CIS 770: Formal Language Theory

Pavithra Prabhakar

Kansas State University

Spring 2015

Operations on Languages

- Recall: A language is a set of strings
- We can consider new languages derived from operations on given languages
    - e.g., $L_1 \cup L_2$, $L_1 \cap L_2$, $\frac{1}{2}L$, ...
- A simple but powerful collection of operations:
    - Union, Concatenation and Kleene Closure

Concatenation of Languages

Definition

Given languages L_1 and L_2, we define their *concatenation* to be the language $L_1 \circ L_2 = \{xy \mid x \in L_1,\ y \in L_2\}$

Example

- $L_1 = \{\text{hello}\}$ and $L_2 = \{\text{world}\}$ then $L_1 \circ L_2 = \{\text{helloworld}\}$
- $L_1 = \{00, 10\}$; $L_2 = \{0, 1\}$. $L_1 \circ L_2 = \{000, 001, 100, 101\}$
- $L_1 =$ set of strings ending in 0; $L_2 =$ set of strings beginning with 01. $L_1 \circ L_2 =$ set of strings containing 001 as a substring
- $L \circ \{\epsilon\} = L$. $L \circ \emptyset = \emptyset$.

Kleene Closure

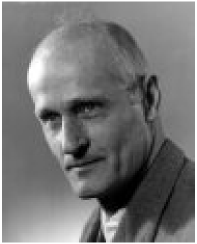
Definition

$$L^n = \begin{cases} \{\epsilon\} & \text{if } n = 0 \\ L^{n-1} \circ L & \text{otherwise} \end{cases} \qquad L^* = \bigcup_{i \geq 0} L^i$$

i.e., L^i is $L \circ L \circ \cdots \circ L$ (concatenation of i copies of L), for $i > 0$.
L^*, the Kleene Closure of L: set of strings formed by taking any number of strings (possibly none) from L, possibly with repetitions and concatenating all of them.

- If $L = \{0, 1\}$, then $L^0 = \{\epsilon\}$, $L^2 = \{00, 01, 10, 11\}$. $L^* =$ set of *all* binary strings (including ϵ).
- $\emptyset^0 = \{\epsilon\}$. For $i > 0$, $\emptyset^i = \emptyset$. $\emptyset^* = \{\epsilon\}$
- $\emptyset$ is one of only two languages whose Kleene closure is finite. Which is the other? $\{\epsilon\}^* = \{\epsilon\}$.

Regular Expressions
A Simple Programming Language



Stephen Cole Kleene

A regular expression is a formula for representing a (complex) language in terms of "elementary" languages combined using the three operations union, concatenation and Kleene closure.

Syntax and Semantics

A regular expression over an alphabet Σ is of one of the following forms:

	Syntax	Semantics
Basis	$\emptyset$	$L(\emptyset) = \{\}$
	ϵ	$L(\epsilon) = \{\epsilon\}$
	a	$L(a) = \{a\}$
Induction	$(R_1 \cup R_2)$	$L((R_1 \cup R_2)) = L(R_1) \cup L(R_2)$
	$(R_1 \circ R_2)$	$L((R_1 \circ R_2)) = L(R_1) \circ L(R_2)$
	(R_1^*)	$L((R_1^*)) = L(R_1)^*$

Notational Conventions

Removing the brackets

To avoid cluttering of parenthesis, we adopt the following conventions.

- Precedence: $*, \circ, \cup$. For example, $R \cup S^* \circ T$ means $(R \cup ((S^*) \circ T))$
- Associativity: $(R \cup (S \cup T)) = ((R \cup S) \cup T) = R \cup S \cup T$ and $(R \circ (S \circ T)) = ((R \circ S) \circ T) = R \circ S \circ T$.

