Automata-Theoretic LTL Model-Checking

Marsha Chechik

University of Toronto

Outline

- Automata-Theoretic Model-Checking
  - Automata on finite and infinite words
  - Representing models and formulas
  - Acceptance Conditions
  - Model checking using automata
  - Partial order reduction and closure under stuttering
- Implementing automata-theoretic model checking
  - Checking emptiness
  - Nested DFS
  - Bitstate hashing
- SPIN/Promela
  - expressing models in Promela
  - using SPIN
Automata on Finite Words

Finite automaton $\mathcal{A}$ over finite words is a tuple $(\Sigma, Q, \Delta, Q^0, F)$ where

- $\Sigma$ is a finite alphabet
- $Q$ is a finite set of states
- $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation
- $Q^0 \subseteq Q$ is a set of initial states
- $F \subseteq Q$ is a set of final states

$\Sigma = \{a, b, c\}, \, Q = \{q_0, q_1\}, \, Q^0 = \{q_0\}, \, F = \{q_1\}.$

Automata on Finite Words, Cont’d

Let $v$ be a word of $\Sigma^*$ of length $|v|$. A run of $\mathcal{A}$ over $v$ is a mapping $\rho : \{0, 1, ..., |v|\} \rightarrow Q$ s.t.

- First state is the initial state: $\rho(0) \in Q^0$
- $\forall 0 \leq i \leq |v| \cdot (\rho(i), v(i), \rho(i + 1)) \in \Delta$

A run $\rho$ of $\mathcal{A}$ on $v$ – a path in automaton to a state $\rho(|v|)$ where the edges are labeled with letters in $v$ (so $v$ is input to $\mathcal{A}$).

A run is accepting if $\rho(|v|) \in F$. An automaton $\mathcal{A}$ accepts a word $v$ iff exists an accepting run of $\mathcal{A}$ on $v$.

Run $aacb$ is accepting.
The language $L(A) \subseteq \Sigma^*$ is all words accepted by $A$.

$$\epsilon + a(a + c)^*b(b + c)^*.$$ This is a regular expression.

- Languages represented by regular expressions (and recognizable by finite automata on finite strings) are regular languages.
- An automaton is deterministic if
  $$\forall a \cdot (q, a, q') \in \Delta \land (q, a, q'') \in \Delta \implies q' = q''.$$ Otherwise, it is non-deterministic.
- Every non-deterministic automaton on finite words can be translated into an equivalent deterministic automaton (which accepts the same language).

**Automata on Infinite Words**

- Reactive programs execute forever – so we want infinite sequences of states.
- Answer: finite automata over infinite words.
- Simplest case: Buchi automata
  - Same structure as automata on finite words
  - ... but different notion of acceptance
  - Recognize words from $\Sigma^\omega$
    - $\Sigma = \{a, b\}$  \( v = abaabaaab... \)
    - $\Sigma = \{a, b, c\}$  \( L_1 \subseteq \Sigma^\omega \) is \( v \in L_1 \) iff after any occurrence of letter $a$ there is some occurrence of letter $b$ in $v$.
    - Possible strings:
      - $ababab...$  \( aaabaaab... \)
      - $abbabbabb...$  \( accbaccb... \)
Automata on Infinite Words (Cont’d)

\[
\begin{align*}
\text{Accepting language: } & ( (b + c)^\omega a(a + c)^* b)^\omega \text{ (\(\omega\)-regular expression)} \\
F & \text{ – the set of accepting states} \\
\text{A run of a Buchi automaton } & \mathcal{A} \text{ over an infinite word } v \in \Sigma^\omega. \text{ Domain of run – the set of all natural numbers.} \\
\text{inf}(\rho) & \text{ – set of states that appear infinitely often in the run } \rho. \text{ A run } \rho \text{ is accepting (Buchi accepting) iff } \text{inf}(\rho) \cap F \neq \emptyset. \\
\text{Language expressible by } \omega\text{-regular expressions (and thus recognizable by some Buchi automaton) is } \omega\text{-regular or Buchi-recognizable.}
\end{align*}
\]

Operations on Buchi Automata

\[
\begin{align*}
\text{Buchi-recognizable languages are closed under complementation.} \\
i.e., \text{ from a Buchi automaton } \mathcal{A} \text{ recognizing } \mathcal{L} \text{ one can construct an automaton recognizing } \Sigma^\omega – \mathcal{L}. \\
The number of states in this automaton is } O(2^{Q \log Q}), \text{ where } Q \text{ – states in } \mathcal{A} \text{ (Safra’s construction)} \\
\text{Easy to do this for deterministic Buchi automata:} \\
\text{Unfortunately, not all non-deterministic Buchi automata can be made deterministic!}
\end{align*}
\]
Complementation Algorithm for DA

Create two copies of an automaton:

- \( A_1 \): Take non-accepting states of \( A \) and make them accepting.
- \( A_2 \): Every transition to non-accepting state gets duplicated to same state in \( A_1 \).

