CS 4510 Automata and Complexity January 18th 2023

Lecture 3: Regular Expressions

Lecturer: Abrahim Ladha *Scribe(s): Yitong Li*

1 Regular Expressions

You may be familiar with regular expressions as they are implemented on UNIX systems or in programming languages. In the terminal, if we enter "`ls *.pdf`", it will replace the `*` by any string, outputing a list of files which end with the `.pdf` extension. The regular expressions we will discuss in this lecture predate those.

A regular expression is a string representation of a regular language. You can think of regular expressions as a kind of very limited programming language. Each regular expression is declarative of exactly what strings it wants to accept. First we define them, and then we will prove they correspond only to the regular languages.

Definition 1.1. We say that R is a **regular expression**, or regex, if R is one of the following:

(a) $\emptyset$ - empty set

(b) ε - empty string

(c) $a \quad \forall a \in \Sigma$

(d) R_i^*, $R_i R_j$ or $R_i \cup R_j$ where R_i, R_j are regular expressions.

$\diamondsuit$

- We have three base cases. Note that although we write ε, a, these actually correspond to the sets $\{\varepsilon\}, \{a\}$. We then have three inductive operations, union, concatenation, and the Kleene star.
- For R_i corresponding to some language L_i and R_j corresponding to some language L_j. The regular expression $R_i \cup R_j$ corresponds to the language $L_i \cup L_j$. Again, $R_i \cup R_j$ is a string containing the "$\cup$" symbol, while $L_i \cup L_j$ is a language, a possibly infinite set.
- For R_i corresponding to some language L_i and R_j corresponding to some language L_j, the regular expression $R_i R_j$ corresponds to the language $L_i L_j$. The concatenation of languages is defined like the cartesian product of sets. $L_i L_j = \{xy \mid \forall x \in L_i, \forall y \in L_j\}$
- For R_i corresponding to some language L_i, the regular expression R_i^* corresponds to the language

$$L_i^* = \bigcup_{k=0}^{\infty} L_i^k = \{\varepsilon\} \cup L_i \cup L_i^2 \cup L_i^3 \cup ...$$

You should think of this as zero or more copies of strings from L_i. Note that for any language, $L^0 = \{\varepsilon\}$.

We have a recursive, or inductive definition for a naturally recursive or inductive object. Each regular expression is a string, but it corresponds to a language, a (possibly infinite) set of strings.

2 Examples

Here are some examples of regular expressions. We often use Σ in regular expressions as a shorthand for $(a \cup b)$ or whatever the alphabet is.

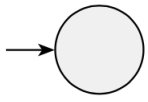
1. $a^* = \{a^i \mid i \in \mathbb{N}\} = \{\varepsilon, a, aa, aaa, \dots\}$
2. Σ^* We introduced this as the definition of all strings, it is actually a regular expression for zero or more copies of any of the letters of the alphabet, which corresponds to all strings.
3. $a^*ba^* = \{a^iba^j \mid i, j \in \mathbb{N}\}$ = all strings with a single b.
4. $\Sigma^*b\Sigma^*$ = all strings with atleaste one b. There can be more than one, but not zero.
5. $\Sigma^*aab\Sigma^* = \{$all strings with aab as a substring$\}$
6. $(a \cup b)^*aab(a \cup b)^*$
7. $(\Sigma\Sigma)^* = ((a \cup b)(a \cup b))^* = ((aa \cup ab \cup ba \cup bb))^*$ = all strings of even length
8. $(a \cup b)(b \cup c) = \{ab, bbac, bc\}$
9. $a^*\emptyset = \emptyset$
10. $\emptyset^* = \{\varepsilon\}$

3 $\mathscr{L}(REX) \subseteq \mathscr{L}(NFA)$

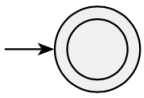
We now prove the regular expressions correspond exactly and only to the regular languages by doing a double set containment. Let $\mathscr{L}(REX)$ correspond to the class of languages which are produced by the regular expressions.

