Introduction to Formal Methods
08 - Automata-Theoretic LTL Model Checking

Roberto Sebastiani - rseba@disi.unitn.it
Stefano Tonetta - tonettas@fbk.eu

A.A. 2008-2009
Last update: February 14, 2009

Content

1 Automata-Theoretic 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
   - Exponential construction of Büchi Automata
   - On-the-fly construction of Büchi Automata
   - Complexity

System's computations

- The behaviors (computations) of a system can be seen as sequences of 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.

```
@done
```

or

```
@done @done
```
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
  \[ \Rightarrow \] 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.

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

To solve the language containment problem, we need to know:
- how to complement an automaton,
- how to intersect two automata,
- how to check the language emptiness of an automaton.

Content

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

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

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

\[ a, b \in \Sigma \text{ finite alphabet.} \]
\[ u, v, w \in \Sigma^* \text{ finite words.} \]
\[ \epsilon \text{ empty word.} \]
\[ u \cdot v \text{ catenation.} \]
\[ u^i = u \cdot u \cdot \ldots \cdot u \text{ repeated } i \text{-times.} \]
\[ U, V \subseteq \Sigma^* \text{ Finite word languages.} \]

FSA Definition

Nondeterministic Finite State Automaton (NFA):

NFA is \((Q, \Sigma, \delta, I, F)\)
- \(Q\) 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 \((q, a, q') \in \delta\).

Deterministic Finite State Automaton (DFA):

DFA has \(\delta : Q \times \Sigma \to Q\), 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. \(q_0 \in I \text{ and } q_i \xrightarrow{a_i} q_{i+1}\) for \(0 \leq i < n\).
- An accepting run is one where the last state \(q_n \in F\).
- The language accepted by \(A\)
  \(L(A) = \{u \in \Sigma^* \mid A \text{ has an accepting run on } u\}\)
- The languages accepted by a NFA are called regular languages.
Finite State Automata

Example: DFA $A_1$ over $\Sigma = \{a, b\}$.
Recognizes words which do not end in $b$.

NFA $A_2$. Recognizes words which end in $b$.

Determinisation

Theorem (determinisation) Given a NFA $A$ we can construct a DFA $A'$ s.t.
$L(A) = L(A')$. Size $|A'| = 2^{O(|A|)}$.

Determinisation [cont.]

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

$A_2$ 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:
- determinizing it into a DFA $A' = (Q', \Sigma', \delta', I', F')$
- complementing it: $\overline{A'} = (Q', \Sigma', \delta', I', F')$
- $|A'| = 2^{O(|A|)}$

Union of two NFAs

Two NFAs $A_1 = (Q_1, \Sigma_1, \delta_1, I_1, F_1)$, $A_2 = (Q_2, \Sigma_2, \delta_2, I_2, F_2)$, $A = A_1 \cup A_2 = (Q, \Sigma, \delta, I, F)$ 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 $L(A_1 \times A_2) = L(A_1) \cap L(A_2)$.

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: $\emptyset \mid \epsilon \mid a \mid \text{reg}_1 . \text{reg}_2 \mid \text{reg}_1 | \text{reg}_2 \mid \text{reg}^*.$

Every regular expression $\text{reg}$ denotes a language $L(\text{reg}).$

Example: $a^* (b|bb).a^*.$ The words with either 1 $b$ or 2 consecutive $b$’s.

Theorem: For every regular expression $\text{reg}$ we can construct a language equivalent NFA of size $O(|\text{reg}|).$

Theorem: For every DFA $A$ we can construct a language equivalent regular expression $\text{reg}(A).$

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, $\alpha : \mathbb{N} \to \Sigma.$
  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 $\alpha = aabb \ldots$ 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,

$$\inf(\rho) = \{ s \in Q \mid \exists^\infty i \in \mathbb{N}. \ \rho(i) = s \}.$$ 

(The set of states occurring infinitely many times in $\rho$.)
Büchi Automata

Nondeterministic Büchi Automaton
A = (Q, Σ, δ, I, F), where F ⊆ Q is the set of accepting states.

