Finite Automata and Languages

Ashutosh Trivedi

Department of Computer Science and Engineering,
IIT Bombay

CS620: New Trends in IT: Modeling and Verification of Cyber-Physical Systems
(2 August 2013)

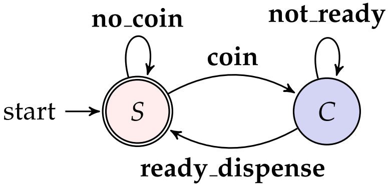
Computation With Finitely Many States

Nondeterministic Finite State Automata

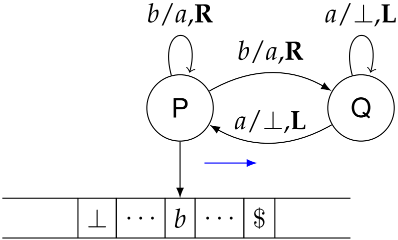
Alternation

Machines and their Mathematical Abstractions

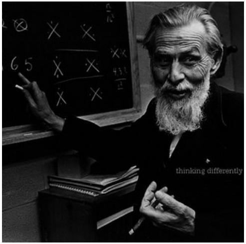
Finite instruction machine with finite memory (Finite State Automata)

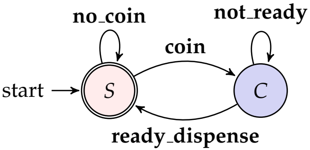


Finite instruction machine with unbounded memory (Turing machine)



Finite State Automata





- Introduced first by two neuro-psychologists Warren S. McCullough and Walter Pitts in 1943 as a model for human brain!
- Finite automata can naturally model microprocessors and even software programs working on variables with bounded domain
- Capture so-called regular sets of sequences that occur in many different fields (logic, algebra, regEx)
- Nice theoretical properties
- Applications in digital circuit/protocol verification, compilers, pattern recognition, etc.

Calculemus! — Gottfried Wilhelm von Leibniz

Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

– Recognize a string of an even length.

Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

- Recognize a string of an even length.
- Recognize a binary string of an even number of 0′s.

Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

- Recognize a string of an even length.
- Recognize a binary string of an even number of 0's.
- Recognize a binary string of an odd number of 0's.

Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

- Recognize a string of an even length.
- Recognize a binary string of an even number of 0's.
- Recognize a binary string of an odd number of 0's.
- Recognize a string that contains your roll number.

Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

- Recognize a string of an even length.
- Recognize a binary string of an even number of 0's.
- Recognize a binary string of an odd number of 0's.
- Recognize a string that contains your roll number.
- Recognize a binary (decimal) string that is a multiple of 2.

Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

- Recognize a string of an even length.
- Recognize a binary string of an even number of 0's.
- Recognize a binary string of an odd number of 0's.
- Recognize a string that contains your roll number.
- Recognize a binary (decimal) string that is a multiple of 2.
- Recognize a binary (decimal) string that is a multiple of 3.

Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

- Recognize a string of an even length.
- Recognize a binary string of an even number of 0's.
- Recognize a binary string of an odd number of 0's.
- Recognize a string that contains your roll number.
- Recognize a binary (decimal) string that is a multiple of 2.
- Recognize a binary (decimal) string that is a multiple of 3.
- Recognize a string with well-matched parenthesis.

Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

- Recognize a string of an even length.
- Recognize a binary string of an even number of 0's.
- Recognize a binary string of an odd number of 0's.
- Recognize a string that contains your roll number.
- Recognize a binary (decimal) string that is a multiple of 2.
- Recognize a binary (decimal) string that is a multiple of 3.
- Recognize a string with well-matched parenthesis.
- Recognize a # separated string of the form $w\#\overline{w}$.

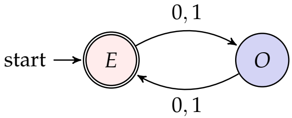
Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

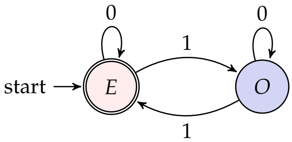
- Recognize a string of an even length.
- Recognize a binary string of an even number of 0's.
- Recognize a binary string of an odd number of 0's.
- Recognize a string that contains your roll number.
- Recognize a binary (decimal) string that is a multiple of 2.
- Recognize a binary (decimal) string that is a multiple of 3.
- Recognize a string with well-matched parenthesis.
- Recognize a # separated string of the form $w\#\overline{w}$.
- Recognize a string with a prime number of 1's

Finite State Automata

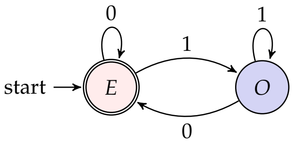
Automaton accepting strings of even length:



