Introduction to Formal Methods
Chapter 08: Automata-theoretic LTL Model Checking

Roberto Sebastiani and Stefano Tonetta

DISI, Università di Trento, Italy – rseba@disi.unitn.it
URL: http://disi.unitn.it/~rseba/DIDATTICA/fm2014/
Teaching assistant: Alessandra Giordani – agiordani@disi.unitn.it

CDLM in Informatica, academic year 2013-2014

last update: May 2, 2014

Copyright notice: some material (text, figures) displayed in these slides is courtesy of M. Benerecetti, A. Cimatti, P. Pandya, M. Pistore, M. Roveri, and S. Tonetta, who detain its copyright. Some examples displayed in these slides are taken from [Clarke, Grunberg & Peled, “Model Checking”, MIT Press], and their copyright is detained by the authors. All the other material is copyrighted by Roberto Sebastiani. Every commercial use of this material is strictly forbidden by the copyright laws without the authorization of the authors. No copy of these slides can be displayed in public without containing this copyright notice.
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.

![Automaton Diagram]

---

Sebastiani and Tonetta () Ch. 08: Automata-theoretic LTL Model Checki May 2, 2014 6 / 92
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.

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

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.:
- \(\delta : Q \times \Sigma \to Q\) 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 $\mathcal{L}(A) = \{ u \in \Sigma^* \mid A \text{ has an accepting run on } u \}$
- The languages accepted by a NFA are called regular languages.
Finite State Automata: examples

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

```
  \* \* \*
  a  b
  \* \* \*
```

Recognizes words which do not end in $b$.

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

```
  a, b
  \* \* \*
  \* b
  \* \* \*
```

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 $\{s_1, ..., s_j\}$ of states in $A$ ($Q' \subseteq 2^Q$), with the intended meaning that:
  - $A'$ is in the state $\{s_1, ..., s_j\}$ if $A$ is in one of the states $s_1, ..., s_j$.

- The deterministic transition relation $\delta' : 2^Q \times \Sigma \rightarrow 2^Q$ is:
  - $\{s\} \xrightarrow{a} \{s_i \mid s \xrightarrow{a} s_i\}$
  - $\{s_1, ..., s_j, ..., s_n\} \xrightarrow{a} \bigcup_{j=1}^{n} \{s_i \mid s_j \xrightarrow{a} s_i\}$

- The (unique) initial state is $I' =_{\text{def}} \{s_i \mid s_i \in I\}$

- The set of final states $F'$ is such that:
  $\{s_1, ..., s_n\} \in F'$ iff $s_i \in F$ for some $i \in \{1, ..., n\}$
Determinisation [cont.]

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

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

There are NFAs of size $n$ for which the size of the minimum sized DFA must have size $O(2^n)$. 
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: $\overline{A'} = (Q', \Sigma', \delta', I', \overline{F'})$
- $|\overline{A'}| = |A'| = 2^{O(|A|)}$
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:

\[
Q := Q_1 \cup Q_2, \quad I := I_1 \cup I_2, \quad F := F_1 \cup F_2
\]

\[
R(s, s') := \begin{cases} 
R_1(s, s') & \text{if } s \in Q_1 \\
R_2(s, s') & \text{if } s \in Q_2
\end{cases}
\]

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 \( A_2 \).
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

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

$|(A_1 \times A_2)| \leq |A_1| \cdot |A_2|$.
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 $F = \{s_0, t_0\}$.
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.

\[ \alpha = aabb \ldots \]

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

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

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

- The run \( \rho \) is accepting if
  \[ \inf(\rho) \cap F \neq \emptyset. \]

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

\[ F = \{s_2\} \]

Thus, $L(A_2) = L(A_1)$. 

With $F = \{s_2\}$, the automaton $A_2$ recognizes words with finitely many $a$. Thus, $L(A_2) = L(A_1)$. 
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 $(b.a)^\omega$)

There is no DBA equivalent to $A_2$. 

![Diagram of automata]
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)
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

\begin{align*}
Q &= Q_1 \times Q_2 \times \{1, 2\}.
\end{align*}

\begin{align*}
l &= l_1 \times l_2 \times \{1\}.
F &= F_1 \times Q_2 \times \{1\}.
\end{align*}

\begin{align*}
<p, q, 1> &\xrightarrow{a} <p', q', 1> \text{ iff } p \xrightarrow{a} p' \text{ and } q \xrightarrow{a} q' \text{ and } p \notin F_1.
\end{align*}

\begin{align*}
<p, q, 1> &\xrightarrow{a} <p', q', 2> \text{ iff } p \xrightarrow{a} p' \text{ and } q \xrightarrow{a} q' \text{ and } p \in F_1.
\end{align*}

\begin{align*}
<p, q, 2> &\xrightarrow{a} <p', q', 2> \text{ iff } p \xrightarrow{a} p' \text{ and } q \xrightarrow{a} q' \text{ and } q \notin F_2.
\end{align*}

\begin{align*}
<p, q, 2> &\xrightarrow{a} <p', q', 1> \text{ iff } p \xrightarrow{a} p' \text{ and } q \xrightarrow{a} q' \text{ and } q \in F_2.
\end{align*}

Theorem

\begin{itemize}
  \item $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$.
  \item $|A_1 \times A_2| \leq 2 \cdot |A_1| \cdot |A_2|$.
\end{itemize}
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.
  \[ \implies \text{in order to visit infinitely often a state in } F \text{ (i.e., } F_1\text{), it must visit infinitely often some state also in } F_2 \]
- 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
Closure Properties (2)

Theorem (complementation) [Safra, MacNaughten]
For the NBA $A_1$ we can construct an NBA $A_2$ such that
\[ \mathcal{L}(A_2) = \overline{\mathcal{L}(A_1)}. \]
\[ |A_2| = O(2^{\epsilon_1} \cdot \log(|A_1|)). \]

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 \( \bigcup_{i=1}^{n} U_i \cdot (V_i)^{\omega} \) 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:
    - T1: reachable states
    - T2: states reachable from a reachable final state
  - two stacks:
    - S1: current branch of states reachable
    - S2: current branch of states reachable from final state f
  - two nested DFS’s:
    - DFS1 looks for a path from an initial state to a cycle starting from an accepting state
    - DFS2 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 DFS2 (an accepting cycle) preceded by the stack of DFS1 (a path from an initial state to the cycle).
Double Nested DFS - First DFS

DFS1(NBA A) {
    stack S1=I; stack S2=∅;
    Hashtable T1=I; Hashtable T2=∅;
    while S1!=∅ {
        v=top(S1);
        if ∃w s.t. w∈δ(v) && T1(w)==0 {
            hash(w,T1);
            push(w,S1);
        } else {
            pop(S1);
            if v∈F DFS2(v,S2,T2,A);
        }
    }
    return EMPTY;
}

Remark: T2 is not reset at each call of DFS2 !
Double Nested DFS - Second DFS

DFS2\((state \; f, \; stack \; S, \; Hashtable \; T, \; NBA \; A) \) \{ \\
\quad hash(f,T);
\quad push(f,S);
\quad while \; S\neq\emptyset \; \{ \\
\quad \quad v=\text{top}(S);
\quad \quad if \; f \in \delta(v) \; \text{return NOT.EMPTY};
\quad \quad if \; \exists w \; s.t. \; w \in \delta(v) \; \&\& \; T(w)==0 \; \{ \\
\quad \quad \quad hash(w);
\quad \quad \quad push(w);
\quad \quad \} \; \text{else pop}(S);
\quad \}
\}
Double nested DFS: intuition

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

- **DFS2** invoked on $f_j$ before than on $f_i$  
  $\implies f_i$ not reachable from (any state $s$ which is reachable from) $f_j$

- If during **DFS2**($f_i, \ldots$) it is encountered a state $s$ which has already been explored by **DFS2**($f_j, \ldots$) for some $f_j$, then we conclude that we cannot reach $f_i$ from $s$.

