Outline

1 Automata-Theory Overview
   - Language Containment
   - Automata on Finite Words
   - Automata on Infinite Words
   - Emptiness Checking

2 The Automata-Theoretic Approach to Model Checking
   - Automata-Theoretic LTL Model Checking
   - From Kripke Structures to Büchi Automata
   - From LTL Formulas to Büchi Automata: generalities
   - On-the-fly construction of Büchi Automata from LTL
   - Complexity

3 Exercises
System’s computations

The behaviors (computations) of a system can be seen as sequences of assignments to propositions.

MODULE main
VAR done: Boolean;
ASSIGN
  init(done):=0;
  next(done):= case
    !done: {0,1};
    done: done;
  esac;

Since the state space is finite, the set of computations can be represented by a finite automaton.
Correct computations

- Some computations are correct and others are not acceptable.
- We can build an automaton for the set of all acceptable computations.
- Example: eventually, done will be true forever.
Language Containment Problem

Solution to the verification problem

⇒ Check if language of the system automaton is contained in the language accepted by the property automaton.

The language containment problem is the problem of deciding if a language is a subset of another language.

\[ \mathcal{L}(A_1) \subseteq \mathcal{L}(A_2) \iff \mathcal{L}(A_1) \cap \overline{\mathcal{L}(A_2)} = \emptyset \]

In order to solve the language containment problem, we need to know:

(i) how to complement an automaton,
(ii) how to intersect two automata,
(iii) how to check the language emptiness of an automaton.
Finite Word Languages

- An **Alphabet** $\Sigma$ is a collection of symbols (letters).
  E.g. $\Sigma = \{a, b\}$.

- A **finite word** is a finite sequence of letters. (E.g. $aabb$.)
  The set of all finite words is denoted by $\Sigma^*$.

- A **language** $U$ is a set of words, i.e. $U \subseteq \Sigma^*$.
  Example: Words over $\Sigma = \{a, b\}$ with equal number of $a$’s and $b$’s.
  (E.g. $aabb$ or $abba$.)

- **Language recognition problem**: determine whether a word belongs to a language.

- **Automata** are computational devices able to solve language recognition problems.
Finite State Automata

- Basic model of computational systems with finite memory.
- Widely applicable
  - Embedded System Controllers.
    - Languages: Ester-el, Lustre, Verilog.
  - Synchronous Circuits.
  - Regular Expression Pattern Matching
    - Grep, Lex, Emacs.
  - Protocols
    - Network Protocols
    - Architecture: Bus, Cache Coherence, Telephony,...
Notation

\( a, b \in \Sigma \) finite alphabet.
\( u, v, w \in \Sigma^* \) finite words.
\( \epsilon \) empty word.
\( u.v \) concatenation.
\( u^i = u.u \ldots u \) repeated \( i \)-times.
\( U, V \subseteq \Sigma^* \) Finite word languages.
FSA Definition

Definition

A Nondeterministic Finite State Automaton (NFA) is $(Q, \Sigma, \delta, I, F)$ s.t.
- $Q$ Finite set of states.
- $\Sigma$ is a finite alphabet
- $I \subseteq Q$ set of initial states.
- $F \subseteq Q$ set of final states.
- $\delta \subseteq Q \times \Sigma \times Q$ transition relation (edges).

We use $q \xrightarrow{a} q'$ to denote $(q, a, q') \in \delta$.

Definition

A Deterministic Finite State Automaton (DFA) is a NFA s.t.:
- $\delta : Q \times \Sigma \rightarrow Q$ is a total function
- Single initial state $I = \{q_0\}$. 
Regular Languages

- A run of NFA $A$ on $u = a_0, a_1, \ldots, a_{n-1}$ is a finite sequence of states $q_0, q_1, \ldots, q_n$ s.t. $q_0 \in I$ and $q_i \xrightarrow{a_i} q_{i+1}$ for $0 \leq i < n$.
- An accepting run is one where $q_n \in F$.
- The language accepted by $A$ is $L(A) = \{ u \in \Sigma^* \mid A \text{ has an accepting run on } u \}$.
- The languages accepted by a NFA are called regular languages.
Finite State Automata: examples

- The DFA $A_1$ over $\Sigma = \{a, b\}$:

  \[ a \xrightarrow{} s_1 \xrightarrow{} b \xrightarrow{} s_2 \]

  Recognizes words which do not end in $b$.

- The NFA $A_2$ over $\Sigma = \{a, b\}$:

  \[ a, b \xrightarrow{} s_1 \xrightarrow{} b \xrightarrow{} s_2 \]

  Recognizes words which end in $b$. 
Determinisation

**Theorem (determinisation)**

Given a NFA $A$ we can construct a DFA $A'$ s.t. $\mathcal{L}(A) = \mathcal{L}(A')$.

Size: $|A'| = 2^{O(|A|)}$.

- Each state of $A'$ corresponds to a set $\{s_1, \ldots, s_j\}$ of states in $A$ ($Q' \subseteq 2^Q$), with the intended meaning that:
  - $A'$ is in the state $\{s_1, \ldots, s_j\}$ if $A$ is in one of the states $s_1, \ldots, s_j$.
- The deterministic transition relation $\delta' : 2^Q \times \Sigma \rightarrow 2^Q$ is:
  - $\{s\} \xrightarrow{a} \{s_i | s \xrightarrow{a} s_i\}$
  - $\{s_1, \ldots, s_j, \ldots, s_n\} \xrightarrow{a} \bigcup_{j=1}^{n} \{s_i | s_j \xrightarrow{a} s_i\}$
- The (unique) initial state is $I' = \text{def} \{s_i | s_i \in I\}$
- The set of final states $F'$ is such that $\{s_1, \ldots, s_n\} \in F'$ iff $s_i \in F$ for some $i \in \{1, \ldots, n\}$.
Determinisation [cont.]

- NFA $A_2$: Words which end in $b$.

\[
\begin{array}{c}
\text{a,b} \\
\downarrow \\
\text{s}_1 \\
\downarrow \text{b} \\
\text{s}_2 \\
\end{array}
\]

- $A_2$ can be determinised into the automaton $DA_2$ below.
  (#States = $2^Q$.)
Closure Properties

Theorem (Boolean closure)

Given NFA $A_1, A_2$ over $\Sigma$ we can construct NFA $A$ over $\Sigma$ s.t.

