Regular Expressions

Greg Plaxton
Theory in Programming Practice, Fall 2005
Department of Computer Science
University of Texas at Austin

What is a Regular Expression?

- A regular expression defines a (possibly infinite) set of strings over a given alphabet
- Analogous to an arithmetic expression
  - The symbols of the alphabet are analogous to the numerical constants in an arithmetic expression
  - Instead of arithmetic operators such as addition, multiplication, and exponentiation, the operators are concatenation, union, and closure

Regular Expressions: Syntax

- The symbols $\emptyset$ (empty set), ϵ (empty string), and any symbol of the alphabet are regular expressions
- For any regular expressions p and q, (pq) (concatenation) and $(p \mid q)$ (union) are regular expressions
- For any regular expression p, p^* (Kleene closure) is a regular expression

Regular Expressions: Semantics

- The regular expression $\emptyset$ corresponds to the empty set of strings
- The regular expression ϵ corresponds to the set of strings $\{\epsilon\}$
- For any symbol a in the alphabet, the regular expression a corresponds to the set of strings $\{a\}$
- For any regular expressions p and q with corresponding sets of strings X and Y, the regular expression (pq) (resp., $(p \mid q)$) denotes the set of strings $\{xy \mid x \in X \wedge y \in Y\}$ (resp., $X \cup Y$)
- For any regular expression p with corresponding set of strings X, the regular expression p^* denotes the set of strings

$$\{x_1 x_2 \cdots x_k \mid k \geq 0 \wedge \langle \forall i : 1 \leq i \leq k : x_i \in X \rangle\}$$

Regular Expressions: Parenthesization

- When writing a regular expression, we generally try to omit as many parentheses as possible without altering the meaning of the expression
- Where parentheses are omitted, Kleene closure has the highest binding power, then concatenation, then union
  - Parentheses may be omitted whenever this convention yields the intended parenthesization
- Note that concatenation and union are associative
  - These facts often enable us to drop parentheses, e.g., we can write abc instead of $((ab)c)$

A Remark on Kleene Closure

- One can think of Kleene closure as follows:

$$p^* = \epsilon \mid p \mid pp \mid ppp \mid \ldots$$

- The RHS above is not a regular expression because it has an infinite number of terms
  - It is straightforward to prove by induction that every regular expression has a finite length
- The motivation for introducing the Kleene closure operator is to make the above RHS into a regular expression

Regular Expressions: Examples

- What is the set of strings corresponding to the regular expression $a \mid bc^*d$?
- It is often convenient to introduce identifiers to stand for certain regular expressions and then to use these identifiers as a shorthand for building up more complex regular expressions
  - $PosDigit = 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
  - $Digit = 0 \mid PosDigit$
  - $Natural = 0 \mid PosDigit\ Digit^*$
- The set of strings over the lowercase English alphabet containing all five vowels in order corresponds to the regular expression

$$(Letter^*)a(Letter^*)e(Letter^*)i(Letter^*)o(Letter^*)u(Letter^*)$$

where

$$Letter = a \mid b \mid c \mid \ldots \mid z$$

A More Elaborate Example

- For any binary string x, let $f(x)$ denote the nonnegative integer corresponding to x
  - Example: If $x = 00110$, then $f(x) = 6$
- Problem: Construct a regular expression corresponding to the set of all binary strings x such that $f(x)$ is a multiple of 3
  - We first inductively define the sets B_0, B_1, and B_2 of all binary strings x such that $f(x)$ is congruent to 0, 1, and 2, respectively, modulo 3
  - We then deduce a regular expression for B_0

Inductive Definition of Sets B_0, B_1, and B_2

(0) The empty string belongs to B_0

(1) For any binary string x in B_0, $x0$ belongs to B_0 and $x1$ belongs to B_1

(2) For any binary string x in B_1, $x0$ belongs to B_2 and $x1$ belongs to B_0

(3) For any binary string x in B_2, $x0$ belongs to B_1 and $x1$ belongs to B_2

Characterization of B_2 in Terms of B_1

- By (2) and (3), any binary string in B_2 is either of the form $x0$ where x belongs to B_1, or is of the form $x1$ where x belongs to B_2
- It follows that B_2 consists of all binary strings of the form $x01^*$ where x belongs to B_1

Characterization of B_1 in terms of B_0

- By (1), (3), and the preceding characterization of B_2, any binary string in B_1 is either of the form $x1$ where x belongs to B_0, or is of the form $x01^*0$ where x belongs to B_1
- It follows that B_1 consists of all binary strings of the form $x1(01^*0)^*$ where x belongs to B_0

Deducing a Regular Expression for B_0

- By (0), (1), (2), and the preceding characterization of B_1, the set B_0 consists of the empty string, all binary strings of the form $x0$ where x belongs to B_0, and all binary strings of the form $x1(01^*0)^*1$ where x belongs to B_0
- It follows that B_0 consists of all binary strings of the form

$$(0 \mid 1(01^*0)^*1)^*$$

Remark: Alternative View of the Preceding Example

- The binary strings in B_0 may be viewed as being generated by the grammar

$$\begin{aligned} S &\longrightarrow B_0 \\ B_0 &\longrightarrow \epsilon \mid B_0 0 \mid B_1 1 \\ B_1 &\longrightarrow B_0 1 \mid B_2 0 \\ B_2 &\longrightarrow B_1 0 \mid B_2 1 \end{aligned}$$

- As we have seen, the above grammar generates a regular language
- Not all grammars generate regular languages