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

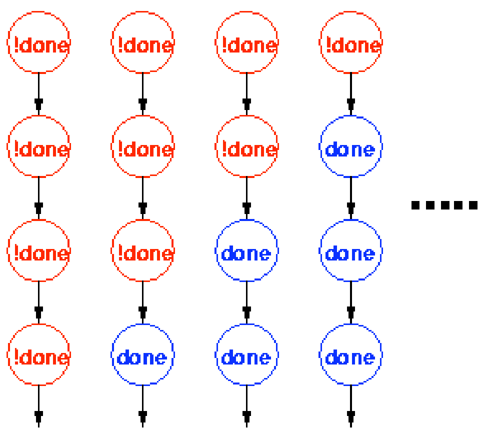
Content

Content

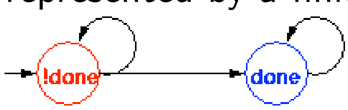
System's computations

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

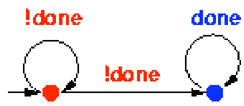
```
MODULE main
VAR  done: Boolean;
ASSIGN
  init(done):=0;
  next(done):= case
      !done: {0,1};
      done: done;
    esac;
```



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

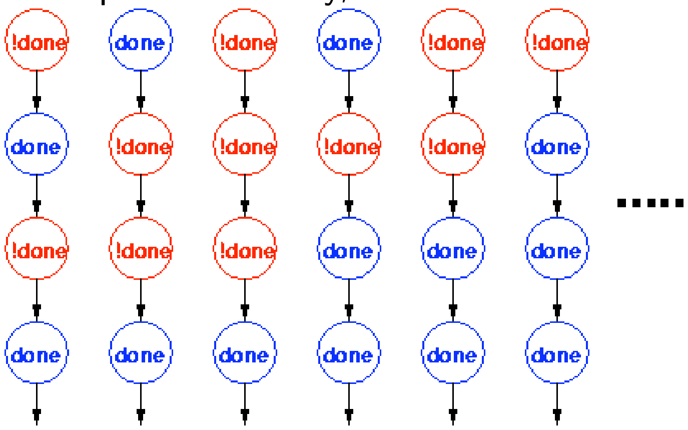


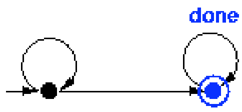
or



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
  $\Longrightarrow$ 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.

$$\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2) \Longleftrightarrow \mathcal{L}(A_1) \cap \overline{\mathcal{L}(A_2)} = \{\}$$

To solve the language containment problem, we need to know:

1. how to complement an automaton,
2. how to intersect two automata,
3. how to check the language emptiness of an automaton.

Content

Finite Word Languages

- An Alphabet Σ 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 Σ^*.
- 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.
  ϵ empty word.
  $u.v$ catenation.
  $u^i = u.u. \quad .u$ repeated i-times.
$U, V \subseteq \Sigma^*$ Finite word languages.

FSA Definition

Nondeterministic Finite State Automaton (NFA):

NFA is $(Q, \Sigma, \delta, I, F)$

Q Finite set of states.
Σ 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):

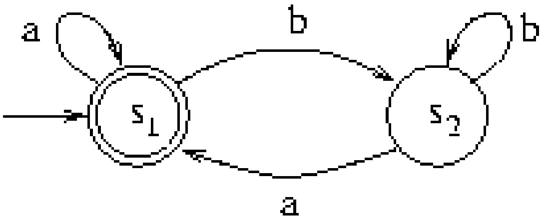
DFA has $\delta : Q \times \Sigma \rightarrow Q$, a total function.
Single initial state $I = \{q_0\}$.

Regular Languages

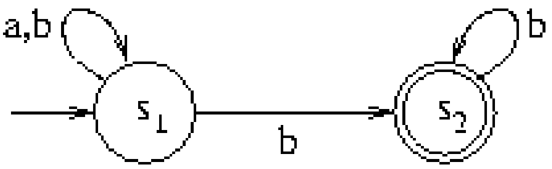
- A run of NFA A on $u = a_0, a_1, \ldots, a_{n-1}$ is a finite sequence of states $q_0, q_1, \ldots, q_n$ s.t. $q_0 \in I$ and $q_i \xrightarrow{a_i} q_{i+1}$ for $0 \leq i < n$.
- An accepting run is one where the last state $q_n \in F$.
- The language accepted by A
  $\mathcal{L}(A) = \{u \in \Sigma^*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

Finite State Automata

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.