$\implies$ it is safe to backtrack.
Double Nested DFS: example
Let $M$ be a Kripke model and $\psi$ be an LTL formula

\[ M \models A\psi \quad \text{(CTL*)} \]

\[ \iff \quad M \models \psi \quad \text{(LTL)} \]

\[ \iff \quad \mathcal{L}(M) \subseteq \mathcal{L}(\psi) \]

\[ \iff \quad \mathcal{L}(M) \cap \mathcal{L}(\psi) = \{\} \]

\[ \iff \quad \mathcal{L}(A_M) \cap \mathcal{L}(A_{\neg \psi}) = \{\} \]

\[ \iff \quad \mathcal{L}(A_M \times A_{\neg \psi}) = \{\} \]

- $A_M$ 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 $\psi$)

\[ \implies A_M \times A_{\neg \psi} \text{ represents all and only the paths appearing in } M \text{ and not in } \psi. \]
Let $M$ be a Kripke model and $\varphi \overset{\text{def}}{=} \neg \psi$ be an LTL formula

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

- $A_M$ 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 $\varphi$

$\Rightarrow A_M \times A_{\varphi}$ 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: $Q := S \cup \{init\}$, $init$ being a new initial state
  - Alphabet: $\Sigma := 2^{AP}$
  - Initial State: $I := \{init\}$
  - Accepting States: $F := Q = S \cup \{init\}$
  - Transitions:

$$\delta : q \xrightarrow{a} q' \text{ iff } (q, q') \in R \text{ and } L(q') = a$$

$$init \xrightarrow{a} q \text{ iff } q \in S_0 \text{ and } L(q) = a$$

- $\mathcal{L}(A_M) = \mathcal{L}(M)$
- $|A_M| = |M| + 1$
Computing a 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. $M = \langle S, S_0, R, L, AP, FT \rangle$, $FT = \{ F_1, \ldots, F_n \}$, into a generalized NBA $A_M = \langle Q, \Sigma, \delta, I, FT' \rangle$ s.t.:

- **States:** $Q := S \cup \{ \text{init} \}$, $\text{init}$ being a new initial state
- **Alphabet:** $\Sigma := 2^{AP}$
- **Initial State:** $I := \{ \text{init} \}$
- **Accepting States:** $FT' := FT$
- **Transitions:**

  $\delta : q \xrightarrow{a} q'$ iff $(q, q') \in R$ and $L(q') = a$

  $\text{init} \xrightarrow{a} q$ iff $q \in S_0$ 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

$\{p,q\}$ $\{p,q\}$ $\{p,\neg q\}$ $\{\neg p,q\}$

$\{p\}$ $\{q\}$ $\{p,\neg q\}$ $\{p,q\}$

Fair Kripke Structure

Generalized Büchi Automaton

$\Rightarrow$ 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 $2^{EL(\varphi)}$ states.
- Kripke models require 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 $\land$, $\lor$, $X$, $U$, $R$ on propositional literals.

- Done by pushing negations down to literal level:
  
  \[
  \neg(\varphi_1 \lor \varphi_2) \implies (\neg \varphi_1 \land \neg \varphi_2) \\
  \neg(\varphi_1 \land \varphi_2) \implies (\neg \varphi_1 \lor \neg \varphi_2) \\
  \neg X \varphi_1 \implies X \neg \varphi_1 \\
  \neg(\varphi_1 U \varphi_2) \implies (\neg \varphi_1 R \neg \varphi_2) \\
  \neg(\varphi_1 R \varphi_2) \implies (\neg \varphi_1 U \neg \varphi_2)
  \]

  $\implies$ the resulting formula is expressed in terms of $\lor$, $\land$, $X$, $U$, $R$ 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.
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:

$$\bigvee_{i,j} \left( \bigwedge_{j} l_{ij} \land \bigwedge_{k} X\psi_{ik} \right)$$

- Each disjunct $\left( \bigwedge_{j} l_{ij} \land \bigwedge_{k} X\psi_{ik} \right)$ 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 $\{p, \neg q, \neg r\}$)
  - $\bigwedge_{k} X\psi_{ik}$ represents the next part of the state (obligations for the successors)

- N.B., if no next part occurs, $X\top$ 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
On-the-fly Construction of $A_\varphi$ (Schema) [cont.]

**Step 4**: For every $\psi_i U \varphi_i$, for every state $q_j$, mark $q_j$ with $F_i$ iff

$(\psi_i U \varphi_i) \not\in q_j \text{ or } \varphi_i \in q_j$

(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 $\mathcal{X}$-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
- $\text{Expand}()$ 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 $s = \langle \lambda, \chi, \sigma \rangle$,
  \[ \text{Expand}(\Phi, s) = \text{Expand}(\Phi \cup \{\psi_1\}\{\psi_1 \lor \psi_2\}, \langle \lambda, \chi, \sigma \cup \{\psi_1 \lor \psi_2\} \rangle) \]
  \[ \cup \text{Expand}(\Phi \cup \{\psi_2\}\{\psi_1 \lor \psi_2\}, \langle \lambda, \chi, \sigma \cup \{\psi_1 \lor \psi_2\} \rangle) \]

(split $s$ in two copies, process $\psi_1$ on the first, $\psi_2$ on the second, add $\psi_1 \lor \psi_2$ to $\sigma$)

- If $\psi_1 U \psi_2 \in \Phi$ and $s = \langle \lambda, \chi, \sigma \rangle$,
  \[ \text{Expand}(\Phi, s) = \text{Expand}(\Phi \cup \{\psi_1\}\{\psi_1 U \psi_2\}, \langle \lambda, \chi \cup \{\psi_1 U \psi_2\}, \sigma \cup \{\psi_1 U \psi_2\} \rangle) \]
  \[ \cup \text{Expand}(\Phi \cup \{\psi_2\}\{\psi_1 U \psi_2\}, \langle \lambda, \chi, \sigma \cup \{\psi_1 U \psi_2\} \rangle) \]

(split $s$ in two copies and process $\psi_1$ on the first, $\psi_2$ on the second, add $\psi_1 U \psi_2$ to $\sigma$)

- If $\psi_1 R \psi_2 \in \Phi$ and $s = \langle \lambda, \chi, \sigma \rangle$,
  \[ \text{Expand}(\Phi, s) = \text{Expand}(\Phi \cup \{\psi_2\}\{\psi_1 R \psi_2\}, \langle \lambda, \chi \cup \{\psi_1 R \psi_2\}, \sigma \cup \{\psi_1 R \psi_2\} \rangle) \]
  \[ \cup \text{Expand}(\Phi \cup \{\psi_1, \psi_2\}\{\psi_1 R \psi_2\}, \langle \lambda, \chi, \sigma \cup \{\psi_1 R \psi_2\} \rangle) \]

(split $s$ in two copies and process $\psi_1$ on the first, $\psi_2$ on the second, add $\psi_1 R \psi_2$ to $\sigma$)
On-the-fly Construction of $A_\phi$ - Expand

Two relevant subcases: $F\psi \stackrel{\text{def}}{=} \top \cup \psi$ and $G\psi \stackrel{\text{def}}{=} \bot \cap R\psi$

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

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

- $\text{Cover}\{\phi\} = \text{Expand}(\{\phi\}, \langle\emptyset, \emptyset, \emptyset\rangle)$
  $= \text{Expand}(\emptyset, \langle\emptyset, \{\phi\}, \{\phi\}\rangle) \cup \text{Expand}(\{Gp\}, \langle\emptyset, \emptyset, \{\phi\}\rangle)$
  $= \{\langle\emptyset, \{\phi\}, \{\phi\}\rangle\} \cup \text{Expand}(\{p\}, \langle\emptyset, \{Gp\}, \{\phi, Gp\}\rangle)$
  $= \{\langle\emptyset, \{\phi\}, \{\phi\}\rangle\} \cup \text{Expand}(\emptyset, \langle\{p\}, \{Gp\}, \{\phi, Gp, p\}\rangle)$
  $= \{\langle\emptyset, \{\phi\}, \{\phi\}\rangle, \langle\{p\}, \{Gp\}, \{\phi, Gp, p\}\rangle\}$

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

- Optimization:
  merge $\langle\{p\}, \{Gp\}, \{\phi, Gp, p\}\rangle$ and $\langle\{p\}, \{Gp\}, \{Gp, p\}\rangle$
Example: $\phi = FGp$