Also will sometimes omit $\circ$: e.g. will write RS instead of $R \circ S$

Regular Expression Examples

R	$L(R)$
$(0 \cup 1)^*$	$= (\{0\} \cup \{1\})^* = \{0,1\}^*$
$0\emptyset$	$\emptyset$
$0^* \cup (0^*10^*10^*10^*)^*$	Strings where the number of 1s is divisible by 3
$(0 \cup 1)^*001(0 \cup 1)^*$	Strings that have 001 as a substring

More Examples

R	$L(R)$
$(10)^* \cup (01)^* \cup 0(10)^* \cup 1(01)^*$	Strings that consist of alternating 0s and 1s
$(\epsilon \cup 1)(01)^*(\epsilon \cup 0)$	Strings that consist of alternating 0s and 1s
$(0 \cup \epsilon)(1 \cup 10)^*$	Strings that do not have two consecutive 0s

Some Regular Expression Identities

We say $R_1 = R_2$ if $L(R_1) = L(R_2)$.

- Commutativity: $R_1 \cup R_2 = R_2 \cup R_1$ (but $R_1 \circ R_2 \neq R_2 \circ R_1$ typically)
- Associativity: $(R_1 \cup R_2) \cup R_3 = R_1 \cup (R_2 \cup R_3)$ and $(R_1 \circ R_2) \circ R_3 = R_1 \circ (R_2 \circ R_3)$
- Distributivity: $R \circ (R_1 \cup R_2) = R \circ R_1 \cup R \circ R_2$ and $(R_1 \cup R_2) \circ R = R_1 \circ R \cup R_2 \circ R$
- Concatenating with ϵ: $R \circ \epsilon = \epsilon \circ R = R$
- Concatenating with $\emptyset$: $R \circ \emptyset = \emptyset \circ R = \emptyset$
- $R \cup \emptyset = R$. $R \cup \epsilon = R$ iff $\epsilon \in L(R)$
- $(R^*)^* = R^*$
- $\emptyset^* = \epsilon$

Useful Notation

Definition

Define $R^+ = RR^*$. Thus, $R^* = R^+ \cup \epsilon$. In addition, $R^+ = R^*$ iff $\epsilon \in L(R)$.

Regular Expressions and Regular Languages

Why do they have such similar names?

Theorem

L is a regular language if and only if there is a regular expression R such that $L(R) = L$

i.e., Regular expressions have the same "expressive power" as finite automata.

Proof.

- Given regular expression R, will construct NFA N such that $L(N) = L(R)$
- Given DFA M, will construct regular expression R such that $L(M) = L(R)$ □

Regular Expressions to Finite Automata

... to Non-determinstic Finite Automata

Lemma

For any regex R, there is an NFA N_R s.t. $L(N_R) = L(R)$.

Proof Idea

We will build the NFA N_R for R, inductively, based on the number of operators in R, $\#(R)$.

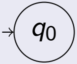
- Base Case: $\#(R) = 0$ means that R is $\emptyset, \epsilon$, or a (from some $a \in \Sigma$). We will build NFAs for these cases.
- Induction Hypothesis: Assume that for regular expressions R, with $\#(R) \leq n$, there is an NFA N_R s.t. $L(N_R) = L(R)$.
- Induction Step: Consider R with $\#(R) = n + 1$. Based on the form of R, the NFA N_R will be built using the induction hypothesis.

Regular Expression to NFA

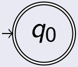
Base Cases

If R is an elementary regular expression, NFA N_R is constructed as follows.