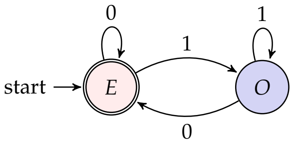
Automaton accepting strings with an even number of 1′s:



Automaton accepting even strings (multiple of 2):



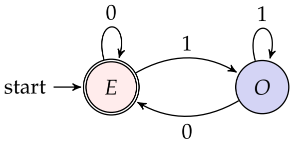
Deterministic Finite State Automata (DFA)



A finite state automaton is a tuple $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$, where:

- S is a finite set called the states;
- Σ is a finite set called the alphabet;
- $\delta : S \times \Sigma \to S$ is the transition function;
- $s_0 \in S$ is the start state; and
- $F \subseteq S$ is the set of accept states.

Deterministic Finite State Automata (DFA)



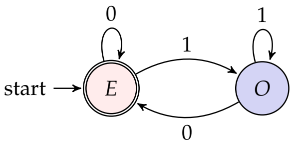
A finite state automaton is a tuple $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$, where:

- S is a finite set called the states;
- Σ is a finite set called the alphabet;
- $\delta : S \times \Sigma \to S$ is the transition function;
- $s_0 \in S$ is the start state; and
- $F \subseteq S$ is the set of accept states.

For a function $\delta : S \times \Sigma \to S$ we define extended transition function $\hat{\delta} : S \times \Sigma^* \to S$ using the following inductive definition:

$$\hat{\delta}(q, w) = \begin{cases} q & \text{if } w = \varepsilon \\ \delta(\hat{\delta}(q, x), a) & \text{if } w = xa \text{ s.t. } x \in \Sigma^* \text{ and } a \in \Sigma. \end{cases}$$

Deterministic Finite State Automata (DFA)



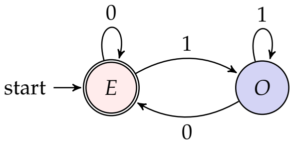
A finite state automaton is a tuple $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$, where:

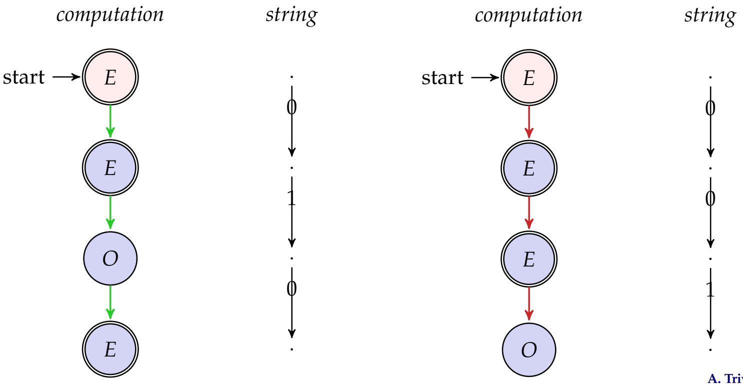
- S is a finite set called the states;
- Σ is a finite set called the alphabet;
- $\delta : S \times \Sigma \to S$ is the transition function;
- $s_0 \in S$ is the start state; and
- $F \subseteq S$ is the set of accept states.

The language $L(\mathcal{A})$ accepted by a DFA $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ is defined as:

$$L(\mathcal{A}) \stackrel{\text{def}}{=} \{w \;:\; \hat{\delta}(w) \in F\}.$$

Computation or Run of a DFA





Deterministic Finite State Automata

Semantics using extended transition function:

– The language $L(\mathcal{A})$ accepted by a DFA $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ is defined as:

$$L(\mathcal{A}) \stackrel{\text{def}}{=} \{w \;:\; \hat{\delta}(w) \in F\}.$$

Semantics using accepting computation:

– A computation or a run of a DFA $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ on a string $w = a_0 a_1 \ldots a_{n-1}$ is the finite sequence

$$s_0, a_1 s_1, a_2, \ldots, a_{n-1}, s_n$$

where s_0 is the starting state, and $\delta(s_{i-1}, a_i) = s_{i+1}$.

– A string w is accepted by a DFA $\mathcal{A}$ if the last state of the unique computation of $\mathcal{A}$ on w is an accept state, i.e. $s_n \in F$.

– Language of a DFA $\mathcal{A}$

$$L(\mathcal{A}) = \{w \;:\; \text{string } w \text{ is accepted by DFA } \mathcal{A}\}.$$

Proposition

Both semantics define the same language. *Proof by induction.*

Properties of Regular Languages

Definition (Regular Languages)

A language is called regular if it is accepted by a finite state automaton.