- $L(A) = \overline{L(A_1)}$ (Complement). $|A| = 2^O(|A_1|)$.
- $L(A) = L(A_1) \cup L(A_2)$ (union). $|A| = |A_1| + |A_2|$.
- $L(A) = L(A_1) \cap L(A_2)$ (intersection). $|A| = |A_1| \cdot |A_2|$.
Complementation of a NFA

A NFA $A = (Q, \Sigma, \delta, I, F)$ is complemented by:

- determinising it into a DFA $A' = (Q', \Sigma', \delta', I', F')$
- complementing it: $\overline{A'} = (Q', \Sigma', \delta', I', \overline{F'})$
- $|\overline{A'}| = |A'| = 2^{O(|A|)}$
Union of two NFAs

Definition: union of NFAs

Let $A_1 = (Q_1, \Sigma_1, \delta_1, I_1, F_1)$, $A_2 = (Q_2, \Sigma_2, \delta_2, I_2, F_2)$. Then $A = A_1 \cup A_2 = (Q, \Sigma, \delta, I, F)$ is defined as follows

- $Q := Q_1 \cup Q_2$, $I := I_1 \cup I_2$, $F := F_1 \cup F_2$
- $R(s, s') := \begin{cases} R_1(s, s') & \text{if } s \in Q_1 \\ R_2(s, s') & \text{if } s \in Q_2 \end{cases}$

Theorem

- $L(A) = L(A_1) \cup L(A_2)$
- $|A| = |A_1| + |A_2|$

Note

$A$ is an automaton which just runs nondeterministically either $A_1$ or $A_2$
Synchronous Product Construction

**Definition: product of NFAs**

Let $A_1 = (Q_1, \Sigma, \delta_1, I_1, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, I_2, F_2)$. Then, $A_1 \times A_2 = (Q, \Sigma, \delta, I, F)$ where

- $Q = Q_1 \times Q_2$,
- $I = I_1 \times I_2$,
- $F = F_1 \times F_2$,
- $<p, q> \xrightarrow{a} <p', q'>$ iff $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$.

**Theorem**

$L(A_1 \times A_2) = L(A_1) \cap L(A_2)$.

$|(A_1 \times A_2)| \leq |A_1| \cdot |A_2|$.
Example

- $A_1$ recognizes words with an even number of $b$'s.
- $A_2$ recognizes words with a number of $a$'s multiple of 3.
- The Product Automaton $A_1 \times A_2$ with $F = \{s_0, t_0\}$. 

Sebastiani and Tonetta
Regular Expressions

- Syntax: $\emptyset \mid \epsilon \mid a \mid reg_1.reg_2 \mid reg_1|reg_2 \mid reg^*$.
- Every regular expression $reg$ denotes a language $L(reg)$.
- Example: $a^*(b|bb).a^*$. The words with either 1 $b$ or 2 consecutive $b$'s.

**Theorem**

For every regular expression $reg$ we can construct a language equivalent NFA of size $O(|reg|)$.

**Theorem**

For every DFA $A$ we can construct a language equivalent regular expression $reg(A)$. 
Infinite Word Languages

Modeling infinite computations of reactive systems.

- An \( \omega \)-word \( \alpha \) over \( \Sigma \) is an infinite sequence
  \[ a_0, a_1, a_2, \ldots \]
  Formally, \( \alpha : \mathbb{N} \rightarrow \Sigma \).
  The set of all infinite words is denoted by \( \Sigma^\omega \).

- A \( \omega \)-language \( L \) is collection of \( \omega \)-words, i.e. \( L \subseteq \Sigma^\omega \).
  **Example** All words over \( \{a, b\} \) with infinitely many \( a \)’s.

**Notation:**
- **omega words** \( \alpha, \beta, \gamma \in \Sigma^\omega \).
- **omega-languages** \( L, L_1 \subseteq \Sigma^\omega \)

For \( u \in \Sigma^+ \), let \( u^\omega = u . u . u . \ldots \)
Omega-Automata

- We consider automaton running over infinite words.

Let $\alpha = aabbbb \ldots$
There are several possible runs.
Run $\rho_1 = s_1, s_1, s_1, s_1, s_2, s_2 \ldots$
Run $\rho_2 = s_1, s_1, s_1, s_1, s_1, s_1 \ldots$

- Acceptance Conditions: Büchi (Muller, Rabin, Street):
  Acceptance is based on states occurring infinitely often

- Notation Let $\rho \in Q^\omega$. Then,
  \[ \text{Inf}(\rho) = \{ s \in Q \mid \exists \infty i \in \mathbb{N}. \ \rho(i) = s \} \]
  (The set of states occurring infinitely many times in $\rho$.)
Büchi Automata

Nondeterministic Büchi Automaton

\[ A = (Q, \Sigma, \delta, I, F) \], where \( F \subseteq Q \) is the set of accepting states.

- A run \( \rho \) of \( A \) on omega word \( \alpha \) is an infinite sequence
  \[ \rho = q_0, q_1, q_2, \ldots \] s.t. \( q_0 \in I \) and \( q_i \xrightarrow{a_i} q_{i+1} \) for \( 0 \leq i \).

- The run \( \rho \) is accepting if
  \[ \text{Inf}(\rho) \cap F \neq \emptyset. \]

- The language accepted by \( A \)
  \[ \mathcal{L}(A) = \{ \alpha \in \Sigma^\omega \mid A \text{ has an accepting run on } \alpha \} \]
Büchi Automaton: Example

Let $\Sigma = \{a, b\}$.
Let a Deterministic Büchi Automaton (DBA) $A_1$ be

- With $F = \{s_1\}$ the automaton recognizes words with infinitely many $a$’s.
- With $F = \{s_2\}$ the automaton recognizes words with infinitely many $b$’s.
Let a Nondeterministic Büchi Automaton (NBA) $A_2$ be

![Diagram](image)

With $F = \{s_2\}$, the automaton $A_2$ recognizes words with finitely many $a$. Thus, $\mathcal{L}(A_2) = \overline{\mathcal{L}(A_1)}$. 
Deterministic vs. Nondeterministic Büchi Automata

Theorem

DBAs are strictly less powerful than NBAs.

