COM364 Automata Theory
Lecture Note[*]3 - Regular Expressions

Kurtuluş Küllü

March 2018

Just like using arithmetic operations (addition, multiplication, ...) to build up arithmetic expressions, we can use regular operations (union, concatenation, star) to build up *regular expressions.* Regular expressions are expressions that describe languages. For example, expression $(0\cup 1)0^*$ stands for the language $\{0, 1, 00, 10, 000, 100, 0000, 1000, \dots\}$.

Note: Technically, union operation should be between sets. However, it is OK to write things like $0 \cup 1$ in the regular expressions because in the regular expressions, we actually use symbols like 0 and 1 to indicate sets $\{0\}$ and $\{1\}$.

Note: In different textbooks or other sources and in some application areas, you can come across slightly different notations. One example is the use of $|$ instead of $\cup$.

Note: Regular expressions provide a powerful mechanism for describing text patterns to search for. For an example usage, read about the grep command in UNIX.

Examples:

a) $(0 \cup 1)^*$ describes the set of all strings (including ε) made with 0 and 1.

b) $\big(0\,(0 \cup 1)^*\big) \cup \big((0 \cup 1)^*\,1\big)$ describes the binary strings that either start with 0 or end with 1.

Note: We accept an order of precedence as follows. Highest precedence (first to apply) is star operation, then concatenation, and then union.

Formal Definition

R is a regular expression if R is

1. a for some symbol a in an alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 R_2)$, where R_1 and R_2 are regular expressions, or
6. R_1^*, where R_1 is a regular expression.

Note: We also sometimes use a + symbol in regular expressions which can be defined using other operations as follows: $R^+ = RR^*$.

To distinguish between a regular expression and the language it describes, we write $L(R)$ for the language that R describes.

Examples: Let $\Sigma = \{0, 1\}$

- 0^*10^*

[*]Based on the book "Introduction to the Theory of Computation" by Michael Sipser.

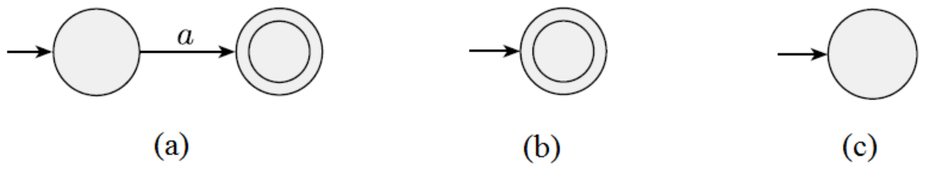

Figure 1: Base cases for converting regular expressions to NFA.

- $\Sigma^*1\Sigma^*$
- $1^*\left(01^+\right)^*$
- $(\Sigma\Sigma)^*$
- $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$
- $1^*\emptyset = \emptyset$ **Note:** Concatenating anything with $\emptyset$ produces $\emptyset$.

Example: Let $D = \{0, 1, 2, ..., 9\}$. What is the language of the following regular expression?

$$(+ \cup - \cup \varepsilon)\left(D^+ \cup D^+.D^* \cup D^*.D^+\right)$$

Equivalance with FA

Any regular expression can be converted into a FA that recognizes the described language and vice versa.

Theorem: A language is regular if and only if some regular expression describes it. If we separate this claim into two directions:

Lemma 1: If a language is described with a regular expression, it is regular.

Lemma 2: If a language is regular, then it can be described with a regular expression.

Proof Idea for Lemma 1: R is a regular expression and $L(R) = A$. Convert R into an NFA that recognizes A. Consider each of six cases in the formal definition of regular expressions.

1. If $R = a$ for some $a \in \Sigma$, then $L(R) = \{a\}$ and the NFA is as shown in Figure 1(a).
2. If $R = \varepsilon$, then $L(R) = \{\varepsilon\}$ and the NFA is as shown in Figure 1(b).
3. If $R = \emptyset$, then $L(R) = \emptyset$ and the NFA is as shown in Figure 1(c).
4. $R = R_1 \cup R_2$
5. $R = R_1R_2$
6. $R = R_1^*$

For each of the cases in 4 to 6 above, use the constructions given as proofs for closure under these operations.

Example: Find the NFA that recognizes the language of $(ab \cup a)^*$. We can construct this machine in a bottom up fashion. First, draw the NFAs that recognize the languages of regular expressions a and b (these are from item 1). Then, concatenate these machines to create a NFA that recognize the language of ab. Then, consider the union of ab with a and finally, consider the star operation. This whole process is shown in Figure 2.

As you can see, this procedure doesn't give us the FA with fewest possible states. But it is an automatic procedure that works for any regular expression.

Proof Idea for Lemma 2: We need to show that if A is a regular language, then we can write a regular expression that describes it. To do this, we will convert the DFA that recognizes A into a regular expression. There are two stages to this task.

- First, convert the given DFA into what we call a *Generalized NFA (GNFA)* (definition below).

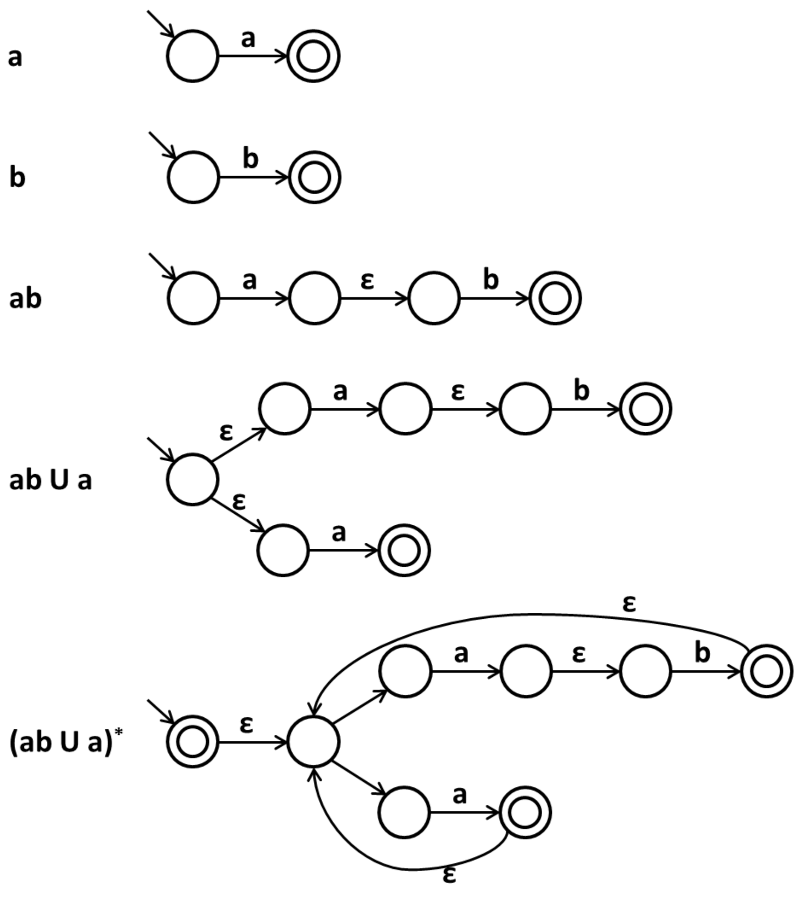

Figure 2: Steps of converting regular expression $(ab \cup a)^*$ to the NFA to recognize that language.

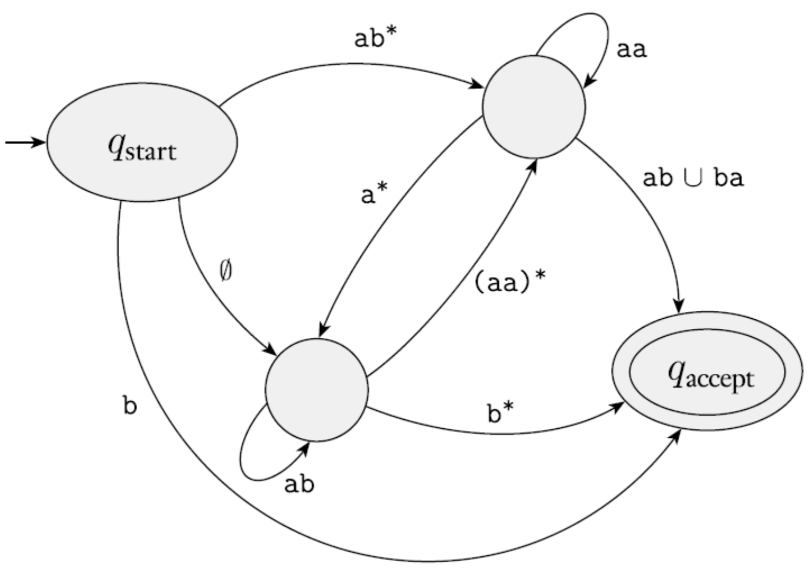

Figure 3: An example GNFA.

- Then, convert the GNFA to a regular expression.

Generalized NFA: This is an NFA in which the transitions can be regular expressions (instead of single symbols). An example GNFA is shown in Figure 3. For convenience, we need GNFAs to meet the following conditions.

- Start state has a transition to all others but none coming in.
- There is a single accept state and it has transitions fro all others but none going out.
- Start state is not the accepting state.
- Except the start and accepting states, one arrow goes from every state to every state (including itself).

We can convert a DFA to a GNFA as follows:

- Add a new start state with an ε-transition to the old one.
- Add a new accept state with ε-transitions from the old ones and make the old accepting states nonaccepting.
- If any arrows have multiple labels (or if there are multiple arrows from one state to another) use a single arrow taking a union.
- For cases where there is no arrow, use $\emptyset$ arrows.

Once we have a GNFA, we can convert it to a regular expression. Let the GNFA have k states (k has to be at least two). If $k > 2$, construct an equivalent GNFA with $k-1$ states. Repeat until $k = 2$. Once $k = 2$, there will be a single arrow from start state to the accept state. The label of that arrow is the regular expression we want. An example process for a DFA with three states is summarized in Figure 4.

To go from k states to $k-1$, select any state that is not the start or the accepting state, remove the state and adjust the labels so that the recognized language will be the same. The logic behind removing a state and changing the transition labels is shown in Figure 5.

Example: Let the DFA in Figure 6(a) be the one we are given. We can find the regular expression for the language this 2-state DFA recognizes as described above. The stages, in particular, the 4-state GNFA (b), the 3-state GNFA (c), and lastly the 2-state GNFA (d) are all shown in Figure 6.

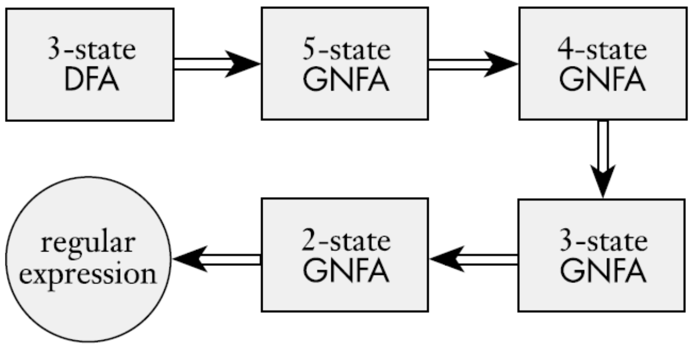

Figure 4: Summary of the process for converting a 3-state DFA to a regular expression.

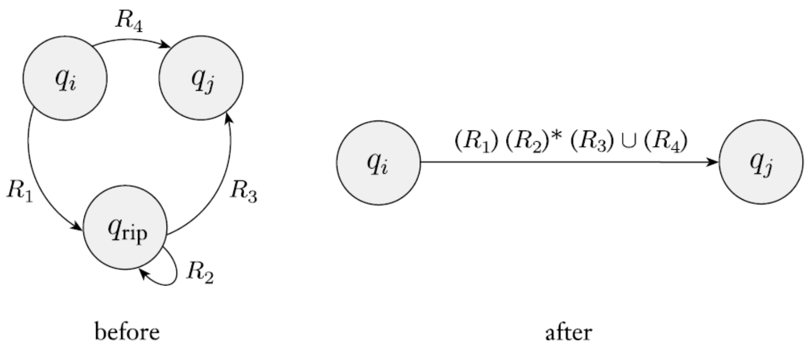

Figure 5: Summary of the process for converting a 3-state DFA to a regular expression.

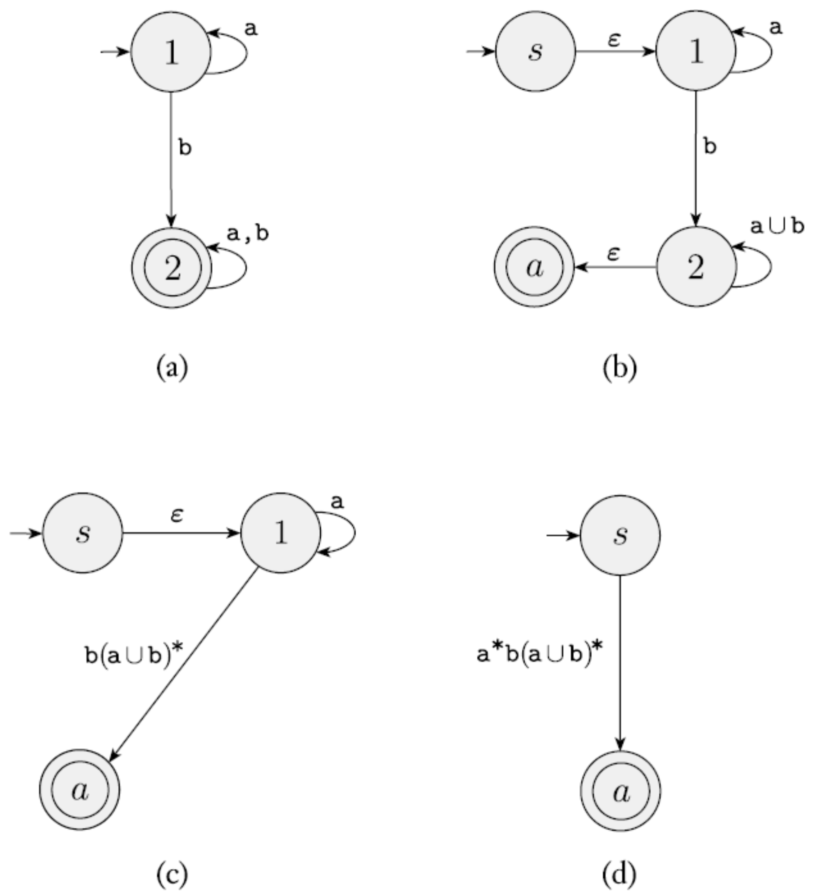

Figure 6: An example for converting a DFA to a regular expression.