Regular Expressions
Theory of Computation (CSCI 3500)

Prof. Harley Eades (heades@gru.edu).

Read chapter 1.3.

Suppose we are programmers at a bank of some kind, and we are tasked with constructing a program that first prompts the user for their US social security number, and then gives the user their credit card report. This program must take in data from the user, and then do a selection from the database containing the users highly sensitive information. Naturally, before doing the selection we must first check that the users input matches the pattern of an US social security number. If do not do this check, then the program would be susceptible to injection attacks which is a major security problem.

The form of a US social security number is *ddd-dd-dddd* where $d \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. As programmers how do we take the input from the user, and check it against this pattern? For example, if the user entered $123-45-7689$ then we would want to proceed, because this matches the pattern, but if they entered $123 - 45 - 67891$ would have need to issue an error message, because this does not match the pattern.

One possible way of matching the input to the pattern is by using a NFA or a DFA. The language $SS = \{ddd\text{-}dd\text{-}dddd \mid d \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}\}$ is the set of all possible security numbers, and it is easy to see that this is a regular language. So we could implement NFAs, and then define the NFA whose language is SS. Then to test to see if the user input matches the social security pattern we simply run SS's NFA on the user input, and then test for acceptance.

Now suppose our employeer asks for in addition to the social security number we also prompt for the email address of the user. How do we insure that the user input matches a valid email address? Again, we can use an NFA, but now we have two somewhat complex NFAs hardcoded into our program.

What if instead we had a small language that allows one to specify patterns, and that can be converted into a NFA on the fly? We would want this pattern specification language to only specify regular languages to insure that an NFA can always be constructed. We are in luck, because such a specification language exists and is called the regular expression language. In fact, regular expressions can be found in any modern day programming language or scripting language.

In this note we define mathematically what regular expressions are and then prove that they define regular languages. We also show that every NFA can be converted into a regular expression. This then implies that NFAs and regular expressions are in bijection with each other. This means that neither is more powerful than the other.

1 The Langauge of a Regular Expression

In this section define what a regular expression looks like, and then explain how we can give them meaning using the regular operators on regular languages.

Definition 1. *Fix an alphabet Σ. We call R a regular expression if and only if R is*

1. *a for some $a \in \Sigma$,*

2. ϵ,

3. $\emptyset$,

4. $(R_1 \cup R_2)$, *where* R_1 *and* R_2 *are regular expressions,*

5. $(R_1 \circ R_2)$, *where* R_1 *and* R_2 *are regular expressions, and*

6. R_1^* *where* R_1 *is a regular expression.*

Thus, to define a regular expression one must first fix an alphabet, and then use only the above operators to construct complex expressions. Throughout the sequel we will often denote $R_1 \circ R_2$ by R_1R_2 in the interest of brevity.

Here are a few examples:

- $(ab)^*a$
- $a(a \cup b)^*$
- $(aabb \cup ab \cup a)^*$
- $ab(bab \cup bb)^*a$
- $(0 \cup 1)^*001(0 \cup 1)^*$
- For any set of generators $G = \{a_0, a_1, \ldots, a_i\}$ we can define the regular expression recognizing it as $a_0 \cup a_1 \cup \cdots \cup a_i$, and we will denote this regular expression by the set itself. Thus, the regular expression Ga_5 corresponds to the regular expression $(a_0 \cup a_1 \cup \cdots \cup a_i)a_5$.
- Let $\Sigma = \{0, 1\}$, then $\Sigma^*1\Sigma^*$ denotes the regular language consisting of all words with at least one 1 as a subword.
- Let $\Sigma = \{a, b, c\}$, then $\Sigma^*\Sigma^*$ denotes the regular language consisting of all words of even length.
- Let A be the english alphabet, and $D = \{edu, com, org, net\}$. Then the regular expression $AA^*@AA^*.D$ recognizes the regular language of email addresses with a S top-level domain.

Similarly, to the previous structures we have studied we denote the language of a regular expression by $L(R)$ for some regular expression R. Given a regular expression it is possible to construct its language.