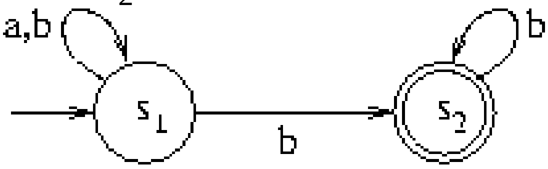
Determinisation

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

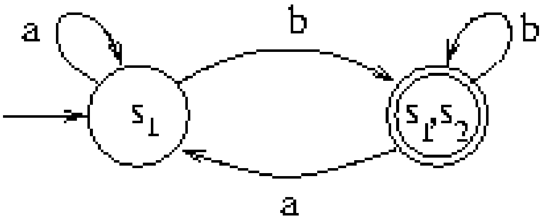
- Each state of A' corresponds to a set $\{s_1, ..., s_j\}$ of states in A ($Q' \subseteq 2^Q$), with the intended meaning that :
  - A' is in the state $\{s_1, .., s_j\}$ if A is in one of the states $s_1, ..., s_j$
- The deterministic transition relation $\delta' : 2^Q \times \Sigma \longmapsto 2^Q$ 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 $I' =_{def} \{s_i \mid s_i \in I\}$
- 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 Σ we can construct NFA A over Σ s.t.

- $\mathcal{L}(A) = \overline{\mathcal{L}(A_1)}$ (Complement). $|A| = 2^{O(|A_1|)}$.
- $\mathcal{L}(A) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$ (union). $|A| = |A_1| + |A_2|$.
- $\mathcal{L}(A) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$ (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: $\overline{A'} = (Q', \Sigma', \delta', I', \overline{F'})$
- $|\overline{A'}| = |A'| = 2^{O(|A|)}$

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

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

  $\Longrightarrow A$ is an automaton which just runs nondeterministically either A_1 or A_2
- $\mathcal{L}(A) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$
- $|A| = |A_1| + |A_2|$

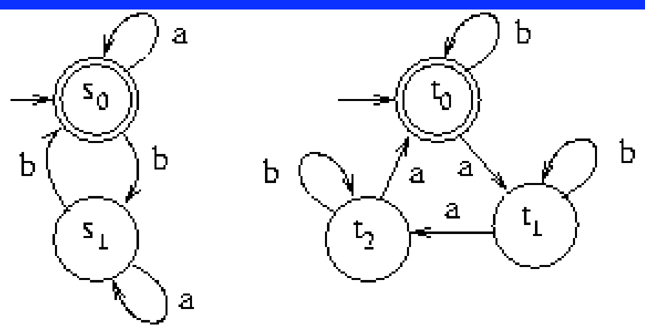
Synchronous Product Construction

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

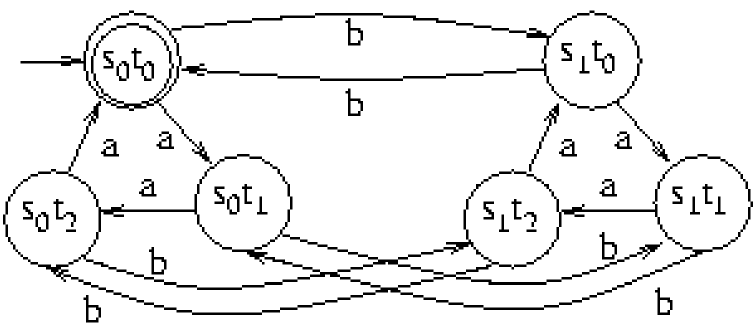
- $Q = Q_1 \times Q_2$. $\quad I = I_1 \times I_2$.
  $F = F_1 \times F_2$.
- $< p, q > \xrightarrow{a} < p', q' >$ iff $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$.

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

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 $\mathcal{L}(reg)$.

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 $reg(A)$.

Content

Infinite Word Languages

Modeling infinite computations of reactive systems.

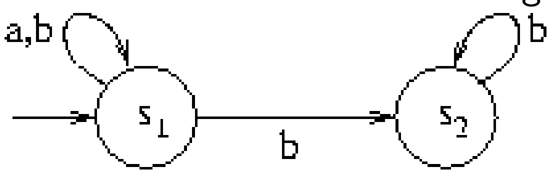
- An ω-word α over Σ is an infinite sequence
  $a_0, \ a_1, \ a_2 \ldots.$
  Formally, $\alpha : \mathbb{N} \rightarrow \Sigma$.
  The set of all infinite words is denoted by Σ^ω.
- A ω-language L is collection of ω-words, i.e. $L \subseteq \Sigma^\omega$.

  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,

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

(The set of states occurring infinitely many times in ρ.)