The subset construction does not work:
let $DA_2$ be

$DA_2$ is not equivalent to $A_2$
(e.g., it recognizes $(b.a)^\omega$)
Closure Properties

**Theorem (union, intersection)**

For the NBAs $A_1, A_2$ we can construct

- the NBA $A$ s.t. $L(A) = L(A_1) \cup L(A_2)$. $|A| = |A_1| + |A_2|$

- the NBA $A$ s.t. $L(A) = L(A_1) \cap L(A_2)$. $|A| = |A_1| \cdot |A_2| \cdot 2$. 
Union of two NBAs

Definition: union of NBAs

Let $A_1 = (Q_1, \Sigma_1, \delta_1, I_1, F_1)$, $A_2 = (Q_2, \Sigma_2, \delta_2, I_2, F_2)$. Then $A = A_1 \cup A_2 = (Q, \Sigma, \delta, I, F)$ is defined as follows

- $Q := Q_1 \cup Q_2$, $I := I_1 \cup I_2$, $F := F_1 \cup F_2$
- $R(s, s') := \begin{cases} R_1(s, s') & \text{if } s \in Q_1 \\ R_2(s, s') & \text{if } s \in Q_2 \end{cases}$

Theorem

- $\mathcal{L}(A) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$
- $|A| = |A_1| + |A_2|$

Note

$A$ is an automaton which just runs nondeterministically either $A_1$ or $A_2$ (same construction as with ordinary automata)
Synchronous Product of NBAs

Definition: synchronous product of NBAs

Let $A_1 = (Q_1, \Sigma, \delta_1, l_1, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, l_2, F_2)$.
Then, $A_1 \times A_2 = (Q, \Sigma, \delta, l, F)$, where

$Q = Q_1 \times Q_2 \times \{1, 2\}$.

$l = l_1 \times l_2 \times \{1\}$.

$F = F_1 \times Q_2 \times \{1\}$.

$<p, q, 1> \xrightarrow{a} <p', q', 1>$ iff $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$ and $p \not\in F_1$.

$<p, q, 1> \xrightarrow{a} <p', q', 2>$ iff $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$ and $p \in F_1$.

$<p, q, 2> \xrightarrow{a} <p', q', 2>$ iff $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$ and $q \not\in F_2$.

$<p, q, 2> \xrightarrow{a} <p', q', 1>$ iff $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$ and $q \in F_2$.

Theorem

- $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$.

- $|A_1 \times A_2| \leq 2 \cdot |A_1| \cdot |A_2|$.
Product of NBAs: Intuition

- The automaton remembers two tracks, one for each source NBA, and it points to one of the two tracks.
- As soon as it goes through an accepting state of the current track, it switches to the other track.
- \( \Rightarrow \) in order to visit infinitely often a state in \( F \) (i.e., \( F_1 \)), it must visit infinitely often some state also in \( F_2 \).
- **Important subcase:** If \( F_2 = Q_2 \), then
  
  \[
  Q = Q_1 \times Q_2. \\
  I = I_1 \times I_2. \\
  F = F_1 \times Q_2. 
  \]
Product of NBAs: Example

---

Automata on Infinite Words
Closure Properties (2)

Theorem (complementation) [Safra, MacNaughten]

For the NBA $A_1$ we can construct an NBA $A_2$ such that
\[
\mathcal{L}(A_2) = \overline{\mathcal{L}(A_1)}.
\]

$|A_2| = O(2^{|A_1| \cdot \log(|A_1|)}).

Method: (hint)

(i) convert a Büchi automaton into a Non-Deterministic Rabin automaton

(ii) determinize and Complement the Rabin automaton

(iii) convert the Rabin automaton into a Büchi automaton.
Generalized Büchi Automaton

Definition

- A Generalized Büchi Automaton is a tuple $A := (Q, \Sigma, \delta, I, FT)$ where $FT = \langle F_1, F_2, \ldots, F_k \rangle$ with $F_i \subseteq Q$.
- A run $\rho$ of $A$ is accepting if $\text{Inf}(\rho) \cap F_i \neq \emptyset$ for each $1 \leq i \leq k$.

Theorem

For every Generalized Büchi Automaton $(A, FT)$ we can construct a language equivalent Büchi Automaton $(A', G')$.

Size: $|A'| \leq |A| \cdot k$.

Construction (Hint)

Let $Q' = Q \times \{1, \ldots, k\}$.

The automaton remains in phase $i$ till it visits a state in $F_i$. Then, it moves to $i + 1$ mode. After phase $k$ it moves to phase 1.
Degeneralizing a Büchi automaton: Example
Omega-regular Expressions

Definition
A language is called $\omega$-regular if it has the form $\bigcup_{i=1}^{n} U_i (V_i)^\omega$ where $U_i, V_i$ are regular languages.

Theorem
A language $L$ is $\omega$-regular iff it is NBA-recognizable.
NFA emptiness checking

- Equivalent of finding a final state reachable from an initial state.
- It can be solved with a DFS or a BFS.
- A DFS finds a counterexample on the fly (it is stored in the stack of the procedure).
- A BFS finds a final state reachable with a shortest counterexample, but it requires a further backward search to reproduce the path.
- Complexity: $O(n)$.
- Hereafter, assume w.l.o.g. that there is only one initial state.
NFA Emptiness Checking (cont.)