Definition 2. *Given a regular expression* R *over the alphabet* Σ*, we can construct its language* $L(R)$ *by recursion as follows:*

- *Suppose* $R = a \in \Sigma$*. Then* $L(R) = \{a\}$.
- *Suppose* $R = \epsilon$*. Then* $L(R) = \{\epsilon\}$.
- *Suppose* $R = \emptyset$*. Then* $L(R) = \{\}$.
- *Suppose* $R = R_1 \cup R_2$ *for some regular expressions* R_1 *and* R_2*. Using the recursive call we obtain* $L(R_1)$ *and* $L(R_2)$*. Thus, we can define* $L(R) = L(R_1) \cup L(R_2)$.
- *Suppose* $R = R_1 \circ R_2$ *for some regular expressions* R_1 *and* R_2*. Using the recursive call we obtain* $L(R_1)$ *and* $L(R_2)$*. Then we define* $L(R) = \{w_1w_2 \mid w_1 \in L(R_1), w_2 \in L(R_2)\}$.
- *Suppose* $R = R_1^*$ *for some regular expression* R_1*. First, consider an arbitrary language* W*. How do we construct the set* W^**? We define this set as follows:*
  - *For any* $w \in W$, $w \in W^*$.
  - *For any* $w_1, w_2 \in W^*$, *then* $w_1w_2 \in W^*$.

We can now use this construction to define $L(R)$. Using the recursive call we obtain $L(R_1)$. Then we define $L(R) = L(R_1)^*$.

Notice that the above construction uses the regular operators on regular languages. This suggests that it might be possible to prove that $L(R)$ is regular.

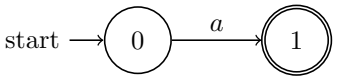
2 Regular Expression Conversion to NFA

In this section we show that the language a regular expression describes is regular. The proof of this fact requires the conversion of a regular expression into a NFA.

Lemma 3 (Regular Languages are Regular Expressions)**.** *For any regular expression R, $L(R)$ is regular.*

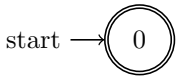
Proof. Suppose R is a regular expression over an alphabet Σ. This proof uses the observation we made about the use of the regular operators in the construction of $L(R)$. We proceed by induction on the form of R, and construct an NFA, M, such that $L(M) = L(R)$.

Case. Suppose $R = a \in \Sigma$. Then we construct M as follows:



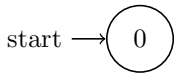
Clearly, $L(M) = L(R)$.

Case. Suppose $R = \epsilon$. Then we construct M as follows:



Clearly, $L(M) = L(R)$.

Case. Suppose $R = \emptyset$. Then we construct M as follows:



Clearly, $L(M) = L(R)$.

Case. Suppose $R = R_1 \cup R_2$ for some regular expressions R_1 and R_2. Using the induction hypothesis on R_1 and R_2 we obtain NFAs M_1 and M_2 such that $L(M_1) = L(R_1)$ and $L(M_2) = L(R_2)$ respectively. Then we can use the union regular operator construction for NFAs to obtain the definition M.

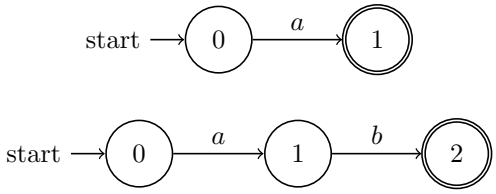
Case. Suppose $R = R_1 \circ R_2$ for some regular expressions R_1 and R_2. Using the induction hypothesis on R_1 and R_2 we obtain NFAs M_1 and M_2 such that $L(M_1) = L(R_1)$ and $L(M_2) = L(R_2)$ respectively. Then we can use the regular-operator concatenation construction for NFAs to obtain the definition M.

Case. Suppose $R = R_1^*$ for some regular expression R_1. Using the induction hypothesis on R_1 we obtain the NFA M_1 such that $L(M_1) = L(R_1)$. Then we can use the union regular operator construction for NFAs to obtain the definition of M.

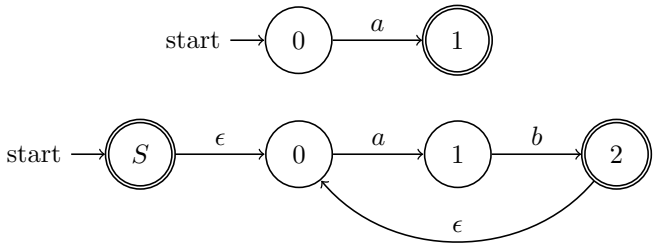
□

Consider the regular expressions $(ab)^*a$. First, convert all of the singleton languages to their respective NFAs, and then apply their respective regular operators to their NFAs.

The singleton languages of the regular expression under consideration are $\{ab\}$ and $\{a\}$. We can easily convert these to NFA's:



Now we can preform the respective regular operators to these NFAs. We first star the second NFA:



Finally, we concatenate them together: