Regular Expressions

CS236 - Discrete Structures
Instructor: Brett Decker
Spring 2021

Formal Languages

We use the term formal languages to distinguish the languages we will study from natural languages. Formal languages have precise rules and mathematical definitions. In order to start studying formal languages we need to introduce key definitions for the following terms: *symbol*, *string*, and *language*.

Language Definitions

A *symbol* is an element of a set.
A *string* is a sequence of symbols.
A *language* is a set of strings.
A *word* is a string of a language.
A *vocabulary* is a finite, non-empty set of symbols.

Definitions Example:

Given the following: $A = \{a, b, c, d, e, f\}$, A is a set that contains the symbols a, b, c, d, e, and f. Examples of strings created from these symbols are a, add, fed, abc, and $deeeee$. A language defines possible strings, or words, that are a part of the language. $L = \{aa, ab, ac, ad, ae, af\}$ is the language made up of words with length two, starting with the symbol a. Given this language, the vocabulary or V of L is the set $\{a, b, c, d, e, f\}$.

Language Example:

Given the following language L, what are the strings of L? What is the vocabulary (or set of symbols) for L?

$$L = \{brett, decker\}$$

There are two strings, or words in the language: *brett* and *decker* (not a very useful language, admittedly). The vocabulary, often denoted as V because it is a set, is $V = \{b, c, d, e, k, r, t\}$ (the order of the elements doesn't matter – here is it just in alphabetical order).

Recognizers and Generators

We begin our study of formal languages by discussing two core concepts: *recognizers* and *generators*. A *recognizer* is a construct that accepts a language. A *generator* is a construct that creates strings of a specific language. We will use *finite state machines* as the recognizers of the languages we will study. Our first generator will be *regular expressions*.

Regular Expression: Definition 1, Section 13.4*

The *regular expressions* over a set I are defined recursively by:

the symbol $\emptyset$ is a regular expression;
the symbol λ is a regular expression;
the symbol $\mathbf{x}$ is a regular expression whenever $x \in I$;
the symbols $(\mathbf{AB})$, $(\mathbf{A} \cup \mathbf{B})$, and $\mathbf{A}^*$ are regular expressions whenever $\mathbf{A}$ and $\mathbf{B}$ are regular expressions

Each regular expression represents a set specified by these rules:
$\emptyset$ represents the empty set, that is, the set with no strings;
λ represents the set $\{lambda\}$, which is the set containing the empty string;
$\mathbf{x}$ represents the set $\{x\}$ containing the string with one symbol x
$(\mathbf{AB})$ represents the concatenation of the sets represented by $\mathbf{A}$ and $\mathbf{B}$;
$(\mathbf{A} \cup \mathbf{B})$ represents the union of the sets represented by $\mathbf{A}$ and $\mathbf{B}$;
$\mathbf{A}^*$ represents the Kleene closure of the set represented by $\mathbf{A}$;

Concatenation: Definition 1, Section 13.3*

The *concatenation* of A and B, denoted by AB, is the set of all strings of the form xy, where x is a string in A and y is a string in B.

Kleene closure: Definition 2, Section 13.3*

The *Kleene closure* of A, denoted by A*, is the set consisting of concatenations of arbitrarily many strings from A.

Regular Expressions

In the above definition, the set I is called the input alphabet which contains the vocabulary of the language. The notation $x \in I$ is said "x in I" and means that x is an element of the set I. Each regular expression is said to *match* patterns of symbols. The symbol $\emptyset$ is called the "empty set" which is a set void of any symbols. The symbol λ, said "lambda," matches the empty string. The regular expression symbol x then denotes that we can use the symbol to match the corresponding element in the input alphabet. The symbols $(\mathbf{AB})$ denotes the concatenation of two regular expression. The symbols $(\mathbf{A} \cup \mathbf{B})$ denotes the union (either $\mathbf{A}$ or $\mathbf{B}$) of two regular expressions. The symbol $\mathbf{A}^*$ is called the Kleene (pronounced *klay-nee*) closure or Kleene star means the regular expression $\mathbf{A}$ is repeated zero or more times. Note

the infinite nature of the Kleene closure. It allows us to use regular expressions to describe languages with an infinite number of words. Let's do some examples to show how regular expressions generate languages.

Regular Expression Example:

Given the input alphabet I and the regular expressions below, what are the words of each generated language L_i?

$I = \{a, b, c\}$
1) Regular expression = aba
2) Regular expression = c^*
3) Regular expression = $(a \cup c)b$
4) Regular expression = $(ab) \cup c$
5) Regular expression = $a(b^*)a$

The first language generated by the first regular expression is $L_1 = \{aba\}$. Note that the regular expression states it will match words that start with an a, then have a b, and end with an a – that's only one possible word, which is why L_1 only contains one string. $L_2 = \{\lambda, c, cc, ccc, cccc, ccccc, \ldots\}$. Note that we cannot write all words generated by the second regular expression, because there are infinitely many. The regular expression states that it matches words of zero or more c's. We can use the $\ldots$ to state that the set is infinite (we just need to show the pattern first). $L_3 = \{ab, cb\}$. The regular expression matches any word that starts with either a or c and then ends with b. $L_4 = \{ab, c\}$. The regular expression matches any word that is the concatenation of a and b, or with a single c. $L_5 = \{aa, aba, abba, abbba, abbbba, \ldots\}$. Note again the infinite set created for the language with the presence of the Kleene closure.

Conclusion

Formal languages are the basis for the field of programming languages in Computer Science. Every programming language you have ever used is a formal language. As our study progresses you will how real programming languages are generated and recognized. See the book* for further examples and details.

*All definitions are from *Discrete Mathematics and Its Applications*, by Kenneth H. Rosen.