Operations on Buchi Automata, Cont’d

Buchi automata are closed under intersection [Chouka74]:

- given two Buchi automata \( B_1 = (\Sigma, Q_1, \Delta_1, Q_0^1, Q_1) \) (all states are accepting) and \( B_2 = (\Sigma, Q_2, \Delta_2, Q_0^2, F_2) \),
  construct \( B_1 \cap B_2 = (\Sigma, Q_1 \times Q_2, \Delta', Q_0^1 \times Q_0^2, Q_1 \times F_2) \),
  where
- \( ((r_i, q_j), a, (r_m, q_n)) \in \Delta' \) iff \( (r_i, a, r_m) \in \Delta_1 \) and \( (q_j, a, q_n) \in \Delta_2 \).
Intersection of Arbitrary Buchi automata

- Main point: determining accepting states: need to go through accepting states of $B_1$ and $B_2$ infinite number of times

- 3 copies of the automaton:
  - 1st copy: start and accept here
  - 2nd copy: move when accepting state from $B_1$ has been seen
  - 3rd copy: move when accepting state from $B_2$ has been seen

Operations, Cont’d

- The emptiness problem for Buchi automata is decidable
  - $\mathcal{L}(A) \neq \emptyset$
  - logspace-complete for NLOGSPACE, i.e., solvable in linear time [Vardi, Wolper]) – see later in the lecture.

- Nonuniversality problem for Buchi automata is decidable
  - $\mathcal{L}(A) \neq \sum^\omega$
  - logspace-complete for PSPACE [Sisla, Vardi, Wolper]
Infinite Occurrences

- $\exists^\omega i \cdot Y(i)$ – there exists infinitely many $i$th such that $Y(i)$
- For $\rho \in Q^\omega$
  - $In(\rho)$ is the set of states that occur infinitely often
  - $In(\rho) = \{q \in Q \mid \exists^\omega i \cdot \rho(i) = q\}$
- Büchi condition
  - $\mathcal{F}$ is $F \subseteq Q$
  - $In(w) \cap F \neq \emptyset$
  - weak fairness – something occurs infinitely often
- Muller condition
  - $\mathcal{F}$ is $\{F_1, \ldots, F_n\} \subseteq 2^Q$
  - $\exists i \cdot In(w) = F_i$

Acceptance Conditions

- Rabin condition ("pairs")
  - $\mathcal{F}$ is $\{(R_1, G_1), \ldots, (R_n, G_n)\}$ with $R_i, G_i \subseteq Q$
  - $\exists i \cdot In(w) \cap R_i = \emptyset \land In(w) \cap G_i \neq \emptyset$
  - Rabin $(\emptyset, F)$ is equivalent to Büchi $F$
- Street condition ("complemented pairs")
  - $\mathcal{F}$ is $\{(F_1, E_1), \ldots, (F_n, E_n)\}$ with $E_i, F_i \subseteq Q$
  - $\forall i \cdot In(w) \cap F_i \neq \emptyset \Rightarrow In(w) \cap E_i \neq \emptyset$
  - strong fairness
    - if infinitely often enabled, then infinitely often executed
    - Street $(Q, F)$ is equivalent to Büchi $F$
Acceptance Conditions

- Parity condition
  - $\mathcal{F}$ is $F_1 \subseteq \cdots \subseteq F_n$ with $F_i \subseteq Q$
  - smallest $i$ for which $In(w) \cap F_i \neq \emptyset$ is even

- co-Büchi condition
  - $\mathcal{F}$ is $F \subseteq Q$
  - accepts $w$ if $In(w) \cap F = \emptyset$

Nondeterministic Büchi-, Muller-, Rabin-, and Street-automata all recognize the same $\omega$-languages

Example: Acceptance

Language over $\{a, b, c\}^\omega$
- if $a$ occurs infinitely often, then so does $b$

