CSE 322 - Introduction to Formal Methods in Computer Science
Regular Expressions

Dave Bacon
Department of Computer Science & Engineering, University of Washington

Having just put behind us our first major theoretical result, that the class of languages recognized by DFAs and NFAs is the same, we now turn to a more useful, and more widely used, concept: the regular expressions. Regular expressions are used fairly ubiquitously in programs like `grep`, emacs, and vi (okay that last one is for the hard-core only). They are also useful for that cool syntax highlighting you get in your editor when your programming. Further they are used widely in lexical analyzers which are used to take a computer program and break it down into tokens.

To begin, lets just introduce a regular expression:

$$(0 \cup 1)1^* \tag{1}$$

Okay, now thats a weird object, involving things which look like they come from some alphabet 0 and 1 along with things which were previously regular operations, like $\cup$ and $*$. Well it turns out that a regular expression is one of our favorite friends, a language. But it doesn't look like a language! Well that's because it is an *expression* which is a shorthand for the language it is describing. In particular objects like 0 really stand for the language $\{0\}$. Similarly the object 1 is the language $\{1\}$. Now you can see that $(0 \cup 1)$ is really just the language $\{0\} \cup \{1\} = \{0, 1\}$. Further $1^* = \{1\}^*$, the set of all strings made up of all 1s and the empty string. But what does it mean to have these two languages sitting beside each other? Well this simply means that we concatenate the languages. Thus the language of this regular expression (sometimes we use $L(x)$ to denote the language of the regular expression x) is

$$(0 \cup 1)1^* = \{0, 1\} \circ (\{1\}^*) \tag{2}$$

Regular expressions are ways of expressing some languages which is very compact and efficient and easy to write down. Regular expression are made up of elements of an alphabet, symbols for empty strings, symbols empty sets, combined using regular operations. Further, these operations, have a rule of precedence. Just like $5 \times 3 + 4$ evaluates the $\times$ before the $+$, there are a similar set of rules for applying the regular operations in regular expressions. In particular, the highest precedence operation is the $*$ operations, the next highest is the concatenation operation, $\circ$, and finally the union $\cup$ is the lowest precedence. Oftentimes you will see the union written $+$ instead of $\cup$. This has the added advantage of making the rules of precedence similar to that of arithmetic. And further, just like in arithmetic, you can used parenthesis to override these rules and specify a certain order in how the regular expression is evaluated.

To make sure you understand the rules of precedence, you can try something like

$$0 \cup 10^* \tag{3}$$

The highest precedence operation is $*$, so we do this first, so 0^* is the language of all strings made up entirely of 0s and the empty string. Then we do the concatenation of 1 and 0^*, producing the language consisting of all strings beginning with a 1 followed by any number of 0s (including zero.) Finally we apply the union operation, unioning the language we just discussed with the language $\{0\}$. The final language is therefore all strings beginning with a 1 followed by any number of 0s and (union) the string 0.

I. FORMAL DEFINITION

We now proceed with a formal definition of regular expressions. This definition is inductive. That is it tells you what some base objects are which are regular expressions, and then tells you rules that you can use to build up regular expressions. Anything which can be built up from these rules will be a regular expression.

1. a for $a \in \Sigma$ is a regular expression. Here Σ is the alphabet over which the regular expression is written. This represents the language $\{a\}$.
2. ε, the empty string, is a regular expression. Here ε represents the language made up of the empty string $\{\varepsilon\}$.
3. $\emptyset$, the empty set, is a regular expression. $\emptyset$ represent the language with no elements $\{\}$.

4. If R_1 and R_2 are regular expressions, then so is $R_1 \cup R_2$. This represents the language made up of the union of the languages represented by R_1 and R_2.

5. If R_1 and R_2 are regular expressions, then so is $R_1 \circ R_2$. This represent the language made up of the concatenation of the languages represented by R_1 and R_2.

6. If R is a regular expression, then so is R^*. This, of course, represents the language made up of taking the $*$ of the language represented by R.

II. SOME OTHER SYMBOLS AND SHORTHANDS

In addition to the concatenation, union, and star operations, it is convenient to define some other operations which we can define in terms of the base operations. In particular it is useful to define $R^+ = RR^*$. This represents the language made up of strings which are concatenations of 1 or more elements of R. In other words, unlike the $*$ operation by itself, which always produces an empty string R^+ doesn't necessarily include the empty string (but it might, if R contains the empty string.) Similarly it is useful to denote R^k for the concatenation of R with itself k times. We also often use Σ in regular expressions. Technically, if $\Sigma = \{a_1, a_2, \ldots, a_m\}$, then Σ represents the language made up of strings of length one taken from the alphabet. (If we are careful, Σ, is made up of symbols, whereas the language $L(\Sigma)$ is the set of strings of length one taken over the entire alphabet.)

III. A FEW EXAMPLES

It is useful to work through some simple examples to test that you understand the languages represented by a regular expression

$$\begin{aligned}
(\Sigma\Sigma\Sigma)^* &= \{w| \text{ the length of } w \text{ is a multiple of three}\} \\
(0 \cup \varepsilon)(1 \cup \varepsilon) &= \{\varepsilon, 0, 1, 01\} \\
1^*\emptyset &= \emptyset \\
1^*(01^+)^* &= \{w| \text{ every 0 in } w \text{ is followed by at least one 1}\} \qquad (4) \\
\emptyset^* &= \{\varepsilon\} \qquad (5)
\end{aligned}$$

Going in the reverse direction is a very important skill. Sometimes this is not trivial. Lets do a hard one. What is the regular expression for the language made up of strings of 0s and 1s where all pairs of adjacent 0s appears before any pair of adjacent 1s. Think about this one a bit before moving on.

See, these are tricky (and this isn't even the trickiest one I can think of!) In order to solve this, lets break this down into two pieces. The first piece is made up of strings where no two 1s appear adjacent to each other. A little thought should show you that $(0 \cup 10)^*(1 \cup \varepsilon)$ is a regular expression for this language. Why does this work? Well $(0 \cup 10)^*$ gives a language made up of strings which do not contain adjacent 0s but which end in 0. We also need to include the strings which end in a 1. This is the role of concatenating with $(1 \cup \varepsilon)$. Now we can ask for the second part of the problem: strings where where no two 0s appear adjacent to each other. This is the same as our previous expression, but with the role of 0 and 1 reversed: $(1 \cup 01)^*(0 \cup \varepsilon)$. Thus a regular expression for the entire language is $(0 \cup 10)^*(1 \cup \varepsilon)(1 \cup 01)^*(0 \cup \varepsilon)$. Clever readers may be able to compress this a bit!

IV. TO TEST YOUR ε AND $\emptyset$ KNOWLEDGE

Don't confuse ε and $\emptyset$ in regular expressions. Here are some example which should help you understand the differences.

1. If R is a regular expression, then $R \cup \emptyset = R$.

2. If R is a regular expression, then $R \circ \varepsilon = R$.

3. If R is a regular expression, then $R \cup \varepsilon$ may not equal R. For example if $R = 1$, then $L(1 \cup \varepsilon) = \{1\} \cup \{\varepsilon\} = \{1, \varepsilon\}$.

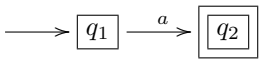
4. If R is a regular expression, then $R \circ \emptyset$ may not equal R. For example if $R = 1$, then $L(1 \circ \emptyset) = \{1\} \circ \{\} = \{\} = \emptyset$.

V. REGULAR EXPRESSIONS YIELD REGULAR LANGUAGES

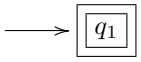
In future lectures we will be showing that regular expressions and finite automata have equivalent descriptive powers. Here we will show the easy direction of this proof: that is that if a language is describable by a regular expression, then it is regular. To do this we need to show how to convert a regular expression into a NFA which recognizes the language of R. Let R be the regular expression we are converting and $A = L(R)$ the language of this regular expression.

To construct a NFA for a regular expression, we will show how to construct a NFA for each step of the formal definition of a regular expression.