First we show that if a language is produced by a regular expression, then it is decided by an NFA. To prove that $\mathscr{L}(REX) \subseteq \mathscr{L}(NFA)$, we want to show that for each regular expression, there exists an equivalent NFA. Given that regular expressions are recursively defined, it is natural to choose to proceed by induction. Let R be a regular expression. We start with the following base cases:

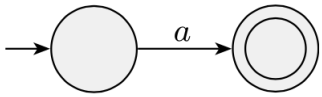
1. $R = \emptyset$.



2. $R = \varepsilon$.

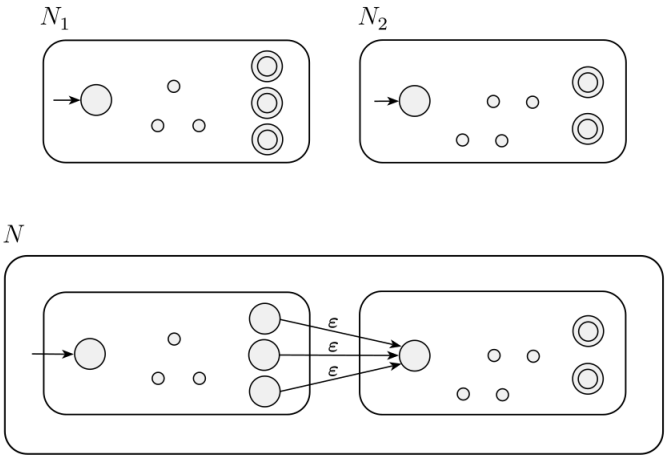


3. $R = a \in \Sigma$.



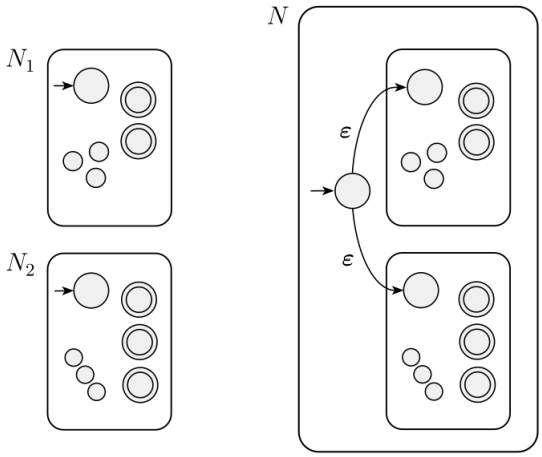
Next, we continue with the inductive steps. Let R_i, R_j be regular expressions that decide regular languages, by strong induction, we assume that there exist NFAs which decide exactly the languages that R_i, R_j produce. We will prove $R_i^*, R_iR_j, R_i \cup R_j$ also have NFAs to decide them. The proofs can be done graphically.

1. $R = R_iR_j$.



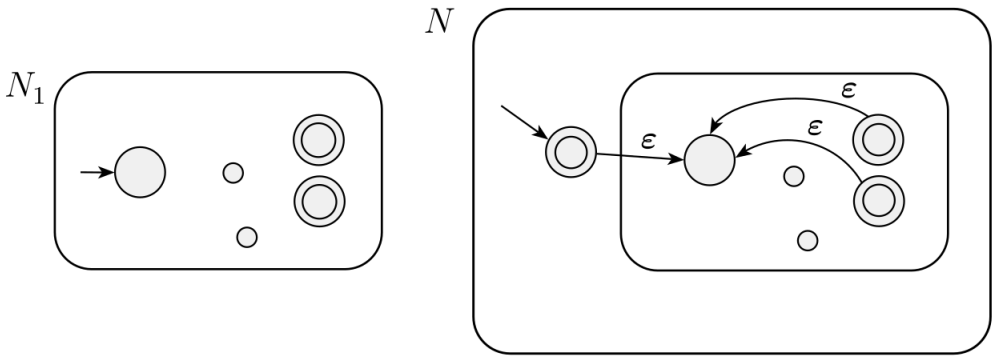
We remove final states F_i and $\forall f \in F_i$, add $\delta(f, \varepsilon) = q_j$ where q_j is the initial state of N_j. Consider a computation like a path through the NFA as a graph. To reach an accept state, you must go through the first NFA, then the second.