Büchi Automata

Nondeterministic Büchi Automaton

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

- A run ρ of A on omega word α is an infinite sequence $\rho = q_o, q_1, q_2, \ldots$ s.t. $q_0 \in I$ and $q_i \xrightarrow{a_i} q_{i+1}$ for $0 \leq i$.
- The run ρ is accepting if

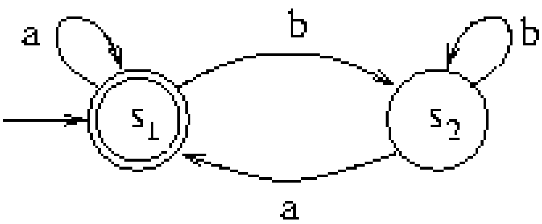
  $$Inf(\rho) \cap F \neq \emptyset.$$

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

Büchi Automaton: Example

Let $\Sigma = \{a, b\}$.

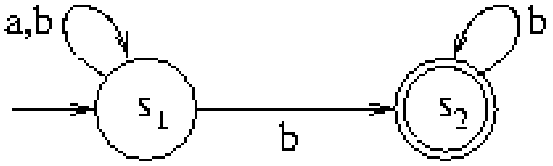
Let a Deterministic Büchi Automaton (DBA) A_1 be



- 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.

Büchi Automaton: Example (2)

Let a Nondeterministic Büchi Automaton (NBA) A_2 be

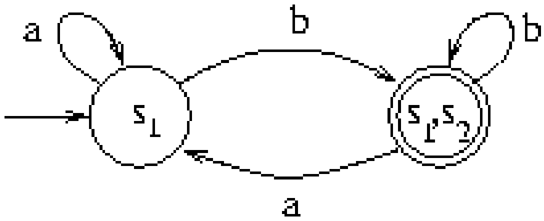


With $F = \{s_2\}$, automaton A_2 recognizes words with finitely many a.
Thus, $\mathcal{L}(A_2) = \overline{\mathcal{L}(A_1)}$.

Deterministic vs. Nondeterministic Büchi Automata

Theorem *DBA*s are strictly less powerful than *NBA*s.

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

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

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

  $\Longrightarrow$ A is an automaton which just runs nondeterministically either A_1 or A_2
- $\mathcal{L}(A) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$
- $|A| = |A_1| + |A_2|$
- (same construction as with ordinary automata)

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

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

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

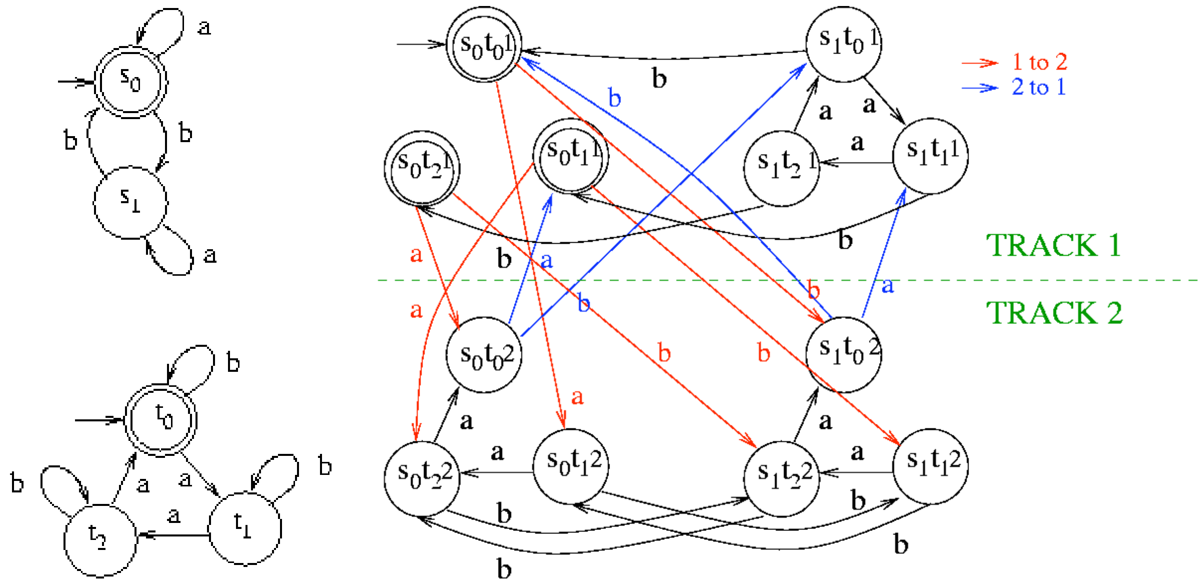
Theorem $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$.

