

Abstraction of Problems

- Data: abstracted as a word in a given alphabet.
 - Σ : alphabet, a finite, non-empty set of symbols.
 - Σ^* : all the words of finite length built up using Σ :
- Conditions: abstracted as a set of words, called *language*
 - Any subset $L \subseteq \Sigma^*$ is a formal language.
- Unknown: Implicitly a Boolean variable: true if a word is the language; no, otherwise.
 - Given $w \in \Sigma^*$ and $L \subseteq \Sigma^*$, does $w \in L$?

22c:131 Limits of Computation

Hantao Zhang

<http://www.cs.uiowa.edu/~hzhang/c131>

The University of Iowa
Department of Computer Science

Formal Definition of Finite Automata

A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set called the *states*
- Σ is a finite set called the *alphabet*
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function
- $q_0 \in Q$ is the *start state* also called *initial state*
- $F \subseteq Q$ is the set of *accept states*, also called the *final states*

Finite Automata

- The simplest computational model is called a *finite state machine* or a *finite automaton*
- Representations:
 - Graphical
 - Tabular
 - Mathematical.

Computation of a Finite Automaton

- The automaton receives the input symbols one by one from left to right
- After reading each symbol, M_1 moves from one state to another along the transition that has that symbol as its label
- When M_1 reads the last symbol of the input it produces the output: *accept* if M_1 is in an accept state, or *reject* if M_1 is not in an accept state

Applications

- Finite automata are popular in parser construction of compilers.
- Finite automata and their probabilistic counterparts, *Markov chain*, are useful tools for pattern recognition used in speech processing and optical character recognition
- Markov chains have been used to model and predict price changes in financial applications

Formal Definition of Acceptance

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and $w = a_1 a_2 \dots a_n$ be a string over Σ .

Then M *accepts* w if a sequence of states r_0, r_1, \dots, r_n exist in Q such that the following hold:

1. $r_0 = q_0$
2. $\delta(r_i, a_{i+1}) = r_{i+1}$ for $i = 0, 1, \dots, n - 1$
3. $r_n \in F$

- Condition (1) says where the machine starts.
- Condition (2) says that the machine goes from state to state according to its transition function δ .
- Condition (3) says when the machine accepts its input: if it ends up in an accept state.

Language of an Automaton

- If L is the set of all strings that a finite automaton M accepts, we say that L is the *language of the machine* M and write $L(M) = L$.
- An automaton may accept several strings, but it always recognizes only one language
- If a machine accepts no strings, it still recognizes one language, namely the empty language \emptyset

Designing Finite Automata

- Whether it be of automaton or artwork, design is a creative process. Consequently it cannot be reduced to a simple recipe or formula.
- The approach:
 - Identify the finite pieces of information you need to solve the problem. These are the *states*.
 - Identify the condition (alphabet) to change from one state to another.
 - Identify the start and final states.
 - Add missing transitions.

Regular Languages

We say that a finite automaton M recognizes the language L if $L = \{w \mid M \text{ accepts } w\}$

Definition A language is called *regular language* if there exists a finite automaton that recognizes it.

Regular Expressions

- Three base cases:
 - \emptyset is a regular expression denoting the language \emptyset ;
 - ϵ is a regular expression denoting the language $\{\epsilon\}$;
 - For any $a \in \Sigma$, a is a regular expression denoting the language $\{a\}$;
- Three recursive cases: If r_1 and r_2 are regular expressions denoting languages L_1 and L_2 , respectively, then
 - **Union:** $r_1 \cup r_2$ denotes $L_1 \cup L_2$;
 - **Concatenation:** $r_1 r_2$ denotes $L_1 L_2$;
 - **Star:** r_1^* denotes L_1^* .

Operations on Regular Languages

Let A and B be languages. We define regular operations *union*, *concatenation*, and *star* as follows:

- **Union:** $A \cup B = \{x \mid x \in A \vee x \in B\}$
- **Concatenation:** $A \circ B = \{xy \mid x \in A \wedge y \in B\}$
- **Star:** $A^* = \{x_1 x_2 \dots x_k \mid k \geq 0 \wedge x_i \in A, 1 \leq i \leq k\}$

Note:

 - (1) Because “any number” includes 0, $\epsilon \in A^*$, no matter what A is.
 - (2) A^+ denotes $A \circ A^*$.

Closure under Complementation

- **Theorem** That class of regular languages is closed under complementation.
- **Proof** For that we will first show that if M is a DFA that recognizes a language B , swapping the accept and non-accept states in M yields a new DFA that recognizes the complement of B .