2. $R_i \cup R_j$.



add new start state q and $\delta(q, \varepsilon) = \{q_i, q_j\}$. Here you nondeterministically choose which NFA you wish to proceed on, so it decides the languages which reach the accepting states of either NFA, representing the union.

3. $R = R_i^*$.



We add new state q', ε-transition from q' and all states of F to the old start state q, mark q' as accepting. Note we could not have just made the start state accepting, but must add a new state. You can traverse an arbirary number of times on the internal NFA so this corresponds to zero or more copies, which is the Kleene star operation.

3.1 Example

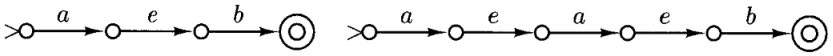
This proof not only shows every regular expression decides a regular language, but it gives a process to convert a regular expression into an NFA. Consider the following examples for $(ab \cup aab)^*$ and $(a \cup b)^*aba$

stage 1

a; b

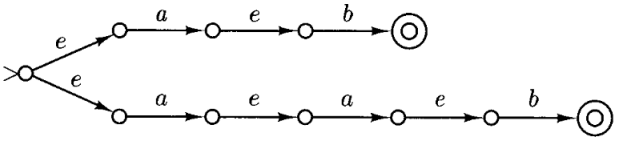
stage 2

ab; aab



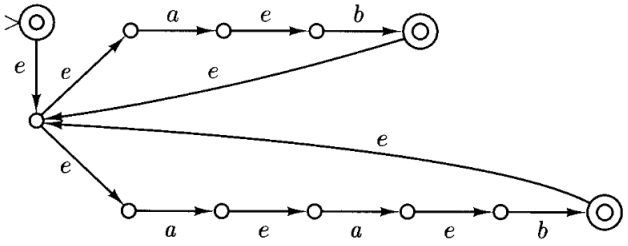
stage 3

$ab \cup aab$



stage 4

$(ab \cup aab)^*$



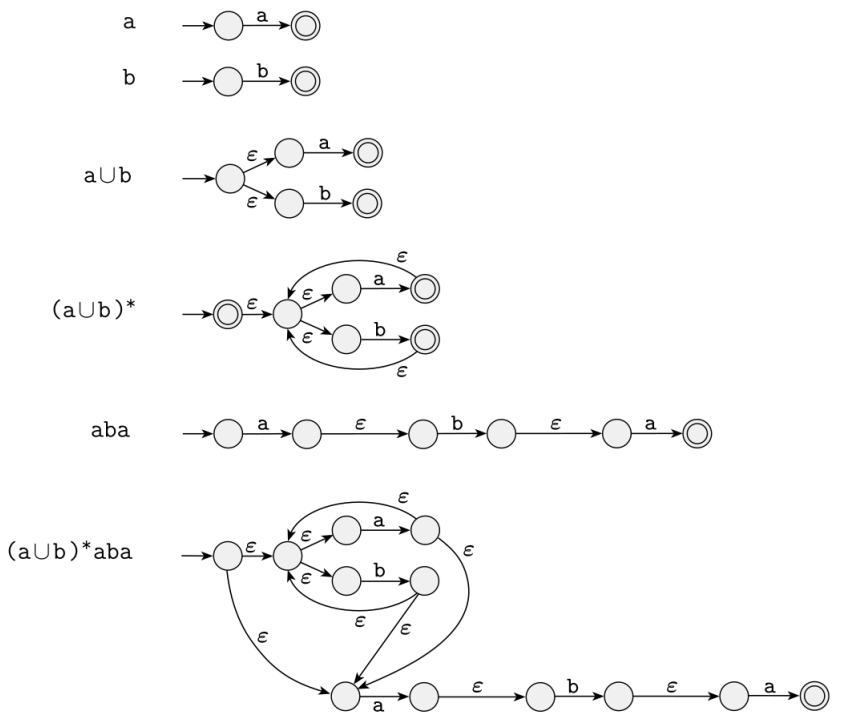


FIGURE 1.59
Building an NFA from the regular expression (a ∪ b)*aba

4 $\mathscr{L}(NFA) \subseteq \mathscr{L}(REX)$

We convert

Definition 4.1. The **GNFA** is defined as an NFA with the following properties:

(a) The transitions have a regular expression on them.