- A run ρ of A on omega word α is an infinite sequence
  ρ = q₀, q₁, q₂, ... s.t. q₀ ∈ I and qᵢ ↦ qᵢ₊₁ for 0 ≤ i.
- The run ρ is accepting if
  Inf(ρ) ∩ F ≠ ∅.
- The language accepted by A
  \( L(A) = \{α ∈ Σ^ω \mid A \text{ has an accepting run on } α \} \).

Büchi Automaton: Example

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

\begin{align*}
& A_1 = (Q_1, Σ, δ_1, I_1, F_1) \nonumber \\
& \text{With } F_1 = \{s_1\} \text{ the automaton recognizes words with infinitely many } a \text{'s.} \\
& \text{With } F_2 = \{s_2\} \text{ the automaton recognizes words with infinitely many } b \text{'s.}
\end{align*}

Büchi Automaton: Example (2)

Let a Nondeterministic Büchi Automaton (NBA) \( A_2 \) be

\begin{align*}
& A_2 = (Q_2, Σ, δ_2, I_2, F_2) \nonumber \\
& \text{With } F_2 = \{s_2\} \text{, automaton } A_2 \text{ recognizes words with finitely many } a \text{'s.} \\
& \text{Thus, } L(A_2) = L(A_1).
\end{align*}

Deterministic vs. Nondeterministic Büchi Automata

Theorem DBAs are strictly less powerful than NBAs.

The subset construction does not work: let \( DA_2 \) be

\begin{align*}
& DA_2 = (Q_A, Σ, δ_A, I_A, F_A) \nonumber \\
& \text{\( DA_2 \) is not equivalent to } A_2 \text{ (e.g., it recognizes } (b.a)^ω) \\
& \text{There is no DBA equivalent to } A_2
\end{align*}
Closure Properties

**Theorem (union, intersection)**

For the NBAs $A_1, A_2$, we can construct
- the NBA $A$ s.t. $\mathcal{L}(A) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$. $|A| = |A_1| + |A_2|$.
- the NBA $A$ s.t. $\mathcal{L}(A) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$. $|A| = |A_1| \cdot |A_2| \cdot 2$.

**Union of two NBAs**

Two NBAs $A_1 = (Q_1, \Sigma_1, \delta_1, I_1, F_1)$, $A_2 = (Q_2, \Sigma_2, \delta_2, I_2, F_2)$, $A = A_1 \cup A_2 = (Q, \Sigma, \delta, I, F)$ is defined as follows
- $Q := Q_1 \cup Q_2$, $I := I_1 \cup I_2$, $F := F_1 \cup F_2$
- $R(s, s') := \begin{cases} R_1(s, s') & \text{if } s \in Q_1 \\ R_2(s, s') & \text{if } s \in Q_2 \end{cases}$

$A$ is an automaton which just runs nondeterministically either $A_1$ or $A_2$.
- $|A| = |A_1| + |A_2|$
- (same construction as with ordinary automata)

**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.
- To visit infinitely often a state in $F$, it must visit infinitely often some state also in $F_2$.
- Important subcase: If $F_2 = Q_2$, then
  - $Q = Q_1 \times Q_2$.
  - $I = I_1 \times I_2$.
  - $F = F_1 \times Q_2$.

Synchronous Product of NBAs

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
- $Q = Q_1 \times Q_2 \times \{1, 2\}$.
- $I = I_1 \times I_2 \times \{1\}$.
- $F = F_1 \times Q_2 \times \{1\}$.

