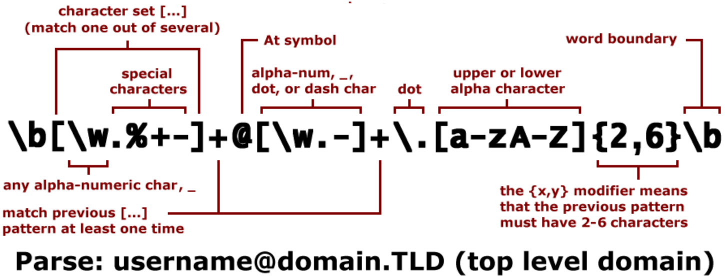
Automata, Languages and Computation

Chapter 3 : Regular Expressions

Master Degree in Computer Engineering
University of Padua
Lecturer : Giorgio Satta

Lecture based on material originally developed by :
Gösta Grahne, Concordia University

Regular Expressions



1. Regular Expressions : a declarative formalism for regular languages

2. FA and regular expressions : the two language classes are the same

3. Algebraic laws for regular expressions

Introduction

A FA (NFA or DFA) is a "blueprint" for **constructing** a machine recognizing a regular language

Though a FA can be easily implemented, it is often difficult to interpret its meaning

A regular expression is a "user-friendly," **declarative** way of describing a regular language

Introduction

Example : $\mathbf{01}^* + \mathbf{10}^*$ denotes all binary strings that

- start with 0 followed by zero or more 1's; or
- start with 1 followed by zero or more 0's

Regular expressions are used in

- UNIX `grep` command
- Perl programming language
- pattern matching applications
- tools for automatic constructions of lexical analyzers

Operations on languages

Union : $L \cup M = \{w \mid w \in L \text{ or } w \in M\}$

Concatenation : $L.M = \{w \mid w = xy,\ x \in L,\ y \in M\}$

Note : dot operator often omitted

Note : $\varnothing.L = L.\varnothing = \varnothing$

Powers :

- $L^0 = \{\epsilon\}$
- $L^k = L.L^{k-1}$, for $k \geqslant 1$

Kleene closure : $L^* = \bigcup_{i=0}^{\infty} L^i$

In mathematics operator '$*$' is also known as the free monoid construction.

Example

Let $L = \{0, 11\}$. In order to construct L^* :

- $L^0 = \{\epsilon\}$
- $L^1 = L = \{0, 11\}$
- $L^2 = L.L^1 = L.L = \{00, 011, 110, 1111\}$
- $L^3 = L.L^2 =$

$$\{000, 0011, 0110, 01111, 1100, 11011, 11110, 111111\}$$

Therefore

$$L^* = \{\epsilon, 0, 11, 00, 011, 110, 1111, 000,$$
$$0011, 0110, 01111, 1100, 11011, 11110, 111111, \ldots\}$$

Example

Construct $\emptyset^*$:

- $\emptyset^0 = \{\epsilon\}$
- $\emptyset^i = \emptyset$, for every $i \geqslant 1$

Therefore $\emptyset^* = \{\epsilon\}$

Operations on languages

Note : We have used the operator $*$ on alphabets (Σ^*); we now use the same operator with languages (L^*)

What happens when $\Sigma = L$?

- elements of Σ are symbols, while elements of L are strings
- the result is the same

Inductive definition of regular expressions

A **regular expression** E over alphabet Σ and the **generated** language $L(E)$ are recursively defined as follows

Base

- ϵ is a regular expression, and $L(\epsilon) = \{\epsilon\}$
- $\emptyset$ is a regular expression, and $L(\emptyset) = \emptyset$
- If $a \in \Sigma$, then $\boldsymbol{a}$ is a regular expression, and $L(\boldsymbol{a}) = \{a\}$

  Note the bold typesetting to distinguish the regular expression from the associated alphabet symbol

Inductive definition of regular expressions

Induction

- If E and F are regular expressions, then $E + F$ is a regular expression, and $L(E + F) = L(E) \cup L(F)$
- If E and F are regular expressions, then EF is a regular expression, and $L(EF) = L(E)L(F)$
- If E is a regular expressions, then E^* is a regular expression, and $L(E^*) = (L(E))^*$

  Note the overloading of the symbol '$*$'
- If E is a regular expressions, then (E) is a regular expression, and $L((E)) = L(E)$

Example

Specify a regular expression for

$$L = \{w \mid w \in \{0,1\}^*, \text{ no occurrence of 00 or 11 in } w\}$$

0 can only be followed by 1; 1 can only be followed by 0

Four cases, based on the choice of the start/end symbol

$$(\mathbf{01})^* + (\mathbf{10})^* + \mathbf{0}(\mathbf{10})^* + \mathbf{1}(\mathbf{01})^*$$

Equivalently, but in a more **compact** form

$$(\epsilon + \mathbf{1})(\mathbf{01})^*(\epsilon + \mathbf{0})$$

ϵ used to make other symbols optional

Operator's precedence

Precedence of operators (higher first)

- Kleene closure ($*$)
- concatenation (dot)
- union ($+$)

Example : $\mathbf{01}^* + \mathbf{1}$ means $(\mathbf{0}(\mathbf{1}^*)) + \mathbf{1}$

Use parentheses to force precedence

Structure of a regular expression

Each regular expression can be naturally associated with a **tree structure** representing its recursive definition

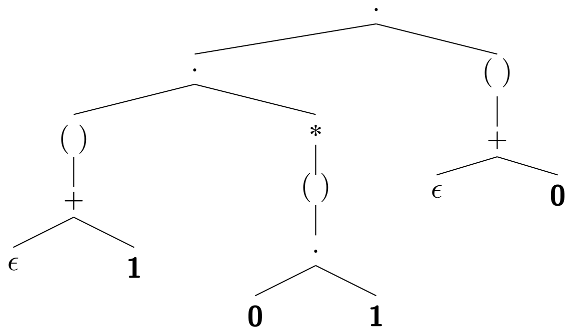
This will be used many times later, in proofs based on structural induction

To this end, we assume binary operators are **left associative**

Example : $\mathbf{010}$ means $((\mathbf{01})\mathbf{0})$

Structure of a regular expression

Example : The regular expression $(\epsilon + \mathbf{1})(\mathbf{01})^*(\epsilon + \mathbf{0})$ can be associated with the following tree



Test

Write regular expressions for the following languages

- strings over $\{a, b\}$ starting with a and ending with bb
- strings over $\{a, b\}$ with at least two occurrences of a
- strings over $\{0, 1\}$ with 1 in the seventh to last position
- strings over $\{0, 1, 2\}$ with zero or more 0's, or else (exclusive) with one or more 1's, or else (exclusive) with two or more 2's
- strings over $\{a, b\}$ with at least one occurrence of a and at least one occurrence of b

Test

Specify in words the languages generated by the following regular expressions, defined over $\Sigma = \{0, 1\}$

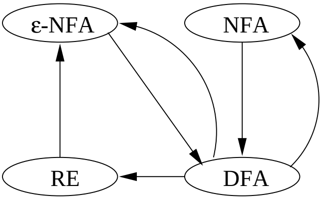
- $(\mathbf{0} + \mathbf{1})^*\mathbf{11}(\mathbf{0} + \mathbf{1})^*$
- $(\mathbf{1} + \epsilon)(\mathbf{00}^*\mathbf{1})\mathbf{0}^*$

Equivalence of FA and regular expressions

We have already shown that DFA, NFA, and ϵ-NFA are equivalent

To show that FA and regular expressions are equivalent, we will show that

- for each DFA A there is a regular expression R such that $L(R) = L(A)$
- for each regular expression R there is a ϵ-NFA A such that $L(A) = L(R)$



From DFA to regular expression

Theorem If $L = L(A)$ for some DFA A, then there exists a regular expression R such that $L(R) = L$

Proof

We construct R from A using the **state elimination** technique

Construction by state elimination

Based on the subsequent elimination of the states of the DFA, without altering the generated language

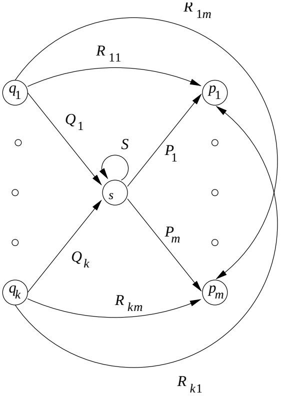
Initially

- transitions on symbol a are relabeled with the equivalent regular expressions $\boldsymbol{a}$
- in some cases: if there is no transition between pair p, q, we create a new transition $p \rightarrow q$ with label $\emptyset$

  p and q could be the same state

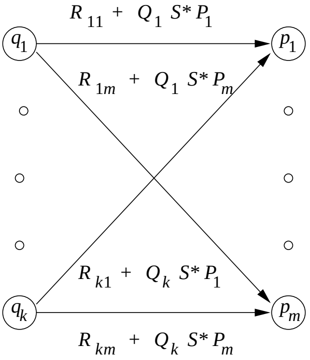
Construction by state elimination

States $q_1, \ldots, q_k$ are the **antecedents** of s and states $p_1, \ldots, p_m$ are the **successors** of s, assuming $s \neq q_i, p_j$; these two sets are not necessarily disjoint



