Regular Expressions 02

Noman Amin

Lecturer, Dept. of CSE

Southeast University

Finite Automata and Regular Expressions

While the regular-expression approach to describing languages is fundamentally different from the finite-automaton approach, these two notations turn out to represent exactly the same set of languages, which we have termed the "regular languages." We have already shown that deterministic finite automata, and the two kinds of nondeterministic finite automata — with and without ϵ-transitions — accept the same class of languages. In order to show that the regular expressions define the same class, we must show that:

1. Every language defined by one of these automata is also defined by a regular expression. For this proof, we can assume the language is accepted by some DFA.

2. Every language defined by a regular expression is defined by one of these automata. For this part of the proof, the easiest is to show that there is an NFA with ϵ-transitions accepting the same language.

Equivalence of four different notations

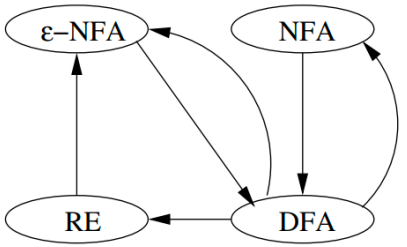


Figure 3.1: Plan for showing the equivalence of four different notations for regular languages

Converting Regular Expressions to Automata

BASIS: There are three parts to the basis, shown in Fig. 3.16. In part (a) we see how to handle the expression ϵ. The language of the automaton is easily seen to be $\{\epsilon\}$, since the only path from the start state to an accepting state is labeled ϵ. Part (b) shows the construction for $\emptyset$. Clearly there are no paths from start state to accepting state, so $\emptyset$ is the language of this automaton. Finally, part (c) gives the automaton for a regular expression **a**. The language of this automaton evidently consists of the one string a, which is also $L(\mathbf{a})$. It

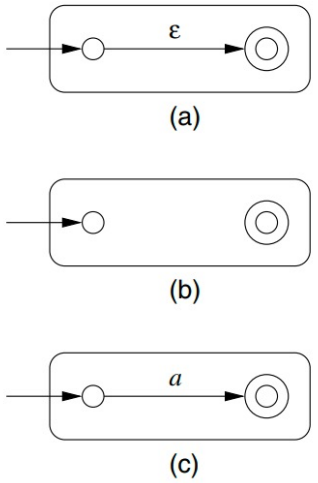


Figure 3.16: The basis of the construction of an automaton from a regular expression

Converting Regular Expressions to Automata

INDUCTION: The three parts of the induction are shown in Fig. 3.17. We assume that the statement of the theorem is true for the immediate subexpressions of a given regular expression; that is, the languages of these subexpressions are also the languages of ϵ-NFA's with a single accepting state. The four cases are:

1. The expression is $R + S$ for some smaller expressions R and S. Then the automaton of Fig. 3.17(a) serves. That is, starting at the new start state, we can go to the start state of either the automaton for R or the automaton for S. We then reach the accepting state of one of these automata, following a path labeled by some string in $L(R)$ or $L(S)$, respectively. Once we reach the accepting state of the automaton for R or S, we can follow one of the ϵ-arcs to the accepting state of the new automaton. Thus, the language of the automaton in Fig. 3.17(a) is $L(R) \cup L(S)$.

2. The expression is RS for some smaller expressions R and S. The automaton for the concatenation is shown in Fig. 3.17(b). Note that the start state of the first automaton becomes the start state of the whole, and the accepting state of the second automaton becomes the accepting state of the whole. The idea is that the only paths from start to accepting state go first through the automaton for R, where it must follow a path labeled by a string in $L(R)$, and then through the automaton for S, where it follows a path labeled by a string in $L(S)$. Thus, the paths in the automaton of Fig. 3.17(b) are all and only those labeled by strings in $L(R)L(S)$.

Converting Regular Expressions to Automata

3. The expression is R^* for some smaller expression R. Then we use the automaton of Fig. 3.17(c). That automaton allows us to go either:

   (a) Directly from the start state to the accepting state along a path labeled ϵ. That path lets us accept ϵ, which is in $L(R^*)$ no matter what expression R is.

   (b) To the start state of the automaton for R, through that automaton one or more times, and then to the accepting state. This set of paths allows us to accept strings in $L(R)$, $L(R)L(R)$, $L(R)L(R)L(R)$, and so on, thus covering all strings in $L(R^*)$ except perhaps ϵ, which was covered by the direct arc to the accepting state mentioned in (3a).

4. The expression is (R) for some smaller expression R. The automaton for R also serves as the automaton for (R), since the parentheses do not change the language defined by the expression.