DFS(NFA A) {
    stack S=I;
    Hashtable T=I;
    while S!=∅ {
        v=top(S);
        if v∈F return NOT_EMPTY;
        if ∃w s.t. w∈δ(v) && T(w)==0 {
            hash(w,T);
            push(w,S);
        } else
            pop(S);
    }
    return EMPTY;
}
NBA emptiness checking

- Equivalent of finding an accepting cycle reachable from an initial state.
- A naive algorithm:
  1. a DFS finds the final states $f$ reachable from an initial state;
  2. for each $f$, a DFS finds if there exists a loop.
    - Complexity: $O(n^2)$.
- SCC-based algorithm:
  1. Tarjan’s algorithm uses a DFS to find the SCCs in linear time;
  2. another DFS finds if a non-trivial final SCC is reachable from an initial state.
    - Complexity: $O(n)$.
    - Drawbacks: it stores too much information and does not find directly a counterexample.
Double Nested DFS algorithm

- Double Nested DFS [Courcoubetis, Vardi, Wolper, Yannakakis, CAV’90]
  - two Hash tables:
    - $T_1$: reachable states
    - $T_2$: states reachable from a reachable final state
  - two stacks:
    - $S_1$: current branch of states reachable
    - $S_2$: current branch of states reachable from final state $f$
  - two nested DFS’s:
    - $\text{DFS}_1$: looks for a path from an initial state to a cycle starting from an accepting state
    - $\text{DFS}_2$: looks for a cycle starting from an accepting state
  - It stops as soon as it finds a counterexample.
  - The counterexample is given by the stack of $\text{DFS}_2$ (an accepting cycle) preceded by the stack of $\text{DFS}_1$ (a path from an initial state to the cycle).
Double Nested DFS - First DFS

DFS1(NBA A) {
    stack $S_1=I$; stack $S_2=\emptyset$;
    Hashtable $T_1=I$; Hashtable $T_2=\emptyset$;
    while $S_1\neq\emptyset$ {
        $v=$top($S_1$);
        if $\exists w$ s.t. $w \in \delta(v)$ && $T_1(w)==0$ {
            hash($w$,T1);
            push($w$,S1);
        } else {
            pop(S1);
            if $v \in F$ DFS2($v$,S2,T2,A);
        }
    }
    return EMPTY;
}

Remark: $T_2$ is not reset at each call of DFS2!
Double Nested DFS - Second DFS

DFS2(state f, stack S, Hashtable T, NBA A) {
    hash(f,T);
    push(f,S);
    while S!=∅ {
        v=top(S);
        if f∈δ(v) return NOT_EMPTY;
        if ∃w s.t. w∈δ(v) && T(w)==0 {
            hash(w);
            push(w);
        } else pop(S);
    }
}
Double nested DFS: intuition

DFS1 invokes DFS2 on each $f_1, ..., f_n$ only after popping it (postorder):

- DFS2 invoked on $f_j$ before than on $f_i$  
  $\Rightarrow$ $f_i$ not reachable from (any state $s$ which is reachable from) $f_j$
- If during $DFS2(f_i, ...)$ it is encountered a state $s$ which has already been explored by $DFS2(f_j, ...)$ for some $f_j$, then we conclude that we cannot reach $f_i$ from $s$.
  $\Rightarrow$ it is safe to backtrack.
Double Nested DFS: example
Let $M$ be a Kripke model and $\psi$ be an LTL formula

$M \models A\psi$ (CTL*)

$\iff M \models \psi$ (LTL)

$\iff \mathcal{L}(M) \subseteq \mathcal{L}(\psi)$

$\iff \mathcal{L}(M) \cap \mathcal{L}(\psi) = \{\}$

$\iff \mathcal{L}(A_M) \cap \mathcal{L}(A_{\neg \psi}) = \{\}$

$\iff \mathcal{L}(A_M \times A_{\neg \psi}) = \{\}$

$A_M$ is a Büchi Automaton equivalent to $M$ (which represents all and only the executions of $M$).

$A_{\neg \psi}$ is a Büchi Automaton which represents all and only the paths that satisfy $\neg \psi$ (do not satisfy $\psi$).

$A_M \times A_{\neg \psi}$ represents all and only the paths appearing in $M$ and not in $\psi$. 
Automata-Theoretic LTL M.C. (dual version)

Let $M$ be a Kripke model and $\varphi \overset{\text{def}}{=} \neg \psi$ be an LTL formula

$$M \models E\varphi$$
$$\iff M \not\models A\neg \varphi$$
$$\iff \ldots$$
$$\iff \mathcal{L}(A_M \times A_{\varphi}) \neq \{\}$$

- $A_M$ is a Büchi Automaton equivalent to $M$ (which represents all and only the executions of $M$)
- $A_{\varphi}$ is a Büchi Automaton which represents all and only the paths that satisfy $\varphi$

$A_M \times A_{\varphi}$ represents all and only the paths appearing in both $A_M$ and $A_{\varphi}$. 
Automata-Theoretic LTL Model Checking

Four steps:

(i) Compute $A_M$

(ii) Compute $A_\varphi$

(iii) Compute the product $A_M \times A_\varphi$

(iv) Check the emptiness of $\mathcal{L}(A_M \times A_\varphi)$
Computing an NBA $A_M$ from a Kripke Structure $M$

- Transform a Kripke structure $M = \langle S, S_0, R, L, AP \rangle$ into an NBA $A_M = \langle Q, \Sigma, \delta, I, F \rangle$ s.t.:
  - States: $Q := S \cup \{\text{init}\}$, init being a new initial state
  - Alphabet: $\Sigma := 2^{AP}$
  - Initial State: $I := \{\text{init}\}$
  - Accepting States: $F := Q = S \cup \{\text{init}\}$
  - Transitions:
    \[
    \delta : \quad q \xrightarrow{a} q' \quad \text{iff} \quad (q, q') \in R \quad \text{and} \quad L(q') = a
    \]
    
    $init \xrightarrow{a} q$ \quad \text{iff} \quad q \in S_0 \quad \text{and} \quad L(q) = a

- $\mathcal{L}(A_M) = \mathcal{L}(M)$
- $|A_M| = |M| + 1$
Computing a NBA $A_M$ from a Kripke Structure $M$: Example

Substantially, add one initial state, move labels from states to incoming edges, set all states as accepting states.
Note that the labels of a Büchi Automaton are different from the labels of a Kripke Structure. Also graphically, they are interpreted differently:

- in a Kripke Structure, it means that $p$ is true and all other propositions are false;
- in a Büchi Automaton, it means that $p$ is true and all other propositions are irrelevant ("don’t care"), i.e. they can be either true or false.
Computing a NBA $A_M$ from a Fair Kripke Structure $M$

