Regular Expressions

Noman Amin

Lecturer, Dept. of CSE

Southeast University

Regular Expressions

Now, we switch our attention from machine-like descriptions of languages — deterministic and nondeterministic finite automata — to an algebraic description: the "regular expression." We shall find that regular expressions can define exactly the same languages that the various forms of automata describe: the regular languages. However, regular expressions offer something that automata do not: a declarative way to express the strings we want to accept. Thus, regular expressions serve as the input language for many systems that process strings.

Regular Expressions

Regular expressions denote languages. For a simple example, the regular expression $\mathbf{01}^* + \mathbf{10}^*$ denotes the language consisting of all strings that are either a single 0 followed by any number of 1's or a single 1 followed by any number of 0's.

Three Operations of Regular Expressions

Before describing the regular expression notation we need to learn the three operations on languages that the operators of regular expressions represent These operations are:

1. The *union* of two languages L and M, denoted $L \cup M$, is the set of strings that are in either L or M, or both. For example, if $L = \{001, 10, 111\}$ and $M = \{\epsilon, 001\}$, then $L \cup M = \{\epsilon, 10, 001, 111\}$.

2. The *concatenation* of languages L and M is the set of strings that can be formed by taking any string in L and concatenating it with any string in M. Recall Section 1.5.2, where we defined the concatenation of a

Three Operations of Regular Expressions

For example, if $L = \{001, 10, 111\}$ and $M = \{\epsilon, 001\}$, then $L.M$, or just LM, is $\{001, 10, 111, 001001, 10001, 111001\}$. The first three strings in LM are the strings in L concatenated with ϵ. Since ϵ is the identity for concatenation, the resulting strings are the same as the strings of L. However, the last three strings in LM are formed by taking each string in L and concatenating it with the second string in M, which is 001. For instance, 10 from L concatenated with 001 from M gives us 10001 for LM.

Three Operations of Regular Expressions

3. The *closure* (or *star*, or *Kleene closure*)[1] of a language L is denoted L^* and represents the set of those strings that can be formed by taking any number of strings from L, possibly with repetitions (i.e., the same string may be selected more than once) and concatenating all of them. For instance, if $L = \{0, 1\}$, then L^* is all strings of 0's and 1's. If $L = \{0, 11\}$, then L^* consists of those strings of 0's and 1's such that the 1's come in pairs, e.g., 011, 11110, and ϵ, but not 01011 or 101. More formally, L^* is the infinite union $\cup_{i \geq 0} L^i$, where $L^0 = \{\epsilon\}$, $L^1 = L$, and L^i, for $i > 1$ is $LL \cdots L$ (the concatenation of i copies of L).

Three Operations of Regular Expressions

Example : Since the idea of the closure of a language is somewhat tricky, let us study a few examples. First, let $L = \{0, 11\}$. $L^0 = \{\epsilon\}$, independent of what language L is; the 0th power represents the selection of zero strings from L. $L^1 = L$, which represents the choice of one string from L. Thus, the first two terms in the expansion of L^* give us $\{\epsilon, 0, 11\}$.

Next, consider L^2. We pick two strings from L, with repetitions allowed, so there are four choices. These four selections give us $L^2 = \{00, 011, 110, 1111\}$. Similarly, L^3 is the set of strings that may be formed by making three choices of the two strings in L and gives us

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

Regular Languages and Their Properties

A regular language is any language that can be described by a regular expression or recognized by a finite automaton. In the theory of computation, regular languages are the simplest class of languages and have the following key properties:

1. Closed Under Union: If L1 and L2 are regular languages, then L1 ∪ L2 is also regular.

2. Closed Under Intersection: If L1 and L2 are regular languages, then L1 ∩ L2 is also regular.

3. Closed Under Complementation: If L is a regular language, then its complement L' is also regular.

4. Closed Under Concatenation: If L1 and L2 are regular languages, then L1 . L2 is also regular.

5. Closed Under Kleene Star: If L is a regular language, then L* is also regular.

Regular Expressions and Finite Automata

The most important connection between regular expressions and automata theory is that regular expressions can be directly translated into finite automata and vice versa. Specifically:

A Deterministic Finite Automaton (DFA) and a Nondeterministic Finite Automaton (NFA) can recognize exactly the same set of languages that can be described by regular expressions, i.e., regular languages.

