CMPSCI 250: Introduction to Computation

Lecture #19: Regular Expressions and Their Languages
David Mix Barrington
11 April 2013

Regular Expressions and Their Languages

- Alphabets, Strings and Languages
- Regular Expressions
- The Kleene Star Operation
- Finite Languages
- The Language (a + ab)*
- Logically Describable Languages
- Languages From Number Theory

Alphabets, Strings, and Languages

- We're going to finish the course by doing a bit of **formal language theory**, the branch of mathematics that deals with finite **strings** and with languages, where a **language** is defined to be a *set of strings*.
- If Σ is a finite **alphabet** (Rosen usually uses {0, 1}, while my book often uses {a, b}), then a **string** over Σ is defined to be a sequence of zero or more letters, each letter being an element of Σ. There is a special **empty string**, called λ, which has no letters. Two strings are equal if they have the same letters in the same order. So there is one empty string, because any two strings with no letters are equal to each other. The length of a string is the number of letters in it.
- The set of all finite strings over Σ is called Σ*. A language is any subset of Σ*, that is, any set of strings. Languages may be finite or infinite. We can use the operators of set theory on languages, and set notation to describe them.

Regular Expressions

- A **regular expression** is a string that **denotes** a particular language. They are defined recursively, much as boolean expressions or arithmetic expressions are defined. A **regular language** is a set of strings that is denoted by a regular expression. We'll see soon that some languages are regular and some are not.

- The "regular expressions" you may have seen in programming languages and operating systems are related to this concept, but the notation will be different. For example, if in Unix you want to delete all files whose names begin with "q", you would say "`rm q*`". The set of all strings beginning with "q" is denoted by the regular expression "$q\Sigma^*$".

- Our main result about regular expressions will be **Kleene's Theorem**, which says that a language can be *denoted* by a regular expression if and only if it can be *decided* by a **finite-state machine**. We'll define finite-state machines next lecture, and prove Kleene's Theorem by the end of the course.

Formal Language Theory

- Why would we care about something like this? The study of formal languages began as an outgrowth of the study of **natural languages**, those such as English or Chinese that are used by humans.

- Natural languages obey **grammatical rules**, and human brains are able to determine whether different utterances are valid in the language, and if so what they mean. Computers can determine whether statements are valid for a programming language, and if so what they mean. We'd like to understand how the humans do it, in part by designing computers to do similar things.

- There's a general phenomenon where the set of languages that can be denoted in a certain way coincides with the set that can be decided in a certain way. Kleene's Theorem is our first example, and you will see others in CMPSCI 401. Formal language theory gives a mathematical way to reason about the resources needed to carry out computational tasks.

The Formal Inductive Definition

- Fix an alphabet Σ. A **regular expression over Σ** is a string over the alphabet Σ ∪ {∅, +, ·, *, (,)} that can be built by the rules below. Each regular expression R denotes a language L(R), also determined by the rules below.
- "∅" is a regular expression and denotes the empty set. If a is any letter in Σ, then "a" is a regular expression and denotes the language {a}.
- If R and S are two regular expressions denoting languages L(R) and L(S), then "R·S" (often written "RS") is a regular expression denoting the **concatenation** L(R)L(S), and "R+S" is a regular expression denoting the **union** L(R) ∪ L(S).
- If R is a regular expression denoting the language L(R), then "R*" is a regular expression denoting the **Kleene star** of L(R), which is written L(R)*.
- Nothing else is a regular expression.

iClicker Question #1: Concatenation of Languages

- If X is the language {ab, aba} and Y is the language {a, ba, aba}, what is XY?
- (a) {a, ab, ba, aba}
- (b) {aab, aaba, baab, baaba, abaab, abaaba}
- (c) {aba, ababa, abaaba}
- (d) {aba, abba, ababa, abaa, abaaba}

iClicker Question #2: Valid Regular Expressions

- Which of these is not a valid regular expression for the alphabet {0, 1}, according to the rules on the previous slide?
- (a) (010)* + 0*1(01)*1
- (b) 0 + 01* + (1 + 001*)*
- (c) 101* + 10(*01)*
- (d) (011)**

The Kleene Star Operation

- If A is any language, the Kleene star of A, written A*, is the set of all strings that *can* be written as the concatenation of *zero or more* strings from A.
- If A = ∅, A* = {λ} because we can only have a concatenation of zero strings from A. If A = {a}, then A* = {λ, a, aa, aaa, aaaa,...}, the set of all strings of a's.
- If A = Σ, then A* is just Σ*, so the star notation we have been using for "Σ*" is just this same Kleene star operation. A string over Σ is just the concatenation of zero or more letters from Σ.
- In general A* is the union of the languages A^0, A^1, A^2, A^3,... where $A^0 = \{\lambda\}$, $A^1 = A$, $A^2 = AA$, $A^3 = AAA$, and so on. (Note that some of the laws of exponents still work, like $A^iA^j = A^{i+j}$ and $(A^i)^j = A^{ij}$.)

iClicker Question #3: Kleene Star

- What is the relationship between the languages denoted by a*b* and (ab)*?
- (a) they are equal
- (b) a*b* $\subseteq$ (ab)*
- (c) (ab)* $\subseteq$ a*b*
- (d) None of the above

Finite Languages

- The regular expression “aba” denotes the concatenation {a}{b}{a} = {aba}, by the definition of concatenation of languages. Thus any language consisting of a single non-empty string has a regular expression, which is (up to a type cast) itself. The language {λ}, as we just saw, can be written “∅*”.