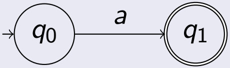
$R = \emptyset$



$R = \epsilon$



$R = a$



Case $R = R_1 \cup R_2$

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ (with $Q_1 \cap Q_2 = \emptyset$) such that $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$. The NFA $N = (Q, \Sigma, \delta, q_0, F)$ is given by

- $Q = Q_1 \cup Q_2 \cup \{q_0\}$, where $q_0 \notin Q_1 \cup Q_2$
- $F = F_1 \cup F_2$
- δ is defined as follows

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{if } q \in Q_1 \\ \delta_2(q, a) & \text{if } q \in Q_2 \\ \{q_1, q_2\} & \text{if } q = q_0 \text{ and } a = \epsilon \\ \emptyset & \text{otherwise} \end{cases}$$

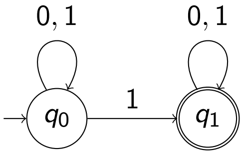
Induction Step: Union

Correctness Proof

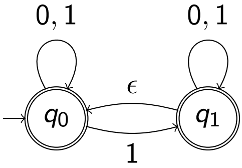
Need to show that $w \in L(N)$ iff $w \in L(N_1) \cup L(N_2)$.

- $\Rightarrow$ $w \in L(N)$ implies $q_0 \xrightarrow{w}_N q$ for some $q \in F$. Based on the transitions out of q_0, $q_0 \xrightarrow{\epsilon}_N q_1 \xrightarrow{w}_N q$ or $q_0 \xrightarrow{\epsilon}_N q_2 \xrightarrow{w}_N q$. Consider $q_0 \xrightarrow{\epsilon}_N q_1 \xrightarrow{w}_N q$. (Other case is similar) This means $q_1 \xrightarrow{w}_{N_1} q$ (as N has the same transition as N_1 on the states in Q_1) and $q \in F_1$. This means $w \in L(N_1)$.
- $\Leftarrow$ $w \in L(N_1) \cup L(N_2)$. Consider $w \in L(N_1)$; case of $w \in L(N_2)$ is similar. Then, $q_1 \xrightarrow{w}_{N_1} q$ for some $q \in F_1$. Thus, $q_0 \xrightarrow{\epsilon}_N q_1 \xrightarrow{w}_N q$, and $q \in F$. This means that $w \in L(N)$.

Example demonstrating the problem



Example NFA N



Incorrect Kleene Closure of N

$L(N) = (0 \cup 1)^*1(0 \cup 1)^*$. Thus, $(L(N))^* = \epsilon \cup (0 \cup 1)^*1(0 \cup 1)^*$. The previous construction, gives an NFA that accepts $0 \notin (L(N))^*$!

Regular Expressions to NFA

To Summarize

We built an NFA N_R for each regular expression R inductively

- When R was an elementary regular expression, we gave an explicit construction of an NFA recognizing $L(R)$
- When $R = R_1 \mathrm{op} R_2$ (or $R = \mathrm{op}(R_1)$), we constructed an NFA N for R, using the NFAs for R_1 and R_2.

Regular Expressions to NFA

An Example

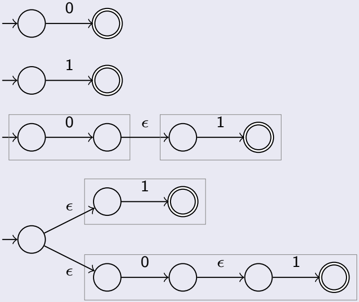
Build NFA for $(1 \cup 01)^*$

N_0

N_1

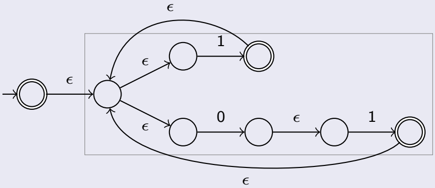
N_{01}

$N_{1 \cup 01}$



Example Continued

Build NFA for $(1 \cup 01)^*$



$N_{(1 \cup 01)^*}$

Today

- Defined Regular Expressions
  - Syntax: what a regex is built out of — $\emptyset$, ϵ, characters in Σ, and operators $\cup, \circ, ^*$.
  - Semantics: what language a regex stands for.
- Expressive power of regular expressions: can express (any and only) regular languages
  - Today: Languages represented by regular expressions are regular (we showed how to build NFAs for them).
  - Coming up: Regular languages can be represented by regular expressions (by building regex for any given DFA).