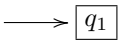
1. $R = a$ for some $a \in \Sigma$. We need a NFA which accepts the language $\{a\}$. This is given by



2. $R = \varepsilon$. We need a NFA which accepts the language $\{\emptyset\}$. This is given by



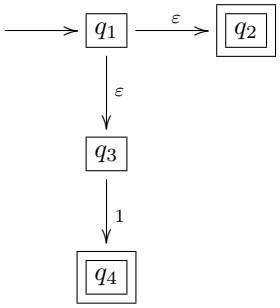
3. $R = \emptyset$. This NFA should have the empty language as its language. This can be achieved by



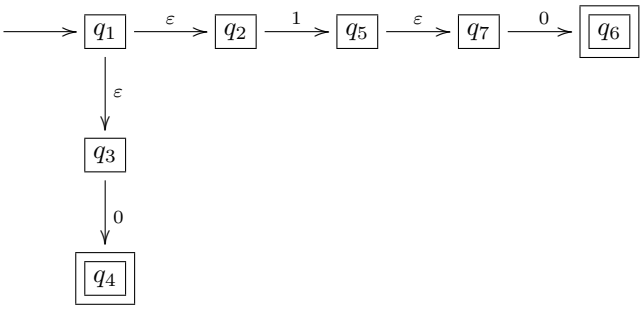
4. $R = R_1 \cup R_2$. We constructed this machine in a previous lecture.
5. $R = R_1 \circ R_2$. We constructed this machine in a previous lecture.
6. $R = R_1^*$. We constructed this machine in a previous lecture.

A. Example

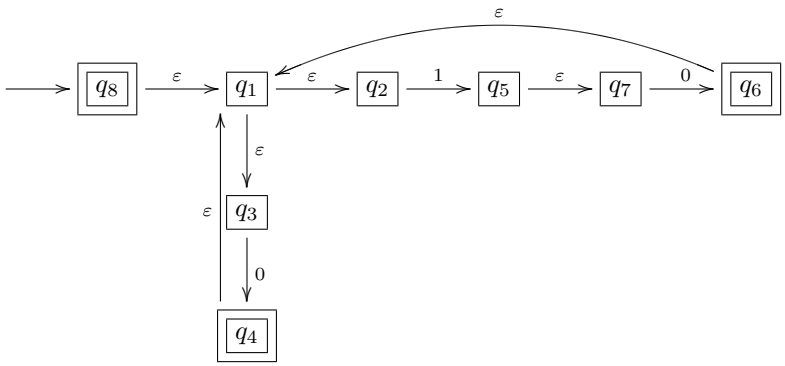
Lets work through an example for the regular expression $(1 \cup \varepsilon)(0 \cup 10)^*$. To do this we break it down into steps. First consider $1 \cup \varepsilon$. The 1 and ε machines are given above and we need to union them. So, via the union construction for NFAs, we produce the machine



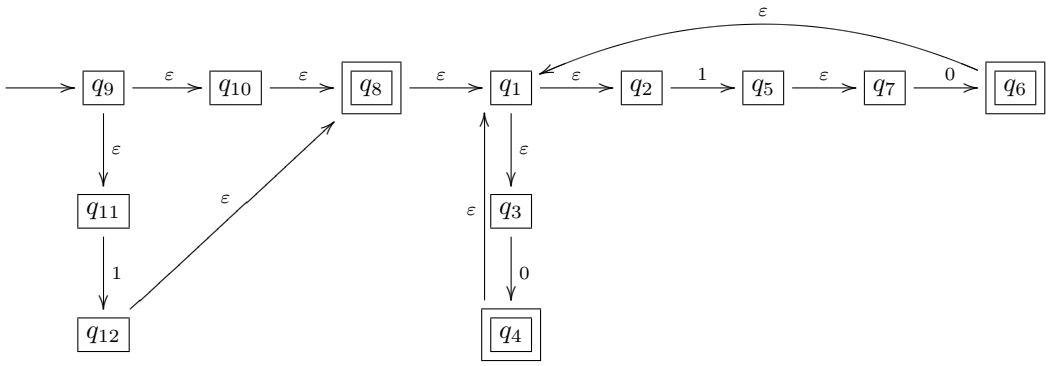
Now lets construct the NFA $(0 \cup 10)$.



To make this into $(0 \cup 10)^*$ we need to add a state and add the appropriate ε transitions



Finally we need to concatenate these structures together.



Wah lah!