- Call $s_1 = \langle \emptyset, \{\phi\}, \{\phi\} \rangle$, $s_2 = \langle \{p\}, \{Gp\}, \{\phi, Gp, p\} \rangle$
- $Q = \{s_1, s_2\}$
- $Q_0 = \{s_1, s_2\}$.
- $T: s_1 \rightarrow \{s_1, s_2\}$,
  $s_2 \rightarrow \{s_2\}$
- $FT = \langle F_1 \rangle$ where $F_1 = \{s_2\}$. 
Example: $\phi = pUq$

$$\text{Cover} \{pUq\}$$

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

$$= \text{Expand}(\{p\}, \langle \emptyset, \{pUq\}, \{pUq\} \rangle) \cup \text{Expand}(\{q\}, \langle \emptyset, \emptyset, \{pUq\} \rangle)$$

$$= \text{Expand}(\emptyset, \langle \{p\}, \{pUq\}, \{pUq, p\} \rangle) \cup \text{Expand}(\emptyset, \langle \{q\}, \emptyset, \{pUq, q\} \rangle)$$

$$= \{\langle \{p\}, \{pUq\}, \{pUq, p\} \rangle \} \cup \{\langle \{q\}, \top, \{pUq, q\} \rangle \}$$

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

- Let $s_1 = \text{def } \langle \{p\}, \{pUq\}, \{pUq, p\} \rangle$, $s_2 = \text{def } \langle \{q\}, \{\top\}, \{pUq, q\} \rangle$, $s_3 = \text{def } \langle \emptyset, \{\top\}, \{\top\} \rangle$.
- $Q = \{s_1, s_2, s_3\}$,
- $Q_0 = \{s_1, s_2\}$,
- $T : \begin{array}{l}
s_1 \to \{s_1, s_2\}, \\
s_2 \to \{s_3\} \\
s_3 \to \{s_3\} \\
\end{array}$
- $FT = \langle F_1 \rangle$ where $F_1 = \{s_2, s_3\}$.
Example: $\phi = \text{GF}p$

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

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

- Let $s_1 = \{p\}, \{GFp\}, \{GFp, Fp, p\}$,
- $s_2 = \emptyset, \{GFp, Fp\}, \{GFp, Fp\}$,
- $Q = \{s_1, s_2\},$
- $Q_0 = \{s_1, s_2\},$
- $T : s_1 \rightarrow \{s_1, s_2\},$
- $s_2 \rightarrow \{s_1, s_2\}$
- $FT = \langle F_1 \rangle$ where $F_1 = \{s_1\}$. 

![Diagram of B"uchi Automata](image-url)
Automata-Theoretic LTL Model Checking: complexity

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
Ex: Product of Büchi automata

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:

\[
\begin{array}{cccc}
& s_1 t_1 & s_1 t_2 & s_1 t_1 & s_1 t_2 \\
\text{track 1} & a & b & a & b \\
& s_2 t_1 & s_2 t_2 & s_2 t_1 & s_2 t_2 \\
\text{track 2} & b & a & b & a
\end{array}
\]
Ex: De-generalization of Büchi Automata

Given the following generalized Büchi automaton $A \triangleq \langle Q, \Sigma, \delta, I, FT \rangle$, with two sets of accepting states $FT \triangleq \{F1, F2\}$ s.t. $F1 \triangleq \{s2\}$, $F2 \triangleq \{s1\}$:

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

[ Solution: The result is:

\[
\begin{align*}
&\text{s11} \\
&\text{s21} \\
&\text{s12} \\
&\text{s22}
\end{align*}
\]

\[
\begin{align*}
&\text{a} \\
&\text{b} \\
&\text{b} \\
&\text{a}
\end{align*}
\]
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 $\varphi \overset{\text{def}}{=} (G \neg p) \rightarrow (p U q)$.

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

[ Solution: $(G \neg p) \rightarrow (p U q) \implies (\neg G \neg p) \vee (p U q) \implies (F p) \vee (p U q)$ ]

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

[ Solution: Applying tableaux rules we obtain: $p \vee XFp \vee q \vee (p \wedge X(p U q))$, which is already in disjunctive normal form. This correspond to the following four initial states: ]

1. $p$ with label $\top$
2. $q$ with label $\top$
3. $p$ with label $[p U q]$
4. $p$ with label $[F p]$
Ex: Büchi automaton

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 ]