Automaton with states $q_a$, $q_b$, and $q_c$, and $\delta$

<table>
<thead>
<tr>
<th>state</th>
<th>$\delta(q, a)$</th>
<th>$\delta(q, b)$</th>
<th>$\delta(q, c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_a$</td>
<td>$q_a$</td>
<td>$q_b$</td>
<td>$q_c$</td>
</tr>
<tr>
<td>$q_b$</td>
<td>$q_a$</td>
<td>$q_b$</td>
<td>$q_c$</td>
</tr>
<tr>
<td>$q_c$</td>
<td>$q_a$</td>
<td>$q_b$</td>
<td>$q_c$</td>
</tr>
</tbody>
</table>

Acceptance conditions
- Street – single pair ($\{q_a\}, \{q_b\}$)
- Muller – all states $F$ where $q_a \in F \Rightarrow q_b \in F$
  - $\{q_b\}, \{q_c\}, \{q_b, q_c\}, \{q_a, q_b\}, \{q_a, q_b, q_c\}$
Example: Acceptance

Automaton with states $q_a$, $q_b$, and $q_c$, and $\delta$

<table>
<thead>
<tr>
<th>state</th>
<th>$\delta(q, a)$</th>
<th>$\delta(q, b)$</th>
<th>$\delta(q, c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_a$</td>
<td>$q_a$</td>
<td>$q_b$</td>
<td>$q_c$</td>
</tr>
<tr>
<td>$q_b$</td>
<td>$q_a$</td>
<td>$q_b$</td>
<td>$q_c$</td>
</tr>
<tr>
<td>$q_c$</td>
<td>$q_a$</td>
<td>$q_b$</td>
<td>$q_c$</td>
</tr>
</tbody>
</table>

Acceptance conditions

- **Rabin**
  - either $b$ occurs infinitely often, or both $a$ and $b$ have finite occurrences
  - two pairs $(\emptyset, \{q_b\}), (\{q_a, q_b\}, \{q_c\})$

- **Parity**
  - $\emptyset, \{q_b\}, \{q_a, q_b\}, \{q_a, q_b, q_c\}$

For Büchi acceptance condition simulate Rabin pairs by nondeterminism

<table>
<thead>
<tr>
<th>state</th>
<th>$\delta(q, a)$</th>
<th>$\delta(q, b)$</th>
<th>$\delta(q, c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_a$</td>
<td>$q_a$</td>
<td>$q_b$</td>
<td>${q_c, q'}$</td>
</tr>
<tr>
<td>$q_b$</td>
<td>$q_a$</td>
<td>$q_b$</td>
<td>${q_c, q'}$</td>
</tr>
<tr>
<td>$q_c$</td>
<td>$q_a$</td>
<td>$q_b$</td>
<td>${q_c, q'}$</td>
</tr>
<tr>
<td>$q'$</td>
<td>$q_b$</td>
<td>$q_b$</td>
<td>$q'$</td>
</tr>
</tbody>
</table>

Every time $c$ occurs, guess that a suffix containing only $c$ is reached

- Büchi acceptance condition
  - $F = \{q_b, q'\}$
Modeling Systems Using Automata

A system is a set of all its executions. So, every state is accepting!

Transform Kripke structure \((S, R, S_0, L)\)

where \(L : S \rightarrow s^{AP}\)

...into automaton \(A = (\Sigma, S \cup \{\ell\}, \Delta, \{\ell\}, S \cup \{\ell\})\),

where \(\Sigma = 2^{AP}\)

\((s, \alpha, s') \in \Delta \text{ for } s, s' \in S \text{ iff } (s, s') \in R \text{ and } \alpha = L(s')\)

\((\ell, \alpha, s') \in \Delta \text{ iff } s \in S_0 \text{ and } \alpha = L(s)\)

$LTL$ and Buchi Automata

Specification – also in the form of an automaton!

Buchi automata can encode all LTL properties.

Examples:

\(a \ U \ b\)

Other examples:

- \(\Box \Diamond p\)
- \(\Box \Diamond (p \lor q)\)
- \(\neg \Box \Diamond (p \lor q)\)
- \(\neg (\Box (p \ U \ q))\)
LTL to Buchi Automata

Theorem [Wolper, Vardi, Sisla 83]: Given an LTL formula $\phi$, one can build a Buchi automaton $S = (\Sigma, Q, \Delta, Q_0, F)$ where

- $\Sigma = 2^{\text{Prop}}$
- the number of automatic propositions, variables, etc. in $\phi$
- $|Q| \leq 2^{O(|\phi|)}$
- $|\phi|$ - length of the formula

... s.t. $L(S)$ is exactly the set of computations satisfying the formula $\phi$.