Let A and B be languages (remember they are sets). We define the following operations on them:

- Union: $A \cup B = \{w \,:\, w \in A \text{ or } w \in B\}$
- Concatenation: $AB = \{wv \,:\, w \in A \text{ and } v \in B\}$
- Closure (Kleene Closure, or Star):
  $A^* = \{w_1 w_2 \ldots w_k \,:\, k \geq 0 \text{ and } w_i \in A\}$. In other words:

  $$A^* = \cup_{i \geq 0} A^i$$

  where $A^0 = \emptyset$, $A^1 = A$, $A^2 = AA$, and so on.
- Complementation $\Sigma^* \setminus A = \{w \in \Sigma^* \,:\, w \notin A\}$.

Properties of Regular Languages

- The class of regular languages is closed under
  - union,
  - intersection,
  - complementation
  - concatenation, and
  - Kleene closure.
- Decidability of language-theoretic problems
  - Emptiness Problem
  - Membership Problem
  - Universality Problem
  - Equivalence Problem
  - Language Inclusion Problem
  - Minimization Problem

Goal: To study these problems for timed automata

Closure under Union

Lemma

The class of regular languages is closed under union.

Proof.

– Let A_1 and A_1 be regular languages.

Closure under Union

Lemma

The class of regular languages is closed under union.

Proof.

- Let A_1 and A_1 be regular languages.
- Let $M_1 = (S_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (S_2, \Sigma, \delta_2, s_2, F_2)$ be finite automata accepting these languages.

Closure under Union

Lemma

The class of regular languages is closed under union.

Proof.

- Let A_1 and A_1 be regular languages.
- Let $M_1 = (S_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (S_2, \Sigma, \delta_2, s_2, F_2)$ be finite automata accepting these languages.
- Simulate both automata together!
- The language $A \cup B$ is accept by the resulting finite state automaton, and hence is regular.

$\square$

Closure under Union

Lemma

The class of regular languages is closed under union.

Proof.

- Let A_1 and A_1 be regular languages.
- Let $M_1 = (S_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (S_2, \Sigma, \delta_2, s_2, F_2)$ be finite automata accepting these languages.
- Simulate both automata together!
- The language $A \cup B$ is accept by the resulting finite state automaton, and hence is regular.

$\square$

Class Exercise: Extend this construction for intersection.

Closure under Concatenation

Lemma

The class of regular languages is closed under concatenation.

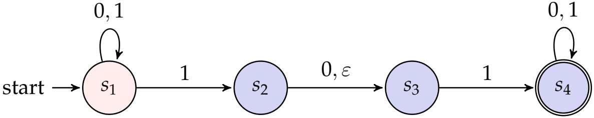
Proof.

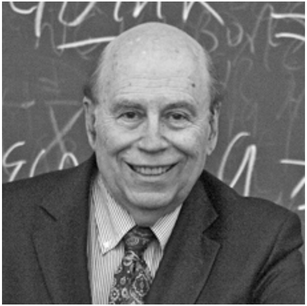
(Attempt).

- Let A_1 and A_1 be regular languages.
- Let $M_1 = (S_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (S_2, \Sigma, \delta_2, s_2, F_2)$ be finite automata accepting these languages.
- How can we find an automaton that accepts the concatenation?
- Does this automaton fit our definition of a finite state automaton?
- Determinism vs Non-determinism

□

Nondeterministic Finite State Automata

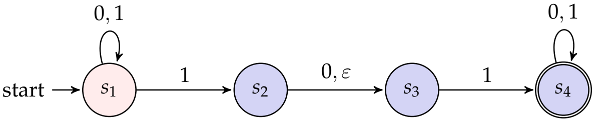




Michael O. Rabin

Dana Scott

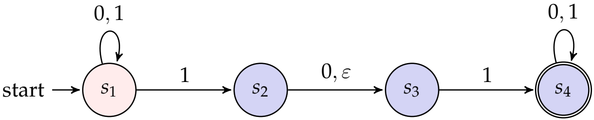
Non-deterministic Finite State Automata



A non-deterministic finite state automaton (NFA) is a tuple $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$, where:

- S is a finite set called the states;
- Σ is a finite set called the alphabet;
- $\delta : S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$ is the transition function;
- $s_0 \in S$ is the start state; and
- $F \subseteq S$ is the set of accept states.

Non-deterministic Finite State Automata



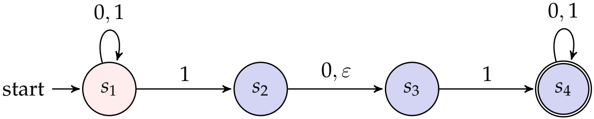
A non-deterministic finite state automaton (NFA) is a tuple $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$, where:

- S is a finite set called the states;
- Σ is a finite set called the alphabet;
- $\delta : S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$ is the transition function;
- $s_0 \in S$ is the start state; and
- $F \subseteq S$ is the set of accept states.