Transforming a fair K.S. $M = \langle S, S_0, R, L, AP, FT \rangle$, $FT = \{F_1, ..., F_n\}$, into a generalized NBA $A_M = \langle Q, \Sigma, \delta, I, FT' \rangle$
s.t.:

- **States:** $Q := S \cup \{init\}$, $init$ being a new initial state
- **Alphabet:** $\Sigma := 2^{AP}$
- **Initial State:** $I := \{init\}$
- **Accepting States:** $FT' := FT$
- **Transitions:**
  \[
  \delta : \quad q \xrightarrow{a} q' \text{ iff } (q, q') \in R \text{ and } L(q') = a
  \]
  \[
  init \xrightarrow{a} q \text{ iff } q \in S_0 \text{ and } L(q) = a
  \]

- $\mathcal{L}(A_M) = \mathcal{L}(M)$
- $|A_M| = |M| + 1$
Computing a (Generalized) BA $A_M$ from a Fair Kripke Structure $M$: Example

- Substantially, add one initial state, move labels from states to incoming edges, set fair states as accepting states.
Translation problem

Problem
Given an LTL formula $\phi$, find a Büchi Automaton that accepts the same language of $\phi$.

- It is a fundamental problem in LTL model checking (in other words, every model checking algorithm that verifies the correctness of an LTL formula translates it in some sort of finite-state machine).
- We will translate an LTL formula into a Generalized Büchi Automata (GBA).
Exponential Translation

- From \( \varphi \), create a fair Kripke model, like in chapter 7.
- Convert it into a (Generalized) Büchi Automaton

Remark

Inefficient: up to \( 2^{\mathit{EL}(\varphi)} \) states.
- Kripke models require \textit{total} truth assignments to state variables
Example
Example
LTL Negative Normal Form (NNF)

- Every LTL formula $\varphi$ can be written into an equivalent formula $\varphi'$ using only the operators $\land$, $\lor$, $\mathbf{X}$, $\mathbf{U}$, $\mathbf{R}$ on propositional literals.

- Done by pushing negations down to literal level:
  
  $\neg(\varphi_1 \lor \varphi_2) \implies (\neg\varphi_1 \land \neg\varphi_2)$
  
  $\neg(\varphi_1 \land \varphi_2) \implies (\neg\varphi_1 \lor \neg\varphi_2)$
  
  $\neg\mathbf{X}\varphi_1 \implies \mathbf{X}\neg\varphi_1$
  
  $\neg(\varphi_1 \mathbf{U}\varphi_2) \implies (\neg\varphi_1 \mathbf{R}\neg\varphi_2)$
  
  $\neg(\varphi_1 \mathbf{R}\varphi_2) \implies (\neg\varphi_1 \mathbf{U}\neg\varphi_2)$

  $\implies$ the resulting formula is expressed in terms of $\lor$, $\land$, $\mathbf{X}$, $\mathbf{U}$, $\mathbf{R}$ and literals (Negative Normal Form, NNF).

- encoding linear if a DAG representation is used

- In the construction of $A_\varphi$ we now assume that $\varphi$ is in NNF.
On-the-fly Construction of $A_\varphi$ (Schema)

Apply recursively the following steps:

**Step 1**: Apply the tableau expansion rules to $\varphi$

$\psi_1 U \psi_2 \implies \psi_2 \lor (\psi_1 \land X(\psi_1 U \psi_2))$

$\psi_1 R \psi_2 \implies \psi_2 \land (\psi_1 \lor X(\psi_1 R \psi_2))$

until we get a Boolean combination of elementary subformulas of $\varphi$

(An elementary formula is a proposition or a $X$-formula.)
Tableaux rules: a quote

“After all... tomorrow is another day.”
[Scarlett O’Hara, “Gone with the Wind”]
On-the-fly Construction of $A_\varphi$ (Schema) [cont.]

Step 2: Convert all formulas into Disjunctive Normal Form:

$$\bigvee_i \left( \bigwedge_j l_{ij} \land \bigwedge_k \mathbf{X} \psi_{ik} \right)$$

- Each disjunct $\left( \bigwedge_j l_{ij} \land \bigwedge_k \mathbf{X} \psi_{ik} \right)$ represents a state:
  - The conjunction of literals $\bigwedge_j l_{ij}$ represents a set of labels in $\Sigma$ (e.g., if $\text{Vars}(\varphi) = \{p, q, r\}$, $p \land \neg q$ represents the two labels $\{p, \neg q, r\}$ and $\{p, \neg q, \neg r\}$)
  - $\bigwedge_k \mathbf{X} \psi_{ik}$ represents the next part of the state (obligations for the successors)

- N.B., if no next part occurs, $\mathbf{X} \top$ is implicitly assumed
Step 3: For every state represented by \((\bigwedge_j l_{ij} \land \bigwedge_k X \psi_{ik})\)

- draw an edge to all states which satisfy \(\bigwedge_k \psi_{ik}\)
- label the incoming edges with \(\bigwedge_j l_{ij}\)

N.B., if no next part occurs, \(X \top\) is implicitly assumed, so that an edge to a “true” node is drawn
Step 4: For every $\psi_i U \varphi_i$, for every state $q_j$, mark $q_j$ with $F_i$ iff $(\psi_i U \varphi_i) \notin q_j$ or $\varphi_i \in q_j$.

