_k \rightarrow q_{k+1}$. Then $q_i = q_j$ for some $0 \leq i < j \leq k + 1$. Hence $q_0 \rightarrow q_1 \rightarrow \cdots \rightarrow q_i \rightarrow q_{j+1} \rightarrow \cdots \rightarrow q_{k+1}$ is a shorter computation path to $q_{k+1}$
It suffices to find the diameter of reachability graph

Consider the following formula

\[ \chi_{Q_0}(\overline{x}_0) \land \]
\[ \chi_\delta(\overline{x}_0, \overline{x}_1) \land \overline{x}_1 \neq \overline{x}_0 \land \]
\[ \chi_\delta(\overline{x}_1, \overline{x}_2) \land \overline{x}_2 \neq \overline{x}_0 \land \overline{x}_2 \neq \overline{x}_1 \land \]
\[ \ldots \]
\[ \chi_\delta(\overline{x}_{k-1}, \overline{x}_k) \land \overline{x}_k \neq \overline{x}_0 \land \overline{x}_k \neq \overline{x}_1 \land \cdots \land \overline{x}_k \neq \overline{x}_{k-1} \]

If the formula is unsatisfiable for some \( k \), we know the diameter of reachability graph is \( k - 1 \)
Here is the algorithm

Input: $K = (Q, Q_0, \delta, L)$ and an atomic proposition $a$
Output: whether $a$ is an invariant in $K$

1. $k := 1$
2. loop
3. perform $k$-induction
4. if a counterexample is found, return $FAIL$
5. if the diameter is $k$, return $PASS$
6. $k := k + 1$
-we have introduced
  - both LTL and CTL
  - an automata-theoretic LTL model checking algorithm
  - a BDD-based CTL model checking algorithm
  - a SAT-based invariant checking algorithm
  - SPIN and NuSMV
Current Research

- Finite-state models to infinite-state models
  - context-free processes and pushdown systems
- Proof theory + model checking = ?
- Computational learning theory
- SAT-based model checking algorithm for universal $\mu$-calculus