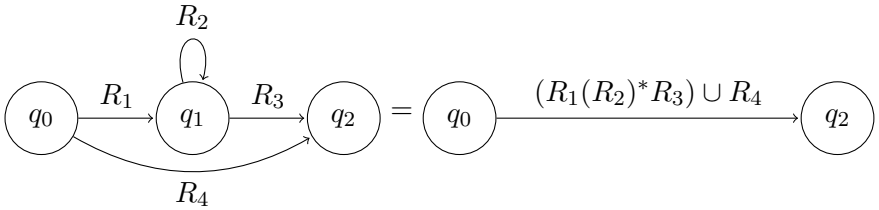
(b) The start state has no incoming transitions

(c) The final state has no outgoing transitions

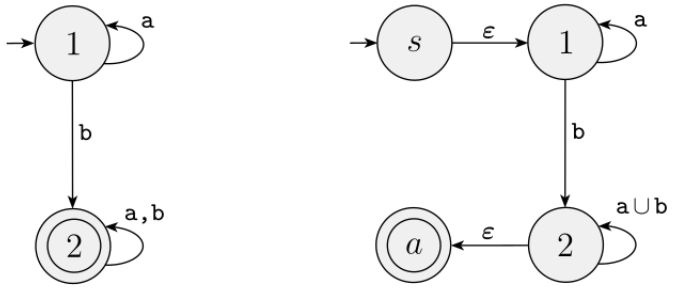
(d) Every pair of states has a transition.

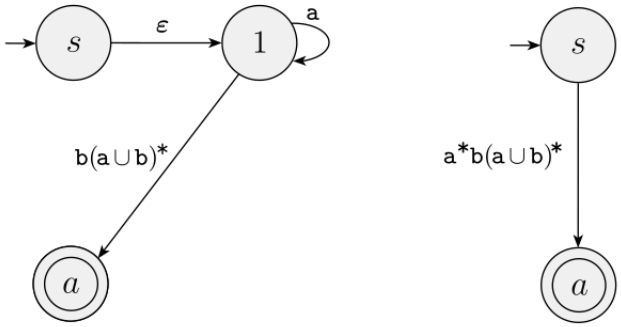
♢

Taking a transition in a DFA is reading some single symbol of the front of the input. Taking a transition of a GNFA is nondeterministically choosing some prefix of the input which satisfies the regex on the transition. To convert an NFA to a regular expression, we first add a new start and final state. Then, rip out one state at a time using the following rules until only two states and one transition is left.



For most very connected NFAs, conversion to a regex will result in one with exponential length. Here is a relatively simple example.





5 Set Closure

We say a set S is **closed** under an operation Δ if $\forall a, b \in S$, $a \Delta b \in S$.

1. $\mathbb{N}$ is closed under $+, \times$ and not closed under $-, \div$.
2. $\mathbb{Z}/\{0\}$ is closed under $+, -, \times$ and not closed under $\div$.
3. $\mathbb{Q}$ is closed under $+, -, \times$ and not closed under $\div$.
4. Regular languages are closed under $*, \circ, \cup$ and complement.

Although regular languages are closed under complement. Complement is not a valid operation of a regular expression. Using the operations regular languages are known to be closed under, we can prove closure under even more operations without having to contruct messy DFAs or NFAs. Recall we did a cartesian product of DFAs to prove that regular languages were closed under intersection. We give a shorter proof in the syntax of set theory.

Suppose that L_i, L_j are two regular languages. Then surely $\overline{L_i}, \overline{L_j}$ are regular, then surely $\overline{L_i} \cup \overline{L_j}$ is regular. Then so must be $\overline{\overline{L_i} \cup \overline{L_j}}$. From Demorgans law, we know that $\overline{\overline{L_i} \cup \overline{L_j}} = L_i \cap L_j$.

Similarly, we can show regular languages are closed under symmetric difference, or xor. We have a formula under composition of operators that we maintain closure under. $L_i \oplus L_j = (\overline{L_i} \cap L_j) \cup (L_i \cap \overline{L_j})$