$\delta(p, \mu, q) = \begin{cases} \delta_1(p, \mu, q') & \text{if } p \xrightarrow{a} p' \text{ and } q \xrightarrow{a} q' \text{ and } p \notin F_1.
, \text{ and } q \notin F_1.
, \text{ and } q \notin F_2. \end{cases}$

$\delta(p, \mu, q) = \begin{cases} \delta_1(p, \mu, q') & \text{if } p \xrightarrow{a} p' \text{ and } q \xrightarrow{a} q' \text{ and } p \notin F_1.
, \text{ and } q \notin F_1.
, \text{ and } q \notin F_2. \end{cases}$

Theorem $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$.
Closure Properties (2)

Theorem (complementation)
For the NBA $A_1$ we can construct an NBA $A_2$ such that $L(A_2) = \overline{L(A_1)}$.
$|A_2| = O(2^{|A_1| \log(|A_1|)}).

Method: (hint)
1. convert a Büchi automaton into a Non-Deterministic Rabin automaton.
2. determinize and complement the Rabin automaton
3. convert the Rabin automaton into a Büchi automaton

Omega Regular Expressions

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.

Content

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

- The Automata-Theoretic Approach to Model Checking
  - Automata-Theoretic LTL Model Checking
  - From Kripke Structures to Büchi Automata
  - From LTL Formulas to Büchi Automata
  - Exponential construction of Büchi Automata
  - On-the-fly construction of Büchi Automata
  - Complexity
**Nonemptiness of NFA Automata**

- The **nonemptiness** problem for an automaton is to decide whether there is at least one word for which there is an accepting run.
- For NFA (i.e., standard nondeterministic finite automata), nonemptiness algorithms are based on reachability.
- In Datalog/Prolog notation:
  
  $$\text{nonempty} \leftarrow \text{initial}(X), \text{cn}(X, Y), \text{final}(Y).$$
  
  $$\text{cn}(X, Y) \leftarrow r(X, A, Y).$$
  
  $$\text{cn}(X, Y) \leftarrow r(X, A, Z), \text{cn}(Z, Y).$$
  
  where \(\text{initial}(X)\) denotes that \(X\) is an initial state; \(\text{final}(X)\) denotes that \(X\) is a final state; \(r(X, A, Y)\) denotes that a transition from \(X\) to \(Y\) reading \(A\); and \(\text{cn}(\cdot, \cdot)\) is the transitive closure of \(r(X, A, Y)\) projected on \(X, Y\).
  
  Notice that \(\text{cn}(\cdot, \cdot)\) is not expressible in FOL.
- Reachability is a well-known problem on graphs, its complexity is \(\text{NLOGSPACE}\)-complete.
  
  Thm. **Nonemptiness for NFA a is \text{NLOGSPACE}-complete.**
  
  Practical algorithms have a linear cost.

**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: \(O(n)\).
- Henceafter, assume w.l.o.g. that there is only one initial state.

**NBA emptiness checking**

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

**Nonemptiness of Büchi Automata**

- For Büchi automata, nonemptiness algorithms are based on fair reachability.
- In Datalog/Prolog notation:
  
  $$\text{nonempty} \leftarrow \text{initial}(X), \text{cn}(X, Y), \text{final}(Y), \text{cn}(Y, Y).$$
  
  $$\text{cn}(X, Y) \leftarrow r(X, A, Y).$$
  
  $$\text{cn}(X, Y) \leftarrow r(X, A, Z), \text{cn}(Z, Y).$$
  
  where, as before, \(\text{initial}(X)\) denotes that \(X\) is an initial state; \(\text{final}(X)\) denotes that \(X\) is a final state; \(r(X, A, Y)\) denotes that a transition from \(X\) to \(Y\) reading \(A\); and \(\text{cn}(\cdot, \cdot)\) is the transitive closure of \(r(X, A, Y)\) projected on \(X, Y\).
- Fair reachability amounts to two separate reachability problems: (1) reach a final state from the initial state, (2) from that final state reach itself through a loop.
- Fair reachability has the same complexity as reachability: \(\text{NLOGSPACE}\)-complete.
  