Algorithm given in Section 9.4

But Buchi automata are more expressive than LTL!

Sketch of the Algorithm

- Compute the set of subformulas that must hold in each reachable state and in each of its successor states.
- Convert formula into normal form (negation for atomic propositions)
- Create initial state, marked with the formula to be matched and a dummy incoming edge
- Recursively
  - take a subformula that remains to be satisfied
  - look at the leading temporal operator: may split the current state into two, each annotated with appropriate subformula
- Make connections to accepting state
Automata-theoretic Model Checking

- The system $A$ satisfies the specification $S$ when
  - $L(A) \subseteq L(S)$
  - ... each behavior of the system is among the allowed behaviours
- Alternatively,
  - let $L(S)$ be the language $\Sigma^\omega - L(S)$. Then, we are looking for
    - $L(A) \cap L(S) = \emptyset$
    - no behavior of $A$ is disallowed by $S$.
- If the intersection is not empty, any behavior in it corresponds to a counterexample.
- Counterexample is always of the form $uv^\omega$, where $u$ and $v$ are finite words.

Complexity

- Checking whether a formula $\phi$ is satisfied by a finite-state model $K$ can be done in time $O(||K|| \times 2^{O(|\phi|)})$ or in space $O((\log ||K|| + ||\phi||)^2)$.
- i.e., checking is polynomial in the size of the model and exponential in the size of the specification.
Partial-order Reduction

Example:

Interleaving:
Run sequences:
1. $x=1$, $g=g+2$, $y=1$, $g=g*2$
2. $x=1$, $y=1$, $g=g+2$, $g=g*2$
3. $x=1$, $y=1$, $g=g*2$, $g=g+2$
4. $y=1$, $g=g*2$, $x=1$, $g=g*2$
5. $y=1$, $x=1$, $g=g*2$, $g=g+2$
6. $y=1$, $x=1$, $g=g+2$, $g=g*2$

Dependent operations
- $g=g*2$, $g=g+2$ (same data object)
- $x=1$, $g=g+2$ (part of T1)
- $y=1$, $g=g*2$ (part of T2)

Independent operations
- $x=1$, $y=1$
- $x=1$, $g=g*2$
- $y=1$, $g=g+2$

1 and 2 – differ only in relative order of $y=1$ and $g=g+2$ which are independent
4 and 5 – only relative order of $x=1$, $g=g+2$ which are independent

Only 2 distinct runs:

2. $x = 1$, $y = 1$, $g = g+2$, $g = g*2$
3. $x = 1$, $y = 1$, $g = g*2$, $g = g+2
Partial-order Reduction

- Two equivalence classes: [1, 2, 6], [3, 4, 5]

For verification, it is sufficient to consider just one run from each equivalence class...

as long as the formulas are closed under stuttering!

Closure Under Stuttering

- **Stuttering** refers to a sequence of identically labeled states along a path in a Kripke structure.

- Intuitively, an LTL formula is *closed under stuttering* if the interpretation of the formula remains the same under state sequences that differ only by repeated states [Abadi, Lamport'01].

  Assume $F$ is closed under stuttering. Then,
  - $\square F$ is closed under stuttering
  - $\diamond F$ is not closed under stuttering
Closure under stuttering and $\text{LTL}_{-X}$

$LTL_{-X}$ – a subset of LTL without the $\circ$ operator.

- Theorem: All $LTL_{-X}$ formulas are closed under stuttering [Lamport’94]

- Theorem: All cus LTL properties can be expressed in $\text{LTL}_{-X}$
  - By exponentially increasing the size of the formula!

- Determining whether an arbitrary LTL formula is closed under stuttering is PSPACE-complete [Peled, Wilke, Wolper’96]

- Exists an algorithm based on *edges* (changes in values of variables) that allows to use full LTL and yet guarantee closure under stuttering [Paun, Chechik’01]

- Observation: stuttering does not add or delete edges or change their relative order

- Theorem [Paun99]: If $A$ and $B$ are cus then so is $\Diamond (\neg A \land \circ A \land \circ B)$