CS112: Theory of Computation (LFA)

Lecture5: Regular Expressions

Dumitru Bogdan

Faculty of Computer Science
University of Bucharest

March 18, 2021

Table of contents

Section 1

Previously on CS112

DFA

Definition

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

1. Q is a finite set called the states
2. Σ is a finite set called the alphabet
3. $\delta : Q \times \Sigma \to Q$ is the transition function
4. $q_0 \in Q$ is the start state
5. $F \subseteq Q$ is the set of accept states

Regular Language

Definition

A language is called a regular language if some finite automaton recognizes it.

Regular operations

Definition

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

- Union: $A \cup B = \{x | x \in A \text{ or } x \in B\}$
- Concatenation: $A \circ B = \{xy | x \in A \text{ and } y \in B\}$
- Star: $A^* = \{x_1 x_2 \dots x_k | k \geq 0 \text{ and each } x_i \in A\}$

Closure under union

Theorem

The class of regular languages is closed under the union operation, meaning that if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Formal definition

Definition

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

1. Q is a finite set of states
2. Σ is a finite alphabet
3. $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ is the transition function
4. $q_0 \in Q$ is the start state
5. $F \subseteq Q$ is the set of accepted states

We denote $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\mathcal{P}(Q)$ as the power set of Q.

Equivalence of NFAs and DFAs

Theorem

Every NFA has an equivalent DFA.

Equivalence of NFAs and DFAs

Corollary

A language is regular if and only if some NFA recognizes it.

DFA minimization

Definition

Let A be a DFA. We say that $w \in \Sigma^*$ **distinguishes** between two states $q_1, q_2 \in Q$ if either $\delta(q_1, w) \in F$ and $\delta(q_2, w) \notin F$ or $\delta(q_1, w) \notin F$ and $\delta(q_2, w) \in F$

Definition

Two states $q_1, q_2 \in Q$ are called **distinguishable** iff there is a word that distinguishes between them. States that are indistinguishable will also be sometimes called **equivalent**

Minimization Algorithm

The reasoning above leads to the following method:

- Start with an DFA A without unreachable states
- If A has distinguishable states q_1, q_2, combine them into one state. (For instance, remove q_1 and reroute all transitions into q_1 to go into q_2 instead)
- Repeat this process until no more distinguishable states can be found. At this point we will not be able to reduce A further

Minimization Algorithm

1. Remove unreachable states
2. Mark the distinguishable pairs of states
   - To achieve this task, we first mark all pairs p, q, where $p \in F$ and $q \notin F$ as distinguishable.
   - Then, we proceed as follows:

```
repeat
    for all non—marked pairs [p,q] do
        for each letter a do
            if the pair [d(p,a), d(q,a))] is marked
                then mark [p,q]
until no new pairs are marked
```

Section 2

Context setup

Context setup

Corresponding to Sipser 1.3

Context setup

In arithmetic, we can use the operations $+$ and $\times$ to build up expressions such as

$$(5 + 3) \times 4$$

Similarly, we can use the regular operations to build up expressions describing languages, which are called **regular expressions**. An example is:

$$(0 \cup 1)0^*$$

While the value of the arithmetic expression is the number 32. The value of a regular expression is a language. In this case, the value is the language consisting of all strings starting with a 0 or a 1 followed by any number of 0s.

Context setup

$$(0 \cup 1)0^*$$

- The value of a regular expression is a language. In this case, the value is the language consisting of all strings starting with a 0 or a 1 followed by any number of 0
- First, the symbols 0 and 1 are shorthand for the sets $\{0\}$ and $\{1\}$. So $(0 \cup 1)$ means $(\{0\} \cup \{1\})$. The value of this part is the language $\{0, 1\}$
- The part 0^* means $\{0\}^*$ and its value is the language consisting of all strings containing any number of 0s
- Second, like the $\times$ symbol in algebra, the concatenation symbol $\circ$ often is implicit in regular expressions
- Thus $(0 \cup 1)0^*$ actually is shorthand for $(0 \cup 1) \circ 0^*$. The concatenation attaches the strings from the two parts to obtain the value of the entire expression

Context setup

The regular expression

$$(0 \cup 1)0^*$$

is in fact:

$$(\{0\} \cup \{1\}) \circ \{0\}^*$$

Context setup

Regular expressions have an important role in computer science applications.

- In applications involving text, users may want to search for strings that satisfy certain patterns. Regular expressions provide a powerful method for describing such patterns
- Utilities such as grep in UNIX, modern programming languages such as Python, and text editors all provide mechanisms for the description of patterns by using regular expressions

Example

Let's take:

$$(0 \cup 1)^*$$

- It starts with the language $(0 \cup 1)$ and applies the $*$ operation
- The value of this expression is the language consisting of all possible strings of 0s and 1s.

Section 3

Formal definition

Formal definition

Definition

We say that R is a regular expression if R is:

1. a for some a in the alphabet Σ
2. ϵ
3. $\emptyset$
4. $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions
5. $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions, or
6. (R_1^*) where R_1 is a regular expression

In items 1 and 2, the regular expressions a and ϵ represent the languages $\{a\}$ and $\{\epsilon\}$, respectively. In item 3, the regular expression $\emptyset$ represents the empty language. In items 4, 5, and 6, the expressions represent the languages obtained by taking the union or concatenation of the languages R_1 and R_2, or the star of the language R_1, respectively.