(If there is no U-subformulas, then mark all states with $F_1$—i.e., $FT \overset{\text{def}}{=} \{Q\}$.)
On-the-fly Construction of $A_{\phi}$ - State

- Henceforth, a state is represented by a tuple $s := \langle \lambda, \chi, \sigma \rangle$ where:
  - $\lambda$ is the set of labels
  - $\chi$ is the next part, i.e. the set of $X$-formulas satisfied by $s$
  - $\sigma$ is the set of the subformulas of $\phi$ satisfied by $s$ (necessary for the fairness definition)

- Given a set of LTL formulas $\Psi \overset{\text{def}}{=} \{ \psi_1, \ldots, \psi_k \}$, we define \( \text{Cover}(\Psi) \overset{\text{def}}{=} \text{Expand}(\Psi, \langle \emptyset, \emptyset, \emptyset \rangle) \) to be the set of initial states of the Buchi automaton representing $\bigwedge_j \psi_j$.
  - Combines steps 1. and 2. of previous slides
  - Expand() defined recursively as follows
On-the-fly Construction of $A_\phi$ - Expand

Given a set of formulas $\Phi$ to expand and a state $s$, we define the set of states $\text{Expand}(\Phi, s)$ recursively as follows:

- if $\Phi = \emptyset$, $\text{Expand}(\Phi, s) = \{ s \}$
- if $\bot \in \Phi$, $\text{Expand}(\Phi, s) = \emptyset$
- if $\top \in \Phi$ and $s = \langle \lambda, \chi, \sigma \rangle$,
  \[ \text{Expand}(\Phi, s) = \text{Expand}(\Phi \setminus \{ \top \}, \langle \lambda, \chi, \sigma \cup \{ \top \} \rangle) \]
- if $l \in \Phi$ and $s = \langle \lambda, \chi, \sigma \rangle$, $l$ propositional literal
  \[ \text{Expand}(\Phi, s) = \text{Expand}(\Phi \setminus \{ l \}, \langle \lambda \cup \{ l \}, \chi, \sigma \cup \{ l \} \rangle) \]
  (add $l$ to the labels of $s$ and to set of satisfied formulas)
- if $X \psi \in \Phi$ and $s = \langle \lambda, \chi, \sigma \rangle$,
  \[ \text{Expand}(\Phi, s) = \text{Expand}(\Phi \setminus \{ X \psi \}, \langle \lambda, \chi \cup \{ \psi \}, \sigma \cup \{X \psi\} \rangle) \]
  (add $\psi$ to the next part of $s$ and $X \psi$ to set of satisfied formulas)
- if $\psi_1 \land \psi_2 \in \Phi$ and $s = \langle \lambda, \chi, \sigma \rangle$,
  \[ \text{Expand}(\Phi, s) = \text{Expand}(\Phi \cup \{ \psi_1, \psi_2 \} \setminus \{ \psi_1 \land \psi_2 \}, \langle \lambda, \chi, \sigma \cup \{ \psi_1 \land \psi_2 \} \rangle) \]
  (process both $\psi_1$ and $\psi_2$ and add $\psi_1 \land \psi_2$ to $\sigma$)
On-the-fly Construction of $A_\phi$ - Expand

- if $\psi_1 \lor \psi_2 \in \Phi$ and $s = \langle \lambda, \chi, \sigma \rangle$,
  \[ \text{Expand}(\Phi, s) = \text{Expand}(\Phi \cup \{ \psi_1 \}\{ \psi_1 \lor \psi_2 \}, \langle \lambda, \chi, \sigma \cup \{ \psi_1 \lor \psi_2 \} \rangle) \]
  union
  \[ \text{Expand}(\Phi \cup \{ \psi_2 \}\{ \psi_1 \lor \psi_2 \}, \langle \lambda, \chi, \sigma \cup \{ \psi_1 \lor \psi_2 \} \rangle) \]
  (split $s$ in two copies, process $\psi_2$ on the first, $\psi_1$ on the second, add $\psi_1 \lor \psi_2$ to $\sigma$)

- if $\psi_1 \mathcal{U} \psi_2 \in \Phi$ and $s = \langle \lambda, \chi, \sigma \rangle$,
  \[ \text{Expand}(\Phi, s) = \text{Expand}(\Phi \cup \{ \psi_1 \}\{ \psi_1 \mathcal{U} \psi_2 \}, \langle \lambda, \chi \cup \{ \psi_1 \mathcal{U} \psi_2 \}, \sigma \cup \{ \psi_1 \mathcal{U} \psi_2 \} \rangle) \]
  union
  \[ \text{Expand}(\Phi \cup \{ \psi_2 \}\{ \psi_1 \mathcal{U} \psi_2 \}, \langle \lambda, \chi, \sigma \cup \{ \psi_1 \mathcal{U} \psi_2 \} \rangle) \]
  (split $s$ in two copies and process $\psi_1$ on the first, $\psi_2$ on the second, add $\psi_1 \mathcal{U} \psi_2$ to $\sigma$)

- if $\psi_1 \mathcal{R} \psi_2 \in \Phi$ and $s = \langle \lambda, \chi, \sigma \rangle$,
  \[ \text{Expand}(\Phi, s) = \text{Expand}(\Phi \cup \{ \psi_2 \}\{ \psi_1 \mathcal{R} \psi_2 \}, \langle \lambda, \chi \cup \{ \psi_1 \mathcal{R} \psi_2 \}, \sigma \cup \{ \psi_1 \mathcal{R} \psi_2 \} \rangle) \]
  union
  \[ \text{Expand}(\Phi \cup \{ \psi_1, \psi_2 \}\{ \psi_1 \mathcal{R} \psi_2 \}, \langle \lambda, \chi, \sigma \cup \{ \psi_1 \mathcal{R} \psi_2 \} \rangle) \]
  (split $s$ in two copies and process $\psi_1$ on the first, $\psi_2$ on the second, add $\psi_1 \mathcal{R} \psi_2$ to $\sigma$)
On-the-fly Construction of $A_\phi$ - Expand

Two relevant subcases: $F\psi \overset{\text{def}}{=} \top U \psi$ and $G\psi \overset{\text{def}}{=} \bot R \psi$

- if $F\psi \in \Phi$ and $s = \langle \lambda, \chi, \sigma \rangle$,
  \[
  \text{Expand}(\Phi, s) = \text{Expand}(\Phi \setminus \{F\psi\}, \langle \lambda, \chi \cup \{F\psi\}, \sigma \cup \{F\psi\} \rangle) \
  \cup \text{Expand}(\Phi \cup \{\psi\} \setminus \{F\psi\}, \langle \lambda, \chi, \sigma \cup \{F\psi\} \rangle)
  \]

- if $G\psi \in \Phi$ and $s = \langle \lambda, \chi, \sigma \rangle$,
  \[
  \text{Expand}(\Phi, s) = \text{Expand}(\Phi \cup \{\psi\} \setminus \{G\psi\}, \langle \lambda, \chi \cup \{G\psi\}, \sigma \cup \{G\psi\} \rangle)
  \]
  Note: $\text{Expand}(\Phi \cup \{\bot, \psi\} \setminus \{G\psi\}, ...) = \emptyset$
Definition of $A_\phi$