  Thm. **Nonemptiness for Büchi automata is \text{NLOGSPACE}-complete.**
  
  Practical algorithms have a linear cost.
Automata-Theoretic LTL Model Checking

- $M \models A\neg \psi$ (CTL*)
- $\iff M \models \psi$ (LTL)
- $\iff \mathcal{L}(M) \subseteq \mathcal{L}(\psi)$
- $\iff \mathcal{L}(M) \cap \mathcal{L}(\neg \psi) = \{\}$
- $\iff \mathcal{L}(A_M) \cap \mathcal{L}(A_{\neg \psi}) = \{\}$
- $\iff \mathcal{L}(A_M \times A_{\neg \psi}) = \{\}$

- $A_M$ 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)

- $M \models E\phi$
- $\iff M \not\models A\neg \phi$
- $\iff ...$
- $\iff \mathcal{L}(A_M \times A_{\phi}) \neq \{\}$

- $A_M$ is a Büchi Automaton equivalent to $M$ (which represents all and only the executions of $M$)
- $A_{\phi}$ is a Büchi Automaton which represents all and only the paths that satisfy $\phi$
- $A_M \times A_{\phi}$ represents all and only the paths appearing in both $A_M$ and $A_{\phi}$.
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.

Labels on Kripke Structures and BA’s - Remark

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 uncertain (they can be either true or false).
Language Containment
Automata on Finite Words
Automata on Infinite Words
Emptiness Checking

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

Translation 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 LTL in a (equivalent) variant of Büchi Automata called Labeled Generalized Büchi Automata (LGBA).

Translation from LTL to Büchi Automaton: examples

- $\Diamond P$
  $\mathcal{L} = \text{true}^* P \text{true}^*$

- $Q \cup P$
  $\mathcal{L} = Q^* P \text{true}^*$

- $\Box P$
  $\mathcal{L} = P^*$

- $Q \cup \Box P$
  $\mathcal{L} = Q^* \text{true} \text{true}^*$

Translation from LTL to Büchi Automaton: examples

- $\Box (P \rightarrow \Diamond Q)$
  $\mathcal{L} = (\text{not } P^* P \text{true } Q \text{true})^* \cup (\text{not } P^* P \text{true } Q \text{true})^* \text{not } P^*$

- $\Box \Diamond P$
  $\mathcal{L} = (\text{true}^* P)^*$

- $\Diamond \Box P$
  $\mathcal{L} = \text{true}^* P^*$
Content

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
   - Exponential construction of Büchi Automata
   - On-the-fly construction of Büchi Automata
   - Complexity

Automata-Theoretic LTL Model Checking: complexity

Four steps:
1. Compute $A_M$: $|A_M| = O(|M|)$
2. Compute $A_\varphi$:
3. Compute the product $A_M \times A_\varphi$:
4. Check the emptiness of $L(A_M \times A_\varphi)$:
Automata-Theoretic LTL Model Checking: complexity

Four steps:
1. Compute $A_M$: $|A_M| = O(|M|)$
2. Compute $A_\phi$: $|A_\phi| = O(2^{|\phi|})$
3. Compute the product $A_M \times A_\phi$: $|A_M \times A_\phi| = |A_M| \cdot |A_\phi| = O(|M| \cdot 2^{|\phi|})$
4. Check the emptiness of $L(A_M \times A_\phi)$:

$\Rightarrow$ the complexity of LTL M.C. grows linearly wrt. the size of the model $M$ and exponentially wrt. the size of the property $\phi$

Final Remarks

- Büchi automata are in general more expressive than LTL!
  $\Rightarrow$ Some tools (e.g., Spin, ObjectGEODE) allow specifications to be expressed directly as NBAs
  $\Rightarrow$ complementation of NBA important!
- for every LTL formula, there are many possible equivalent NBAs
  $\Rightarrow$ lots of research for finding “the best” conversion algorithm
- performing the product and checking emptiness very relevant
  $\Rightarrow$ lots of techniques developed (e.g., partial order reduction)
  $\Rightarrow$ lots of ongoing research