Formal definition

Definition

We say that R is a regular expression if R is:

1. a for some a in the alphabet Σ
2. ϵ
3. $\emptyset$
4. $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions
5. $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions, or
6. (R_1^*) where R_1 is a regular expression

Important

Don't confuse the regular expressions ϵ and $\emptyset$. The expression ϵ represents the language containing a single string—namely, the empty string—whereas $\emptyset$ represents the language that doesn't contain any strings

Few remarks

- Are we in danger of a circular definition?
- No, because R_1 and R_2 are always smaller than R
- A definition of this type is called an **inductive definition**
- Parentheses in an expression may be omitted. If they are, **evaluation is done in the precedence order: star, then concatenation, then union**
- We use R^+ be shorthand for RR^* or we can write $R^+ \cup \epsilon = R^*$
- In other words, whereas R^* has all strings that are 0 or more concatenations of strings from R, the language R^+ has all strings that are 1 or more concatenations of strings from R
- In addition, we let R^k be shorthand for the concatenation of k R's with each other
- When we want to distinguish between a regular expression R and the language that it describes, we write $L(R)$ to be the language of R

Examples

We assume $\Sigma = \{0, 1\}$:

1. $0^*10^* = \{w | \text{w contains a single 1}\}$
2. $\Sigma^*1\Sigma^* = \{w | \text{w hast at least one 1}\}$
3. $\Sigma^*001\Sigma^* = \{w | \text{w contains the string 001 as substring}\}$
4. $1^*(01^+)^* = \{w | \text{every 0 in w is followed by at least one 1}\}$
5. $(\Sigma\Sigma)^* = \{w | \text{w is a string of even length}\}$
6. $(\Sigma\Sigma\Sigma)^* = \{w | \text{w is a string with length multiple of 3}\}$
7. $01 \cup 10 = \{01, 10\}$
8. $(0 \cup \epsilon)1^* = 01^* \cup 1^*$
9. $1^*\emptyset = \emptyset$

More remarks

Let R be any regular expression. Then we have the following identities:

- $R \cup \emptyset = R$. Adding the empty language to any other language will not change it
  $R \circ \epsilon = R$, Joining the empty string to any string will not change it

If we replace ϵ with $\emptyset$ then the preceding identities may not work:

- If $R = 0$, then $L(R) = \{0\}$ but $L(R \cup \epsilon) = \{0, \epsilon\}$. So $R \neq R \cup \epsilon$ in this case
- If $R = 0$, then $L(R) = \{0\}$ but $L(R \circ \emptyset) = \emptyset$. So $R \neq R \circ \emptyset$ in this case

Section 4

Equivalence with finite automata

Few remarks

- Regular expressions and finite automata are equivalent in their descriptive power even if they are different
- Any regular expression can be converted into a finite automaton that recognizes the language it describes, and vice versa
- Recall that a regular language is one that is recognized by some finite automaton

Equivalence with finite automata

Theorem

A language is regular if and only if some regular expression describes it.

Proof.

Two directions proof. We state and prove each direction as a separate lemma □

Equivalence with finite automata

Lemma

If a language is described by a regular expression, then it is regular ($\Leftarrow$).

Lemma

If a language is regular, then it is described by a regular expression ($\Rightarrow$).

Equivalence with finite automata

Lemma

If a language is described by a regular expression, then it is regular ($\Leftarrow$).

Proof idea: Let's say that we have a regular expression R describing some language A. We show how to convert R into an *NFA* recognizing A. And we know that if a *NFA* recognize A then A is regular

Equivalence with finite automata

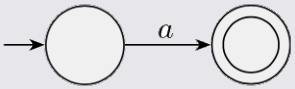
Lemma

If a language is described by a regular expression, then it is regular ($\Leftarrow$).

Proof.

Let us convert R into a *NFA* N. We do this by considering the six cases in the formal definition of regular expressions.

1. $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$ (language recognized by R). The below NFA recognizes $L(R)$



Note that this machine fits the definition of an NFA but not that of a DFA because it has some states with no exiting arrow for each possible input symbol. Of course, we could have presented an equivalent DFA here; but an NFA is all we need for now, and it is easier to

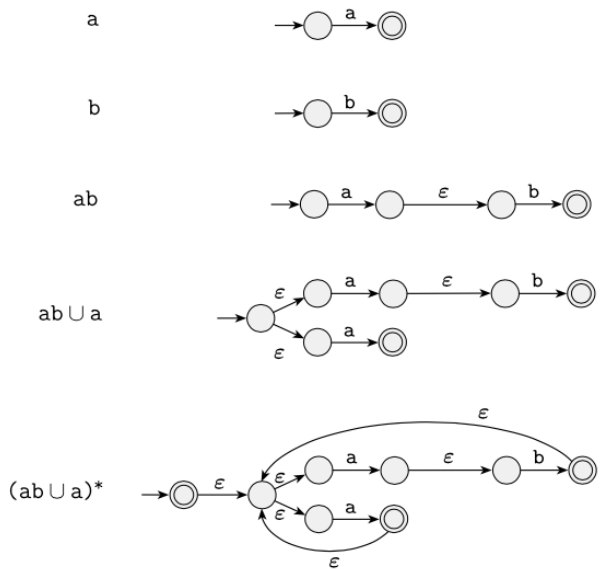
Examples

That ends the first part of the proof of theorem, giving the easier direction of the if and only if condition. Before going on to the other direction, let's consider some examples whereby we use this procedure to convert a regular expression to an NFA.

Examples

- Let's convert the regular expression $(ab \cup a)^*$ to a *NFA*
- We build up from the smallest subexpressions to larger subexpressions until we have an NFA for the original expression

Example: $(ab \cup a)^*$



Example: $(ab \cup a)^*$

- Note that this procedure generally doesn't give the NFA with the fewest states.
- In this example, the procedure gives an NFA with eight states, but the smallest equivalent NFA has only two states. Can you find it? ($\Leftarrow$ first to find it will get a CS112 T-shirt)

Examples

- Let's convert the regular expression $(a \cup b)^* aba$ to a *NFA*
- We skip few minor steps

Example: $(a \cup b)^* aba$