Given a set of LTL formulas $\Psi$, we define

$$Cover(\Psi) \overset{\text{def}}{=} Expand(\Psi, \langle\emptyset, \emptyset, \emptyset\rangle).$$

For an LTL formula $\phi$, we construct a Generalized NBA $A_\phi = (Q, Q_0, \Sigma, L, T, FT)$ as follows:

- $\Sigma = 2^{\text{vars}(\phi)}$
- $Q$ is the smallest set such that
  - $Cover(\{\phi\}) \subseteq Q$
  - if $\lambda, \chi, \sigma \in Q$, then $Cover(\chi) \in Q$
- $Q_0 = Cover(\{\phi\})$.
- $L(\lambda, \chi, \sigma) = \{a \in \Sigma | a \models \lambda\}$
- $(s, s') \in T$ iff, $s = \lambda, \chi, \sigma$ and $s' \in Cover(\chi)$
- $FT = \langle F_1, F_2, \ldots, F_k \rangle$ where, for all $(\psi_i \mathbf{U} \phi_i)$ occurring positively in $\phi$, $F_i = \{\lambda, \chi, \sigma \in Q | (\psi_i \mathbf{U} \phi_i) \notin \sigma \text{ or } \phi_i \in \sigma\}$. (If there is no U-subformulas, then $FT \overset{\text{def}}{=} \{Q\}$).
Example: $\phi = \text{FG}p$

$$\text{Cover}(\{\text{FG}p\}) = \text{Expand}(\{\text{FG}p\}, \langle\emptyset, \emptyset, \emptyset\rangle)$$

$$= \text{Expand}(\emptyset, \langle\emptyset, \{\text{FG}p\}, \{\text{FG}p\}\rangle) \cup \text{Expand}(\{\text{G}p\}, \langle\emptyset, \emptyset, \{\text{FG}p\}\rangle)$$

$$= \{\langle\emptyset, \{\text{FG}p\}, \{\text{FG}p\}\rangle\} \cup \text{Expand}(\{p\}, \langle\{p\}, \{\text{G}p\}, \{\text{FG}p, \text{G}p\}\rangle)$$

$$= \{\langle\emptyset, \{\text{FG}p\}, \{\text{FG}p\}\rangle\} \cup \text{Expand}(\emptyset, \langle\{p\}, \{\text{G}p\}, \{\text{FG}p, \text{G}p, p\}\rangle)$$

$$= \{\langle\emptyset, \{\text{FG}p\}, \{\text{FG}p\}\rangle, \langle\{p\}, \{\text{G}p\}, \{\text{FG}p, \text{G}p, p\}\rangle\}$$

$$\text{Cover}(\{\text{G}p\}) = \text{Expand}(\{\text{G}p\}, \langle\emptyset, \emptyset, \emptyset\rangle)$$

$$= \text{Expand}(\{p\}, \langle\emptyset, \{\text{G}p\}, \{\text{G}p\}\rangle)$$

$$= \text{Expand}(\emptyset, \langle\{p\}, \{\text{G}p\}, \{\text{G}p, p\}\rangle)$$

$$= \{\langle\{p\}, \{\text{G}p\}, \{\text{G}p, p\}\rangle\}$$

**Optimization:**

merge $\langle\{p\}, \{\text{G}p\}, \{\text{FG}p, \text{G}p, p\}\rangle$ and $\langle\{p\}, \{\text{G}p\}, \{\text{G}p, p\}\rangle$
Example: $\phi = \text{FG}\ p$

- Call $s_1 = \langle \emptyset, \{\text{FG}\ p\}, \{\text{FG}\ p\} \rangle$, $s_2 = \langle \{p\}, \{\text{G}\ p\}, \{\text{FG}\ p, \text{G}\ p, p\} \rangle$
- $Q = \{s_1, s_2\}$
- $Q_0 = \{s_1, s_2\}$.
- $T: \ s_1 \rightarrow \{s_1, s_2\}$,
  $s_2 \rightarrow \{s_2\}$
- $FT = \langle F_1 \rangle$ where $F_1 = \{s_2\}$. 

[ XG ]
Example: $\phi = pUq$

\[
\text{Cover}([pUq]) = \text{Expand}([pUq], \langle \emptyset, \emptyset, \emptyset \rangle) \cup \text{Expand}([q], \langle \emptyset, \emptyset, \{pUq\} \rangle) \\
= \text{Expand}(\emptyset, \langle \{p\}, \{pUq\}, \{pUq, p\} \rangle) \cup \text{Expand}(\emptyset, \langle \{q\}, \emptyset, \{pUq, q\} \rangle) \\
= \{\langle \{p\}, \{pUq\}, \{pUq, p\} \rangle \cup \{\langle \{q\}, \top, \{pUq, q\} \rangle \}
\]

\[
\text{Cover}([\top]) = \{\langle \emptyset, \{\top\}, \{\top\} \rangle \}
\]
Example: $\phi = pUq$

- Let $s_1 = \langle \{p\}, \{pUq\}, \{pUq, p\} \rangle$, $s_2 = \langle \{q\}, \{\top\}, \{pUq, q\} \rangle$, $s_3 = \langle \emptyset, \{\top\}, \{\top\} \rangle$.
- $Q = \{s_1, s_2, s_3\}$,
- $Q_0 = \{s_1, s_2\}$,
- $T: s_1 \rightarrow \{s_1, s_2\}$,
  $s_2 \rightarrow \{s_3\}$
  $s_3 \rightarrow \{s_3\}$
- $FT = \langle F_1 \rangle$ where $F_1 = \{s_2, s_3\}$. 