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
  $\Longrightarrow$ 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



Closure Properties (2)

Theorem (complementation)
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)
(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 ω-regular if it has the form $\cup_{i=1}^{n} \; U_i.(V_i)^{\omega}$ where U_i, V_i are regular languages.

Theorem A language L is ω-regular iff it is NBA-recognizable.

Content

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:

```
nonempty :- initial(X),cn(X,Y),final(Y).

cn(X,Y) :- r(X,A,Y).
cn(X,Y) :- r(X,A,Z),cn(Z,Y).
```

  where `initial(X)` denotes that `X` is an initial state; `final(X)` denotes that `X` is a final state; `r(X,A,Y)` denotes that a transition from `X` to `Y` reading `A`; and `cn(.,.)` is **the transitive closure** of `r(X,A,Y)` projected on `X`,`Y`.

  Notice that `cn(.,.)` is not expressible in FOL.

- Reachability is a well-known problem on graphs, its complexity is NLOGSPACE-complete. →

  Thm. *Nonemptiness for NFA a is **NLOGSPACE**-complete.*

  Practical algorithms have a **linear** cost.

Nonemptiness of Büchi Automata

- For Büchi automata, nonemptiness algorithms are based on **fair reachability**
- In Datalog/Prolog notation:

```
nonempty :- initial(X),cn(X,Y),final(Y),cn(Y,Y).

cn(X,Y) :- r(X,A,Y).
cn(X,Y) :- r(X,A,Z),cn(Z,Y).
```

  where, as before, `initial(X)` denotes that `X` is an initial state; `final(X)` denotes that `X` is a final state; `r(X,A,Y)` denotes that a transition from `X` to `Y` reading `A`; and `cn(.,.)` 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: NLOGSPACE-complete. →

  Thm. *Nonemptiness for Büchi automata is **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.

Content

Automata-Theoretic LTL Model Checking

- $M \models \mathbf{A}\psi$ (CTL*)

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

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

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

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

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

- 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 ψ)

$\Longrightarrow A_M \times A_{\neg\psi}$ represents all and only the paths appearing in M and not in ψ.

Automata-Theoretic LTL M.C. (dual version)

- $M \models \mathbf{E}\varphi$

$\Longleftrightarrow$ $M \not\models \mathbf{A}\neg\varphi$

$\Longleftrightarrow$ ...

$\Longleftrightarrow$ $\mathcal{L}(A_M \times A_\varphi) \neq \{\}$

- A_M is a Büchi Automaton equivalent to M (which represents all and only the executions of M)
- A_φ is a Büchi Automaton which represents all and only the paths that satisfy φ

$\Longrightarrow$ $A_M \times A_\varphi$ represents all and only the paths appearing in both A_M and A_φ.

Automata-Theoretic LTL Model Checking

Four steps:

1. Compute A_M
2. Compute A_φ
3. Compute the product $A_M \times A_\varphi$
4. Check the emptiness of $\mathcal{L}(A_M \times A_\varphi)$

Content

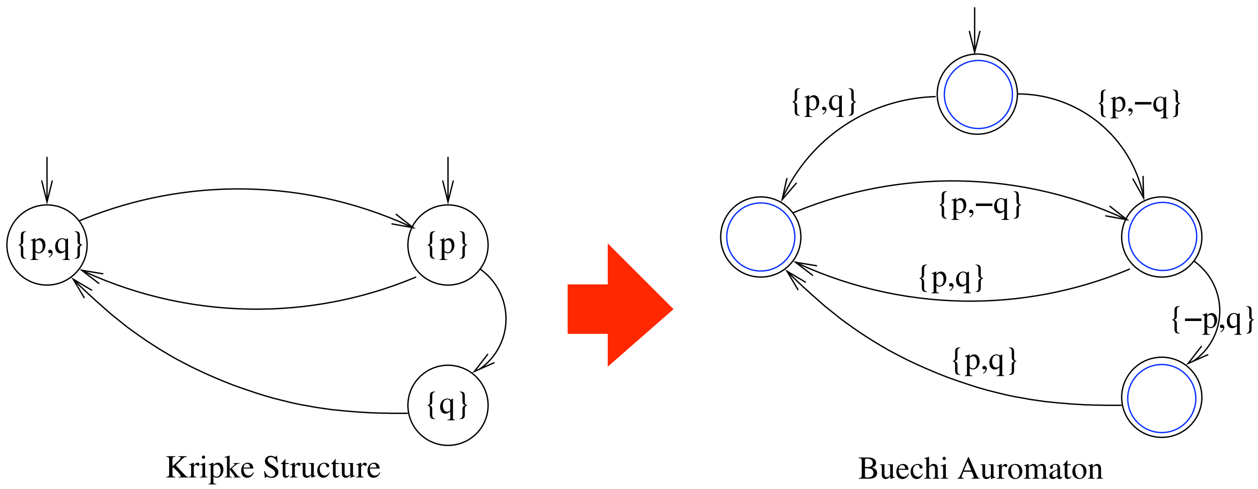
Computing an NBA A_M from a Kripke Structure M