Construction by state elimination

We can now eliminate state s



If antecedent or successor set is empty, we can eliminate s without adding arcs (R empty, Q or P empty)

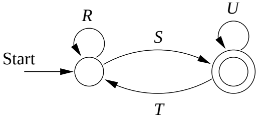
Construction by state elimination

Construction of the regular expression :

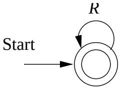
- for each final state q, we remove from the initial automaton all states except q_0 and q, resulting in an automaton A_q with at most two states
- we convert each automaton A_q to a regular expression E_q and combine with the union operator

Construction by state elimination

A_q can be in one of the two following forms :



corresponding to the regular expression $E_q = (R + SU^*T)^*SU^*$



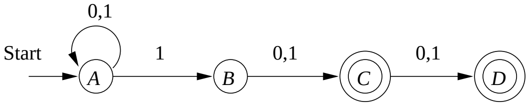
corresponding to the regular expression $E_q = R^*$

The final regular expression is then

$$\bigoplus_{q \in F} E_q$$

Example

The construction by state elimination works for every type of FA. Consider NFA M

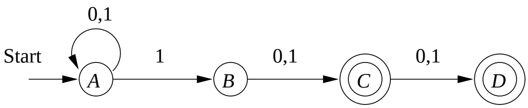


recognizing the language

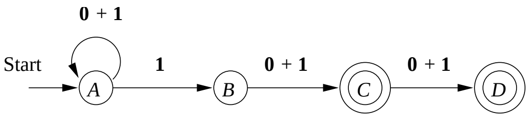
$$L(M) = \{w \mid w = x1b \text{ or } w = x1bc,\ x \in \{0,1\}^*,\ b, c \in \{0,1\}\}$$

Construct from M a regular expression generating $L(M)$

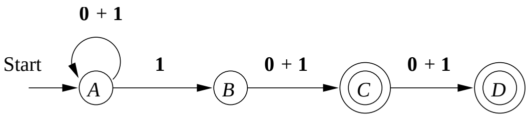
Example



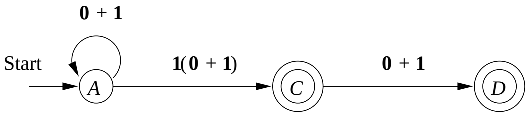
We transform M into an automaton with equivalent regular expressions at each transition



Example

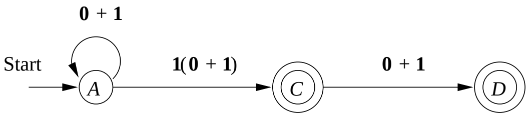


We eliminate state B

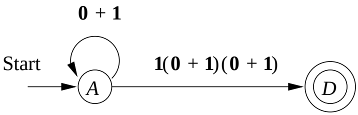


We have simplified the regular expression $\mathbf{1}\emptyset^*(\mathbf{0}+\mathbf{1})$ as $\mathbf{1}(\mathbf{0}+\mathbf{1})$, since $L(\emptyset^*) = \{\epsilon\}$

Example

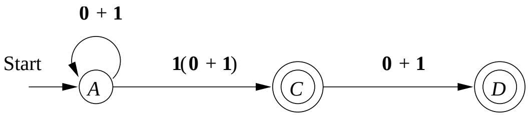


We eliminate state C resulting in M_D

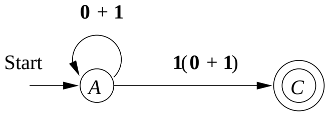


corresponding to the regular expression
$E_D = (\mathbf{0} + \mathbf{1})^*\mathbf{1}(\mathbf{0} + \mathbf{1})(\mathbf{0} + \mathbf{1})$

Example



We eliminate state D resulting in M_C



corresponding to the regular expression $E_C = (\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})$

The desired regular expression is the sum of E_D and E_C :

$$(\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})(\mathbf{0}+\mathbf{1}) + (\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})$$

Exercise

Write a regular expression for the language L over $\Sigma = \{0, 1, 2\}$ such that, for each string in L, the sum of its digits is an odd number

Suggestion

- start specifying a DFA that accepts L
- then construct the equivalent regular expression