- If I have any **finite** language, I can denote it by a regular expression, as the union of the one-string languages for each of the strings in it. For example, the finite language {λ, aba, abbb, b} is denoted by the regular expression “∅* + aba + abbb + b”. (Note that “+” is *not* the Java concatenation operator!)

- A regular expression that never uses the star operator must denote a finite set of non-empty strings. (We can prove this fact using induction!) If we use the star operator on any language that contains a non-empty string, the result is an infinite language, such as (aa)* = {λ, aa, aaaa, aaaaaa,...}.

The Language (a + ab)*

- Here is a more interesting regular language, denoted by the regular expression “(a + ab)*. (Note that the parentheses are important -- “a + ab*” and “a + (ab)*” denote quite different languages.) The strings in (a + ab)* are exactly those strings that can be made by concatenating zero or more strings, each of which is equal to either a or ab.

- We can systematically list (a + ab)* by listing $(a + ab)^i$ in turn for each natural i. We get $(a + ab)^0 = \{\lambda\}$, $(a + ab)^1 = \{a, ab\}$, $(a + ab)^2 = \{aa, aab, aba, abab\}$, $(a + ab)^3 = (aaa, aaab, aaba, aabab, abaa, abaab, ababa, ababab\}$, and so forth.

- How can we tell whether a given string of a’s and b’s is in (a + ab)*? If it ends in a, we know that the last string used in the concatenation was “a”, and if it ends in b, the last string used was “ab”. So we can delete a’s and ab’s from the right as long as we can, and if we produce λ then the string was in the language. It turns out that (a + ab)* is the set of strings that don’t begin with b and never have two b’s in a row. (How would you *prove* this assertion?)

Logically Describable Languages

- We can say “the first letter is not b and there are never two b’s in a row” in the predicate calculus. One way to do it is to have variables that range over *positions in the string*. Our atomic predicates are “$C_a(x)$” (“position x contains an a”, “$C_b(x)$” (“position x contains a b”), “$x = y$” (“x and y are the same position”), and “$x < y$” (“x is to the left of y”).

- So we can say that the first letter is not b, “$\neg\exists x: C_b(x) \wedge \forall y: x \leq y$”, and that there are never two b’s in a row, “$\neg\exists x: \exists y: C_b(x) \wedge C_b(y) \wedge (x < y) \wedge \forall z: (z \leq x) \vee (z \geq y)$”. One way to say both things at once is “$\forall x: C_b(x) \rightarrow \exists y: \mathrm{Pred}(x, y) \wedge C_a(y)$”, where “$\mathrm{Pred}(x, y)$” abbreviates “$(x < y) \wedge \forall z: (z \leq x) \vee (z \geq y)$”.

- In last year’s honors section of CMPSCI 401, we proved that a language is **logically describable** in this way if and only if it has a certain kind of regular expression, and that (aa)* is *not* logically describable. Prof. Immerman was the prime developer of an extensive theory about more complicated logical descriptions.

Languages From Number Theory

- We can easily make a regular expression for the set of even-length strings of a's, "(aa)*", or the odd-length strings of a's, "(aa)*a", or the set of strings of a's whose length is congruent to 3 modulo 7, "$a^3(a^7)^*$", or the set of strings whose length is congruent to 1, 2, or 5 modulo 6, "$(a + a^2 + a^5)(a^6)^*$".

- What about the set of strings over {a,b} that have an even number of a's? A good first guess is that such a string is a concatenation of zero or more strings, each of which has exactly two a's. This would be the language (b*ab*ab*)*.

- But this isn't exactly right, because "bb", for example, has 0 a's and 0 is even. A correct regular expression for this language is (b + ab*a)* -- we can divide any such string into pieces which either have exactly two a's (with some number of b's between) or are just b's themselves.

- It's harder to get a number of a's congruent to 3 mod 7, or the strings with an even number of a's *and* an even number of b's, but both are possible.

iClicker Question #4: Counting Languages

- Which regular expression denotes the same language as (aa)* + (aaa)*? Note that we can also write this language as $\{a^i$: i is even or i is divisible by 3$\}$.
- (a) (aaaaa)*
- (b) (∅* + aa + aaa + aaaa)(aaaaaa)*
- (c) (aa)*(aaa)*
- (d) (aaaaaa)*