\[ X(pUq) \]
Example: \( \phi = \text{GF}p \)

\[
\text{Cover}\{\text{GF}p\}
= E(\{\text{GF}p\}, \langle\emptyset, \emptyset, \emptyset\rangle)
= E(\{p\}, \langle\{\}, \{\text{GF}p\}, \{\text{GF}p\}\rangle)
= E(\{\}, \langle\emptyset, \{\text{GF}p, Fp\}, \{\text{GF}p, Fp\}\rangle) \cup E(\{p\}, \langle\{\}, \{\text{GF}p\}, \{\text{GF}p, Fp\}\rangle)
= E(\{\}, \langle\emptyset, \{\text{GF}p, Fp\}, \{\text{GF}p, Fp\}\rangle) \cup E(\{\}, \langle\{p\}, \{\text{GF}p\}, \{\text{GF}p, Fp, p\}\rangle)
= \{\langle\emptyset, \{\text{GF}p, Fp\}, \{\text{GF}p, Fp\}\rangle\} \cup \{\langle\{p\}, \{\text{GF}p\}, \{\text{GF}p, Fp, p\}\rangle\}
\]

Note: \( \text{GF}p \land Fp \iff \text{GF}p \), s.t. \( \text{Cover}(\text{GF}p \land Fp) = \text{Cover}(\text{GF}p) \)
Example: $G F \rho$

- Let $s_1 = \{p\}, \forall \{G F \rho\}, \forall \{G F \rho, F \rho, p\}$,
  $s_2 = \emptyset, \forall \{G F \rho, F \rho\}, \forall \{G F \rho, F \rho\}$,
- $Q = \{s_1, s_2\}$,
- $Q_0 = \{s_1, s_2\}$,
- $T: s_1 \rightarrow \{s_1, s_2\}$,
  $s_2 \rightarrow \{s_1, s_2\}$
- $FT = \langle F_1 \rangle$ where $F_1 = \{s_1\}$. 

\[ XG F \rho \]

\[ XG F \rho \]

\[ XG F \rho \]

\[ XG F \rho \]
Four steps:

(i) Compute $A_M$: $|A_M| = O(|M|)$

(ii) Compute $A_{\varphi}$: $|A_{\varphi}| = O(2^{|\varphi|})$

(iii) Compute the product $A_M \times A_{\varphi}$:

$|A_M \times A_{\varphi}| = |A_M| \cdot |A_{\varphi}| = O(|M| \cdot 2^{|\varphi|})$

(iv) Check the emptiness of $L(A_M \times A_{\varphi})$: $O(|A_M \times A_{\varphi}|) = O(|M| \cdot 2^{|\varphi|})$

$\implies$ the complexity of LTL M.C. grows linearly wrt. the size of the model $M$ and exponentially wrt. the size of the property $\varphi$
Final Remarks

- Büchi automata are in general more expressive than LTL!
  - Some tools (e.g., Spin, ObjectGEODE) allow specifications to be expressed directly as NBAs
  - Complementation of NBA important!

- For every LTL formula, there are many possible equivalent NBAs
  - Lots of research for finding “the best” conversion algorithm

- Performing the product and checking emptiness very relevant
  - Lots of techniques developed (e.g., partial order reduction)
  - Lots on ongoing research
Given the following two Büchi automata (doubly-circled states represent accepting states, $a$, $b$ are labels):

Write the product Büchi automaton $BA1 \times BA2$. 
Ex: Product of Büchi automata

[ Solution: The product is:

track 1

\[
\begin{array}{l}
\text{s1t1} \\
\text{s2t1} \\
\text{s1t2} \\
\text{s2t2} \\
\end{array}
\]

\[
\begin{array}{l}
\text{a} \\
\text{b} \\
\text{a} \\
\text{a} \\
\end{array}
\]

\[
\begin{array}{l}
\text{track 2} \\
\text{s1t1} \\
\text{s2t1} \\
\text{s1t2} \\
\text{s2t2} \\
\end{array}
\]

\[
\begin{array}{l}
\text{a} \\
\text{a} \\
\text{b} \\
\text{b} \\
\end{array}
\]

]
Ex: De-generalization of Büchi Automata

Given the following generalized Büchi automaton $A \overset{\text{def}}{=} \langle Q, \Sigma, \delta, I, FT \rangle$, with two sets of accepting states $FT \overset{\text{def}}{=} \{ F_1, F_2 \}$ s.t. $F_1 \overset{\text{def}}{=} \{ s_2 \}$, $F_2 \overset{\text{def}}{=} \{ s_1 \}$:

convert it into an equivalent plain Büchi automaton.
Ex: De-generalization of Büchi Automata

[ Solution: The result is:

\[ \begin{array}{ccc}
 s_{11} & \rightarrow & a & \rightarrow & a & \rightarrow & s_{12} \\
 & \downarrow & b & \downarrow & b & \downarrow & b \\
 s_{21} & \rightarrow & a & \rightarrow & a & \rightarrow & s_{22}
\end{array} \]
Ex: From Kripke models to Büchi automata

Given the following fair Kripke model $M$, convert it into an equivalent Büchi automaton.

[ Solution: ]
Consider the LTL formula $\varphi \overset{\text{def}}{=} (G\neg p) \rightarrow (p \mathsf{U} q)$.

(a) rewrite $\varphi$ into Negative Normal Form

[ Solution: $(G\neg p) \rightarrow (p \mathsf{U} q) \implies (\neg G\neg p) \lor (p \mathsf{U} q) \implies (Fp) \lor (p \mathsf{U} q)$ ]

(b) find the initial states of a corresponding Büchi automaton (for each state, define the labels of the incoming arcs and the “next” section.)

[ Solution: Applying tableaux rules we obtain: $p \lor \mathsf{XF} p \lor q \lor (p \land \mathsf{X}(p \mathsf{U} q))$, which is already in disjunctive normal form. This correspond to the following four initial states: ]

- $p$ with label $\top$
- $q$ with label $\top$
- $p$ with label $\mathsf{U} p$
- $p$ with label $\mathsf{U} q$
Given the following Büchi automaton BA (doubly-circled states represent accepting states):

Say which of the following sentences are true and which are false.

(a) BA accepts all and only the paths verifying $\mathbf{GF}q$. [Solution: false]

(b) BA accepts all and only the paths verifying $\mathbf{FG}q$. [Solution: true]

(c) BA accepts only paths verifying $\mathbf{F}q$, but not all of them. [Solution: true]

(d) BA accepts all the paths verifying $\mathbf{F}q$, but not only them. [Solution: false]