For a function $\delta : S \times \Sigma \to 2^S$ we define extended transition function $\hat{\delta} : S \times \Sigma^* \to 2^S$ using the following inductive definition:

$$\hat{\delta}(q, w) = \begin{cases} \{q\} & \text{if } w = \varepsilon \\ \bigcup_{p \in \hat{\delta}(q,x)} \delta(p, a) & \text{if } w = xa \text{ s.t. } x \in \Sigma^* \text{ and } a \in \Sigma. \end{cases}$$

Non-deterministic Finite State Automata



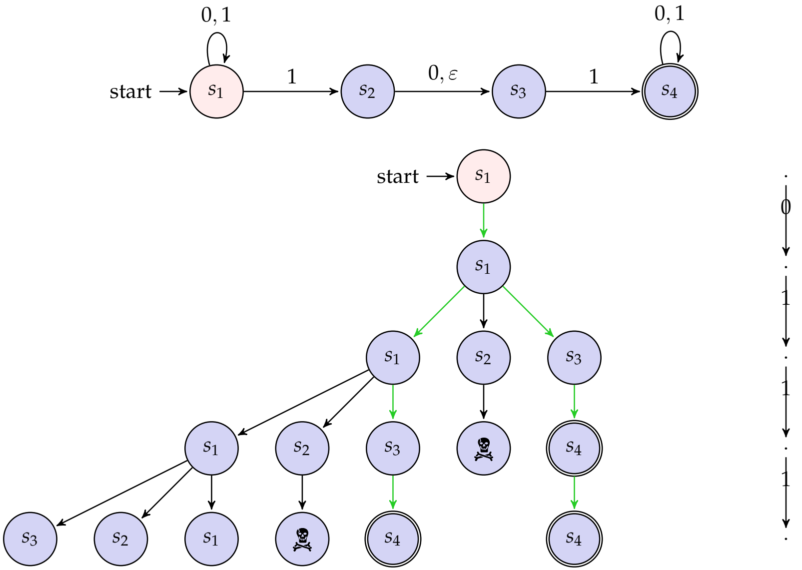
A non-deterministic finite state automaton (NFA) is a tuple $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$, where:

- S is a finite set called the states;
- Σ is a finite set called the alphabet;
- $\delta : S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$ is the transition function;
- $s_0 \in S$ is the start state; and
- $F \subseteq S$ is the set of accept states.

The language $L(\mathcal{A})$ accepted by an NFA $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ is defined as:

$$L(\mathcal{A}) \stackrel{\text{def}}{=} \{w \ : \ \hat{\delta}(w) \cap F \neq \emptyset\}.$$

Computation or Run of an NFA



Non-deterministic Finite State Automata

Semantics using extended transition function:

- The language $L(\mathcal{A})$ accepted by an NFA $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ is defined:

$$L(\mathcal{A}) \stackrel{\text{def}}{=} \{w \ : \ \hat{\delta}(w) \cap F \neq \emptyset\}.$$

Semantics using accepting computation:

- A computation or a run of a NFA on a string $w = a_0 a_1 \dots a_{n-1}$ is a finite sequence

$$s_0, r_1, s_1, r_2, \dots, r_{k-1}, s_n$$

  where s_0 is the starting state, and $s_{i+1} \in \delta(s_{i-1}, r_i)$ and $r_0 r_1 \dots r_{k-1} = a_0 a_1 \dots a_{n-1}$.
- A string w is accepted by an NFA $\mathcal{A}$ if the last state of some computation of $\mathcal{A}$ on w is an accept state $s_n \in F$.
- Language of an NFA $\mathcal{A}$

$$L(\mathcal{A}) = \{w \ : \ \text{string } w \text{ is accepted by NFA } \mathcal{A}\}.$$

Proposition

Both semantics define the same language. *Proof by induction.*

Why study NFA?

NFA are often more convenient to design than DFA, e.g.:

- $\{w : w$ contains 1 in the third last position$\}$.
- $\{w :: w$ is a multiple of 2 or a multiple of 3$\}$.
- Union and intersection of two DFAs as an NFA
- Exponentially succinct than DFA
  - Consider the language of strings having n-th symbol from the end is 1.
  - DFA has to remember last n symbols, and
  - hence any DFA needs at least 2^n states to accept this language.

Why study NFA?

NFA are often more convenient to design than DFA, e.g.:

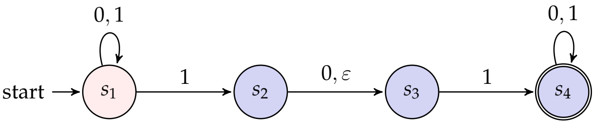
- $\{w : w \text{ contains 1 in the third last position}\}$.
- $\{w :: w \text{ is a multiple of 2 or a multiple of 3}\}$.
- Union and intersection of two DFAs as an NFA
- Exponentially succinct than DFA
  - Consider the language of strings having n-th symbol from the end is 1.
  - DFA has to remember last n symbols, and
  - hence any DFA needs at least 2^n states to accept this language.