Closure under Intersection

- **Theorem** That class of regular languages is closed under intersection.
- **Proof** Use cross-product construction of states.

Two ways to Introduce Nondeterminism

- More choices for the next state: Zero, one, or many arrows may exit from each state.
- A state may change to the next state without spending an input symbol: ϵ -transitions.

Nondeterminism

- So far in our discussion, every step of a finite automaton computation follows in a unique way from the preceding step.
- We call this a *deterministic computation*. In a *nondeterministic* computation, choices may exist for the next state at any point.
- Nondeterminism is a generalization of determinism; hence, every finite automaton is a nondeterministic finite automaton (NFA).

Tree Computation of NFA

A way to think of a nondeterministic computation is as a tree of possibilities

- The root of the tree corresponds to the start of the computation
- Every branching point in the tree corresponds to a point in the computation at which the machine has multiple choices
- The machine accepts if at least one of the computation branches ends in an accept state.

Formal Definition of NFA

- An NFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:
 - Q is a finite set called the *states*
 - Σ is a finite set called the *alphabet*
 - $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$
where $\mathcal{P}(Q)$ is the power set of Q .
 - $q_0 \in Q$ is the *start state* also called *initial state*
 - $F \subseteq Q$ is the set of *accept states*, also called the *final states*
- In a DFA transition function is $\delta : Q \times \Sigma \rightarrow Q$.
- **notation:** For any alphabet Σ , $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$

Computation by an NFA

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and w a string over Σ . We say that N accepts w if $w = a_1 a_2 \dots a_m$, $a_i \in \Sigma_\epsilon$, $1 \leq i \leq m$, and a sequence of states r_0, r_1, \dots, r_m exists in Q such that:

1. $r_0 = q_0$
2. $r_{i+1} \in \delta(r_i, a_{i+1})$, for $i = 0, 1, \dots, m - 1$
3. $r_m \in F$

- Condition 1 says the machine's starting state.
- Condition 2 says that state r_{i+1} is one of the allowable new states when N is in state r_i and reads a_{i+1} . Note that $\delta(r_i, a_{i+1})$ is a set
- Condition 3 says that the machine accepts the input if the last state is in the accept state set.

Why NFA?

- Constructing NFA is sometimes easier than constructing DFA
- An NFA may be much smaller than a DFA that performs the same task.
- Computation of NFA is usually more expensive than that of DFA.
- Every NFA can be converted into an equivalent DFA.
- NFA provides good introduction to nondeterminism in more powerful computational models.

Application of NFA

- Now we use the NFA to show that collection of regular languages is closed under regular operations union, concatenation, and star.
- **Theorem 1.25, 1.45** The class of regular languages is closed under the union operation.
- **Theorem 1.26, 1.47** The class of regular languages is closed under concatenation operation.
- **Theorem 1.49** The class of regular languages is closed under star operation.

Equivalence of NFA and DFA

Theorem Every NFA automation has an equivalent DFA to recognize the same language.

- DFAs and NFAs recognize the same class of languages
- This equivalence is both surprising and useful.
 - It is surprising because NFAs appears to have more power than DFA, so we might expect that NFA recognizes more languages
 - It is useful because describing an NFA for a given language sometimes is much easier than describing a DFA

Equivalence of NFA and DFA

The Formal Proof

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing the language A . We construct the DFA M recognizing A .

- Before doing the full construction, consider first the easier case when N has no ϵ transitions.

Question

Is the class of languages recognized by NFAs closed under complementation?

Corollary 1.40

A language is regular iff some NFA recognizes it

Proof:

- If a language A is recognized by an NFA then A is recognized by the DFA equivalent, hence, A is regular
- If a language A is regular, it means that it is recognized by a DFA. But any DFA is also an NFA, hence, the language is recognized by an NFA

Equivalence of NFA and DFA

The Formal Proof

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing the language A . We construct the DFA M recognizing A .

- Before doing the full construction, consider first the easier case when N has no ϵ transitions.
- Then we consider the ϵ transitions.
- **Notation:** For any $R \subseteq Q$ define $E(R)$ to be the collection of states that can be reached from R by going only along ϵ transitions, including the members of R themselves. Formally:

$$E(R) = R \cup \{q \in Q \mid \exists r_1 \in R, r_2, \dots, r_k \in Q, r_{i+1} \in \delta(r_i, \epsilon), r_k = q\}$$