From regular expression to ϵ-NFA

Theorem For every regular expression R we can construct an ϵ-NFA E such that $L(E) = L(R)$

Proof

We construct E with

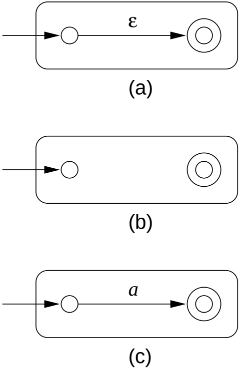
- only one final state
- no arc entering the initial state
- no arc exiting the final state

This will make it easier/safer to connect FAs

The construction uses structural induction

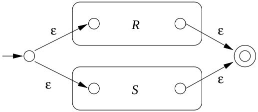
From regular expression to ϵ-NFA

Base Automata for regular expressions ϵ, $\emptyset$, and ***a***

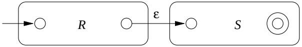


From regular expression to ϵ-NFA

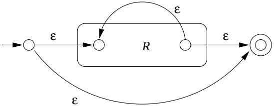
Induction Automata for $R + S$, RS, e R^*



(a)



(b)

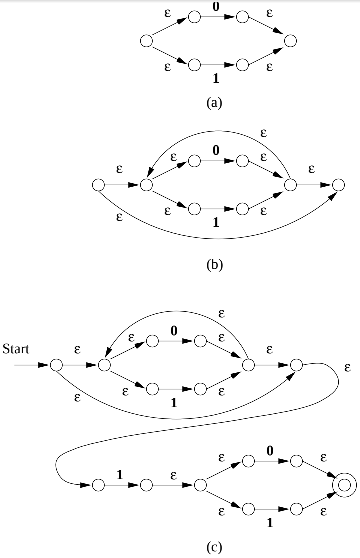


(c)

□

Example

Construct ϵ-NFA for the regular expression $(\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})$



Algebraic laws

There are some similarities between regular expressions and **arithmetic expressions**, if we consider the union as the sum and concatenation as the product

As for arithmetic expressions, there are similar properties for regular expressions (commutativity, distributivity, etc.)

There exists also **specific** properties for regular expressions, mainly related to Kleene's closure operator, which do not correspond to any laws of arithmetic

In the following slides, L, M, N are regular expressions, not languages

Commutativity and associativity

Union is **commutative** : $L + M = M + L$

Union is **associative** : $(L + M) + N = L + (M + N)$

Concatenation is **associative** : $(LM)N = L(MN)$

Concatenation is **not commutative** : there exist L and M such that $LM \neq ML$. **Example** : $\mathbf{10} \neq \mathbf{01}$

Identity and annihilators

Very useful in simplifying regular expressions :

$\emptyset$ is the **identity** for union: $\emptyset + L = L + \emptyset = L$

ϵ is the **left identity** and the **right identity** for concatenation : $\epsilon L = L\epsilon = L$

$\emptyset$ is the **left annihilator** and the **right annihilator** for concatenation : $\emptyset L = L\emptyset = \emptyset$

Distributivity and idempotence

Concatenation is **left distributive** over union :
$L(M + N) = LM + LN$

Concatenation is **right distributive** over union :
$(M + N)L = ML + NL$

Union is **idempotent** : $L + L = L$

Kleene closure & other operators

$(L^*)^* = L^*$ (proof in later slides)

$\emptyset^* = \epsilon$

$\epsilon^* = \epsilon$

$L^+ = LL^* = L^*L$

$L^* = L^+ + \epsilon$

$L? = \epsilon + L$

Exercise with solution

Prove that the regular expressions $(R^*)^*$ and R^* are equivalent

$L(R^*) = (L(R))^*$

$L((R^*)^*) = (L(R^*))^* = ((L(R))^*)^*$

Assuming $L(R) = L_R$, we need to show $(L_R^*)^* = L_R^*$

Exercise with solution

$$
\begin{aligned}
w \in (L_R^*)^* \quad & \Longleftrightarrow \quad w \in \bigcup_{i=0}^{\infty} \left(\bigcup_{j=0}^{\infty} L_R^j \right)^i \\
& \Longleftrightarrow \quad \exists k, m \in \mathbb{N} \ : \ w \in (L_R^m)^k \\
& \Longleftrightarrow \quad \exists p \in \mathbb{N} \ : \ w \in L_R^p \\
& \Longleftrightarrow \quad w \in \bigcup_{i=0}^{\infty} L_R^i \\
& \Longleftrightarrow \quad w \in L_R^*
\end{aligned}
$$

In the right to left direction, choose $k = 1$ and $m = p$