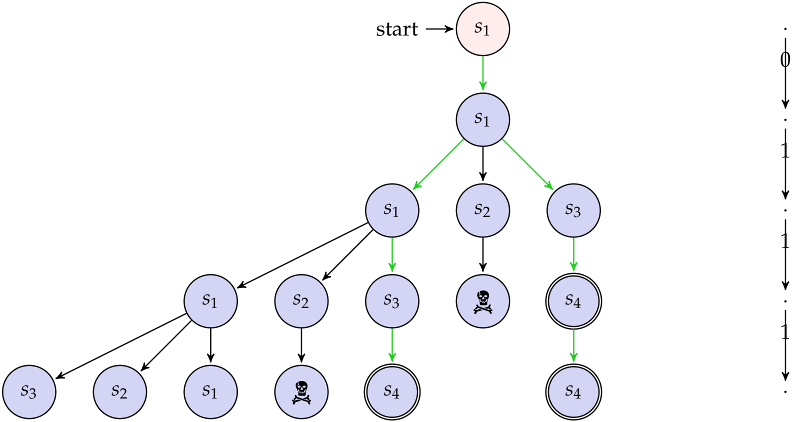
And, surprisingly perhaps:

Theorem (DFA=NFA)

Every non-deterministic finite automaton has an equivalent (accepting the same language) deterministic finite automaton. Subset construction.

Computation of an NFA: An observation





ε-free NFA = DFA

Let $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ be an ε-free NFA. Consider the DFA $Det(\mathcal{A}) = (S', \Sigma', \delta', s'_0, F')$ where

- $S' = 2^S$,
- $\Sigma' = \Sigma$,
- $\delta' : 2^S \times \Sigma \to 2^S$ such that $\delta'(P, a) = \bigcup_{s \in P} \delta(s, a)$,
- $s'_0 = \{s_0\}$, and
- $F' \subseteq S'$ is such that $F' = \{P \,:\, P \cap F \neq \emptyset\}$.

Theorem (ε-free NFA = DFA)

$L(\mathcal{A}) = L(Det(\mathcal{A}))$. *By induction, hint $\hat{\delta}(s_0, w) = \hat{\delta}'(\{s_0\}, w)$.*

Exercise (3.1)

Extend the proof for NFA with ε transitions. *hint: ε-closure*

Proof of correctness: $L(\mathcal{A}) = L(Det(\mathcal{A}))$.

The proof follows from the observation that $\hat{\delta}(s_0, w) = \hat{\delta}'(\{s_0\}, w)$. We prove it by induction on the length of w.

- Base case: Let the size of w be 0, i.e. $w = \varepsilon$. The base case follows immediately from the definition of extended transition functions:

$$\hat{\delta}(s_0, \varepsilon) = \varepsilon \text{ and } \hat{\delta}'(\{s_0\}, w) = \varepsilon.$$

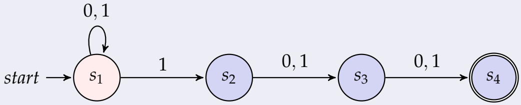
- Induction Hypothesis: Assume that for all strings $w \in \Sigma^*$ of size n we have that $\hat{\delta}(s_0, w) = \hat{\delta}'(\{s_0\}, w)$.
- Induction Step: Let $w = xa$ where $x \in \Sigma^*$ and $a \in \Sigma$ be a string of size $n + 1$, and hence x is of size n. Now observe,

$$\begin{aligned}
\hat{\delta}(s_0, xa) &= \bigcup_{s \in \hat{\delta}(s_0, x)} \delta(s, a), \text{by definition of } \hat{\delta}. \\
&= \bigcup_{s \in \hat{\delta}'(\{s_0\}, x)} \delta(s, a), \text{from inductive hypothesis.} \\
&= \delta'(\hat{\delta}'(\{s_0\}, x), a), \text{ from definition } \delta'(P, a) = \bigcup_{s \in P} \delta(s, a). \\
&= \hat{\delta}'(\{s_0\}, xa), \text{ by definition of } \hat{\delta}'.
\end{aligned}$$