- Transforming a K.S. $M = \langle S, S_0, R, L, AP \rangle$ 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:

$$\begin{array}{ll} \delta: & q \xrightarrow{a} q' \text{ iff } (q, q') \in R \textit{ and } L(q') = a \\ & init \xrightarrow{a} q \text{ iff } q \in S_0 \textit{ and } L(q) = a \end{array}$$

- $\mathcal{L}(A_M) = \mathcal{L}(M)$
- $|A_M| = |M| + 1$

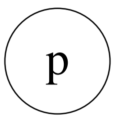
Computing a NBA A_M from a Kripke Structure M: Example



⟹Substantially, add one initial state, move labels from states to incoming edges, set all states as accepting states

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).

Content

Translation problem

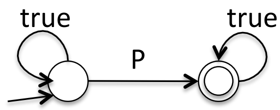
Problem

Given an LTL formula ϕ, find a Büchi Automaton that accepts the same language of ϕ.

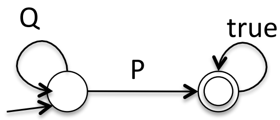
- It is a fundamental problem in LTL model checking (in other words, every model checking algorithm that verifies the correctness of an LTL formula translates it in some sort of finite-state machine).
- We will translate LTL in a (equivalent) variant of Büchi Automata called Labeled Generalized Büchi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

Translation from LTL to Büchi Automata: examples

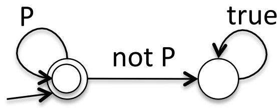
- ◆P
  $\mathcal{L}$ = true* P true$^{\omega}$



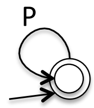
- Q **U** P
  $\mathcal{L}$ = Q* P true$^{\omega}$



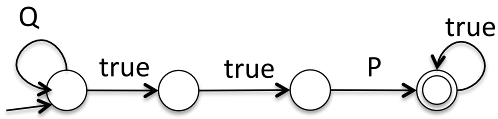
- ■P
  $\mathcal{L}$ = P$^{\omega}$



oppure

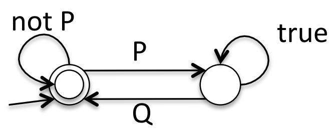


- Q **U** ●●P
  $\mathcal{L}$ = Q* true true P true$^{\omega}$

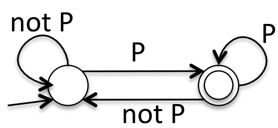


Translation from LTL to Büchi Automata: examples

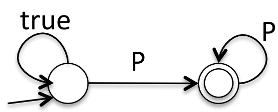
- ■(P -> ◆Q)
  $\mathcal{L}$ = (not P* P true Q true)$^{\omega}$ U
  (not P* P true Q true)* not P$^{\omega}$



- ■◆P
  $\mathcal{L}$ = (true*P)$^{\omega}$



- ◆■P
  $\mathcal{L}$ = true*P$^{\omega}$



Content

Automata-Theoretic LTL Model Checking: complexity

Four steps:

1. Compute A_M:
2. Compute A_φ:
3. Compute the product $A_M \times A_\varphi$:
4. Check the emptiness of $\mathcal{L}(A_M \times A_\varphi)$:

Automata-Theoretic LTL Model Checking: complexity

Four steps:

1. Compute A_M: $|A_M| = O(|M|)$
2. Compute A_φ:
3. Compute the product $A_M \times A_\varphi$:
4. Check the emptiness of $\mathcal{L}(A_M \times A_\varphi)$:

Automata-Theoretic LTL Model Checking: complexity

Four steps:

1. Compute A_M: $|A_M| = O(|M|)$
2. Compute A_φ: $|A_\varphi| = O(2^{|\varphi|})$
3. Compute the product $A_M \times A_\varphi$:
4. Check the emptiness of $\mathcal{L}(A_M \times A_\varphi)$:

Automata-Theoretic LTL Model Checking: complexity

Four steps:

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

Automata-Theoretic LTL Model Checking: complexity

Four steps:

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

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

Final Remarks

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