- **Conversion from Regular Expressions to Finite Automata:**
  - Thompson's Construction: This algorithm constructs an NFA from a regular expression.
  - Subset Construction: This method is used to convert an NFA to a DFA.
- **Conversion from Finite Automata to Regular Expressions:**
  - There are algorithms, such as the state elimination method, that can convert an NFA (or DFA) into an equivalent regular expression.

Examples

1. **<u>Regular expression: a*</u>**

   Description: This regular expression represents the language of strings that contain zero or more occurrences of the letter a. It includes the empty string ε, a, aa, aaa, etc.

   Language: {ε, a, aa, aaa, ...}

2. **<u>Regular expression: (a|b)*</u>**

   Description: This regular expression matches strings consisting of any combination of a and b, including the empty string.

   Language: {ε, a, b, ab, ba, aa, bb, aba, bab, ...}

3. **<u>Regular expression: a(b|c)*d</u>**

   Description: This regular expression matches any string that starts with a, followed by any number of b or c (including none), and ends with d.

   Language: {abd, acd, abbd, abcd, acbd, ...}

Example

Regular Expression: ab*

Explanation: This regular expression matches the string a followed by zero or more occurrences of the letter b.

Language: {a, ab, abb, abbb, ...}

Description: The string must begin with a, and the number of bs following a can be zero or more. The b* part indicates zero or more occurrences of b.

Example

<u>Regular Expression: (ab|cd)*</u>

Explanation: This regular expression matches zero or more occurrences of either the string ab or cd.

Language: {ε, ab, cd, abcd, ababcd, ...}

Description: The grouping (ab|cd) means that either ab or cd can repeat any number of times, including zero times (the empty string is also valid here due to the Kleene star).

Challenges and Limitations of Regular Expressions

While regular expressions are incredibly useful, they have limitations when it comes to expressing more complex languages:

1. **Non-Regular Languages:** Some languages cannot be described by regular expressions, such as the language {a^n b^n | n ≥ 0}, which requires a pushdown automaton (PDA) to be recognized, not a finite automaton.
2. **Limited Memory:** Regular expressions and finite automata have limited memory—they cannot "remember" past states beyond the finite set of states they have.
3. **Nested or Recursive Patterns:** Regular expressions are not capable of handling nested or recursive patterns that require memory beyond a finite state machine, such as matching balanced parentheses.

Precedence in Regular Expression

From highest to lowest precedence:

1. **Closure operators (Highest precedence):** These operators apply only to the immediate preceding symbol or group. **Example:** ab* Interpreted as a(b*) NOT: (ab)*
2. **Concatenation**
   - Implied operation (no symbol used)
   - Binds tighter than union but weaker than closure
   - **Example:** abc Interpreted as: ((a b) c)
3. **Union / Alternation (Lowest precedence)**: Symbol: | or + (depending on notation)
   - Example: a|bc Interpreted as: a | (bc) NOT: (a | b) c

Example

1. **Binary strings whose 10th symbol from the right is 1:** $(0+1)^*1(0+1)^9$
2. **Over Σ = {0,1}, all strings that start with 0:** $0(0+1)^*$
3. **Over Σ = {0,1}, all strings that end with 1:** $(0+1)^*1$
4. **Over Σ = {a,b}, a single symbol only:** $a + b$
5. **Over Σ = {a,b}, strings of length at least 1:** $(a+b)(a+b)^*$
6. **Over Σ = {0,1}, strings containing at least one 1:** $(0+1)^*1(0+1)^*$
7. **Over Σ = {a,b}, strings starting with a:** $a(a+b)^*$
8. **Over Σ = {0,1}, strings containing substring 01:** $(0+1)^*01(0+1)^*$

Example

1. **Over Σ = {0,1}, strings with no consecutive 1's:** (0+10)*(1+ ϵ)
2. **Over Σ = {0,1}, strings with no consecutive 0's:** (1+01)*(0 + ϵ)