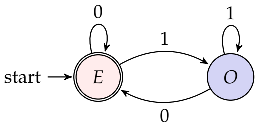
Equivalence of NFA and DFA

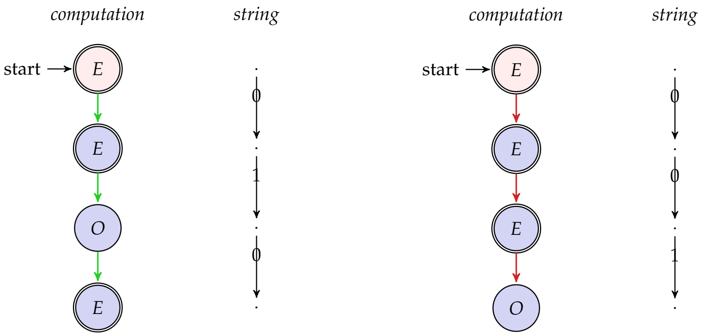
Exercise (In class)

Determinize the following automaton:



Complementation of the Language of a DFA





Hint: Simply swap the accepting and non-accepting states!

Complementation of a DFA

Theorem

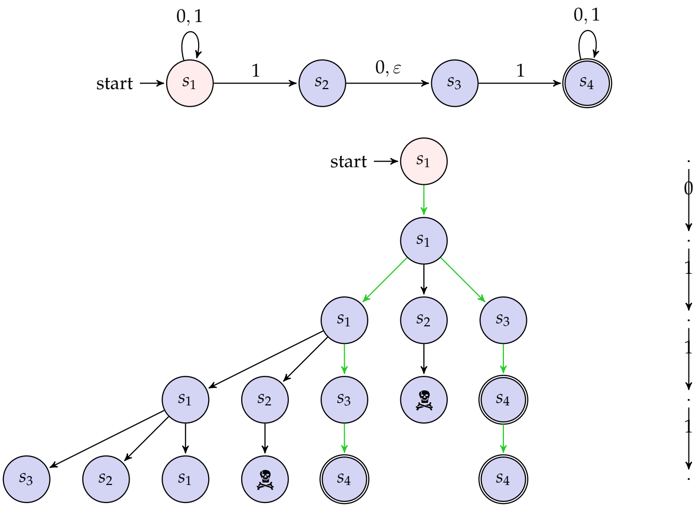
Complementation of the language of a DFA $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ *is the language accepted by the DFA* $\mathcal{A}' = (S, \Sigma, \delta, s_0, S \setminus F)$.

Proof.

- $L(\mathcal{A}) = \{w \in \Sigma^* \ : \ \hat{\delta}(s_0, w) \in F\}$,
- $\Sigma^* \setminus L(\mathcal{A}) = \{w \in \Sigma^* \ : \ \hat{\delta}(s_0, w) \notin F\}$,
- $L(\mathcal{A}') = \{w \in \Sigma^* \ : \ \hat{\delta}(s_0, w) \in S \setminus F\}$, and
- transition function is total.

□

Complementation of the language of an NFA



Question: Can we simply swap the accepting and non-accepting states?

Complementation of the language of a NFA

Question: Can we simply swap the accepting and non-accepting states?

Let the NFA $\mathcal{A}$ be $(S, \Sigma, \delta, s_0, F)$ and let the NFA $\mathcal{A}'$ be $(S, \Sigma, \delta, s_0, S \setminus F)$ the NFA after swapping the accepting states.

- $L(\mathcal{A}) = \{w \in \Sigma^* \,:\, \hat{\delta}(s_0, w) \cap F \neq \emptyset\}$,
- $L(\mathcal{A}') = \{w \in \Sigma^* \,:\, \hat{\delta}(s_0, w) \cap (S \setminus F) \neq \emptyset\}$.
- Consider, the complement language of $\mathcal{A}$

$$\begin{aligned} \Sigma^* \setminus L(\mathcal{A}) &= \{w \in \Sigma^* \,:\, \hat{\delta}(s_0, w) \cap F = \emptyset\} \\ &= \{w \in \Sigma^* \,:\, \hat{\delta}(s_0, w) \subseteq S \setminus F\}. \end{aligned}$$

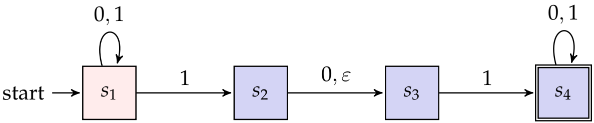
- Hence $L(\mathcal{A}')$ does not quite capture the complement. Moreover, the condition for $\Sigma^* \setminus L(\mathcal{A})$ is not quite captured by either DFA or NFA.

Computation With Finitely Many States

Nondeterministic Finite State Automata

Alternation

Universal Non-deterministic Finite Automata



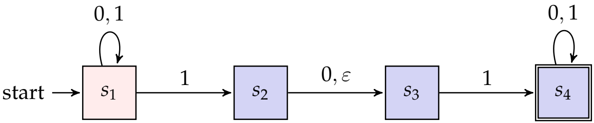
A universal non-deterministic finite state automaton (UNFA) is a tuple $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$, where:

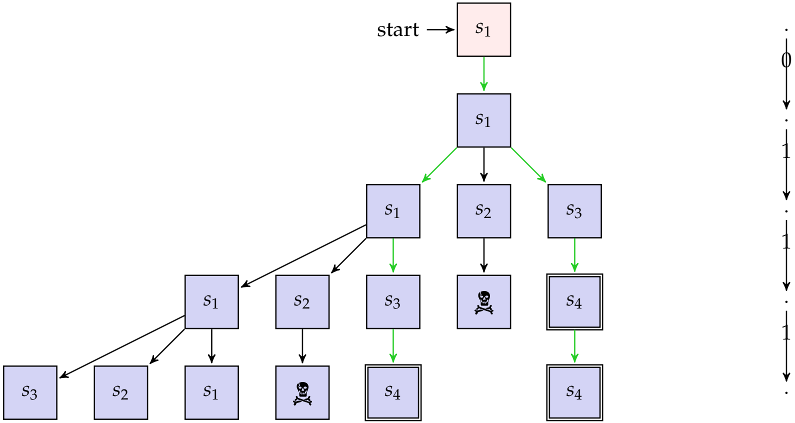
- S is a finite set called the states;
- Σ is a finite set called the alphabet;
- $\delta : S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$ is the transition function;
- $s_0 \in S$ is the start state; and
- $F \subseteq S$ is the set of accept states.

The language $L(\mathcal{A})$ accepted by a UNFA $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ is defined as:

$$L(\mathcal{A}) \stackrel{\text{def}}{=} \{w \ : \ \hat{\delta}(w) \subseteq F\}.$$

Computation or Run of an UNFA





Universal Non-deterministic Finite Automata

Semantics using extended transition function:

- The language $L(\mathcal{A})$ accepted by an NFA $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ is defined as:

$$L(\mathcal{A}) \stackrel{\text{def}}{=} \{w \ : \ \hat{\delta}(w) \subseteq F\}.$$

Semantics using accepting computation:

- A computation or a run of a NFA on a string $w = a_0 a_1 \dots a_{n-1}$ is a finite sequence

$$s_0, r_1, s_1, r_2, \dots, r_{k-1}, s_n$$

  where s_0 is the starting state, and $s_{i+1} \in \delta(s_{i-1}, r_i)$ and $r_0 r_1 \dots r_{k-1} = a_0 a_1 \dots a_{n-1}$.
- A string w is accepted by an NFA $\mathcal{A}$ if the last state of all computations of $\mathcal{A}$ on w is an accept state $s_n \in F$.
- Language of an NFA $\mathcal{A}$

$$L(\mathcal{A}) = \{w \ : \ \text{string } w \text{ is accepted by NFA } \mathcal{A}\}.$$

Proposition

Both semantics define the same language. *Proof by induction.*

ε-free UNFA = DFA

Let $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ be an ε-free UNFA. Consider the DFA $Det(\mathcal{A}) = (S', \Sigma', \delta', s'_0, F')$ where

- $S' = 2^S$,
- $\Sigma' = \Sigma$,
- $\delta' : 2^S \times \Sigma \to 2^S$ such that $\delta'(P, a) = \bigcup_{s \in P} \delta(s, a)$,
- $s'_0 = \{s_0\}$, and
- $F' \subseteq S'$ is such that $F' = \{P \,:\, P \subseteq F\}$.

Theorem (ε-free UNFA = DFA)

$L(\mathcal{A}) = L(Det(\mathcal{A}))$. *By induction, hint $\hat{\delta}(s_0, w) = \hat{\delta}'(s_0, w)$.*

ε-free UNFA = DFA

Let $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ be an ε-free UNFA. Consider the DFA $Det(\mathcal{A}) = (S', \Sigma', \delta', s'_0, F')$ where

- $S' = 2^S$,
- $\Sigma' = \Sigma$,
- $\delta' : 2^S \times \Sigma \to 2^S$ such that $\delta'(P, a) = \bigcup_{s \in P} \delta(s, a)$,
- $s'_0 = \{s_0\}$, and
- $F' \subseteq S'$ is such that $F' = \{P \ : \ P \subseteq F\}$.

Theorem (ε-free UNFA = DFA)

$L(\mathcal{A}) = L(Det(\mathcal{A}))$. *By induction, hint $\hat{\delta}(s_0, w) = \hat{\delta}'(s_0, w)$.*

Exercise (3.2)

Extend the proof for UNFA with ε transitions.

Complementation of an NFA

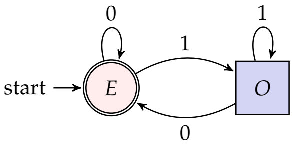
Theorem

Complementation of the language of an NFA $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ *is the language accepted by the UNFA* $\mathcal{A}' = (S, \Sigma, \delta, s_0, S \setminus F)$.

Exercise (3.3)

Write a formal proof for this theorem.

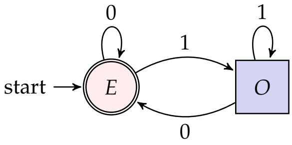
Alternating Finite State Automata



Ashok K. Chandra

Larry J. Stockmeyer

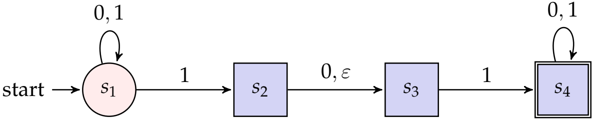
Alternating Finite State Automata

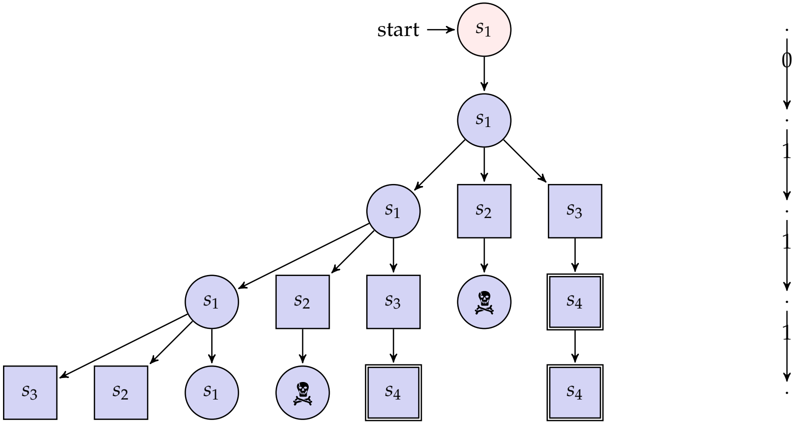


An alternating finite state automaton (AFA) is a tuple $\mathcal{A} = (S, S_\exists, S_\forall, \Sigma, \delta, s_0, F)$, where:

- S is a finite set called the states with a partition $S_\exists$ and $S_\forall$;
- Σ is a finite set called the alphabet;
- $\delta : S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$ is the transition function;
- $s_0 \in S$ is the start state; and
- $F \subseteq S$ is the set of accept states.

Computation or Run of an AFA





Universal Non-deterministic Finite Automata

- A computation or a run of a AFA on a string $w = a_0 a_1 \dots a_{n-1}$ is a game graph $\mathcal{G}(\mathcal{A}, w) = (S \times \{0, 1, 2, \dots, n-1\}, E)$ where:
  - Nodes in $S_\exists \times \{0, 1, 2, \dots, n-1\}$ are controlled by Eva and nodes in $S_\forall \times \{0, 1, 2, \dots, n\}$ are controlled by Adam; and
  - $((s, i), (s', i+1)) \in E$ if $s' \in \delta(s, a_i)$.
- Initially a token is in $(s_0, 0)$ node, and at every step the controller of the current node chooses the successor node.
- Eva wins if the node reached at level i is an accepting state node, otherwise Adam wins.
- We say that Eva has a winning strategy if she can make her decisions no matter how Adam plays.
- A string w is accepted by an AFA $\mathcal{A}$ if Eva has a winning strategy in the graph $\mathcal{G}(\mathcal{A}, w)$.
- Language of an AFA $\mathcal{A}$ $L(\mathcal{A}) = \{w \ : \ \text{string } w \text{ is accepted by AFA } \mathcal{A}\}$.
- Example.

ε-free AFA = NFA

Let $\mathcal{A} = (S, S_\exists, S_\forall, \Sigma, \delta, s_0, F)$ be an ε-free AFA. Consider the NFA $NDet(\mathcal{A}) = (S', \Sigma', \delta', s_0', F')$ where

- $S' = 2^S$,
- $\Sigma' = \Sigma$,
- $\delta' : 2^S \times \Sigma \to 2^{2^S}$ such that $Q \in \delta'(P, a)$ if
  - for all universal states $p \in P \cap S_\forall$ we have that $\delta(p, a) \subseteq Q$ and
  - for all existential states $p \in P \cap S_\exists$ we have that $\delta(p, a) \cap Q \neq \emptyset$,
- $s_0' = \{s_0\}$, and
- $F' \subseteq S'$ is such that $F' = 2^F \setminus \emptyset$.

Theorem (ε-free AFA = NFA)

$L(\mathcal{A}) = L(Det(\mathcal{A}))$.