Converting Regular Expressions to Automata

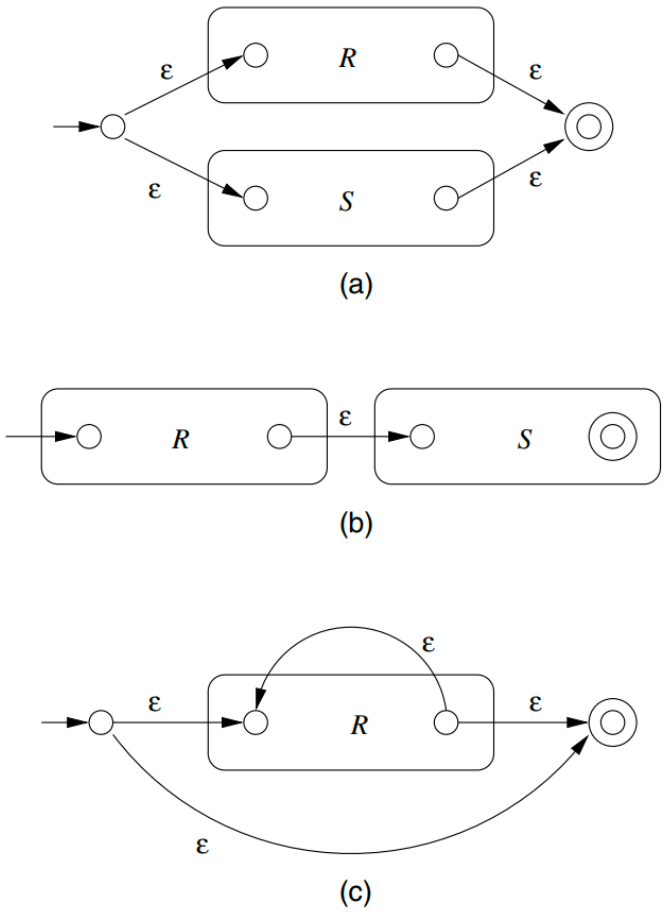


Figure 3.17: The inductive step in the regular-expression-to-ϵ-NFA construction

Converting Regular Expressions to Automata

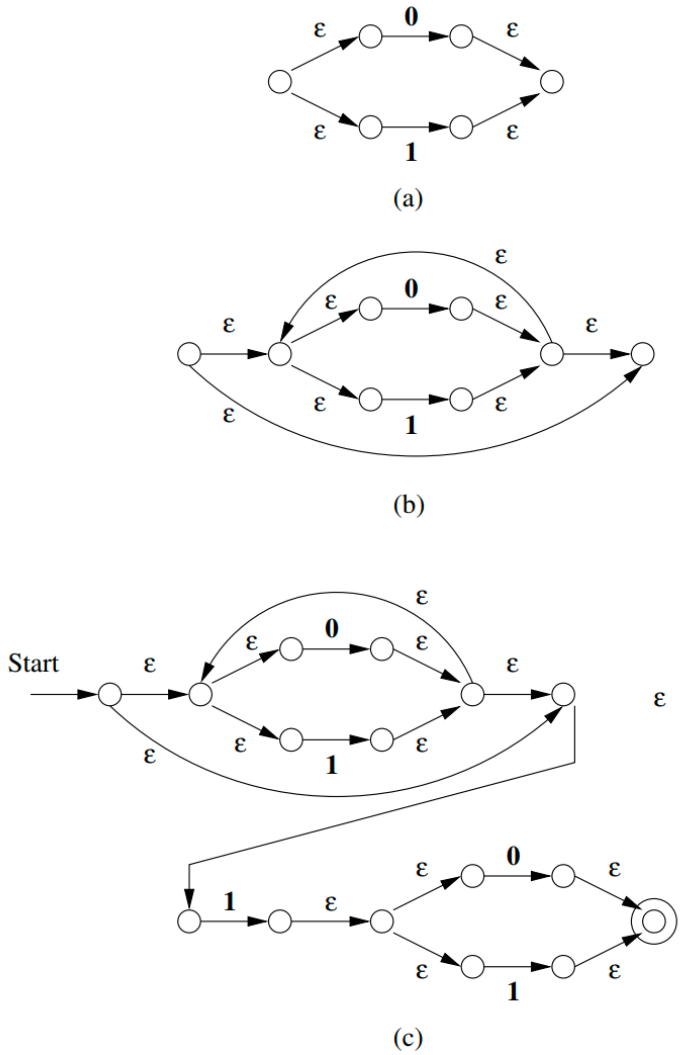
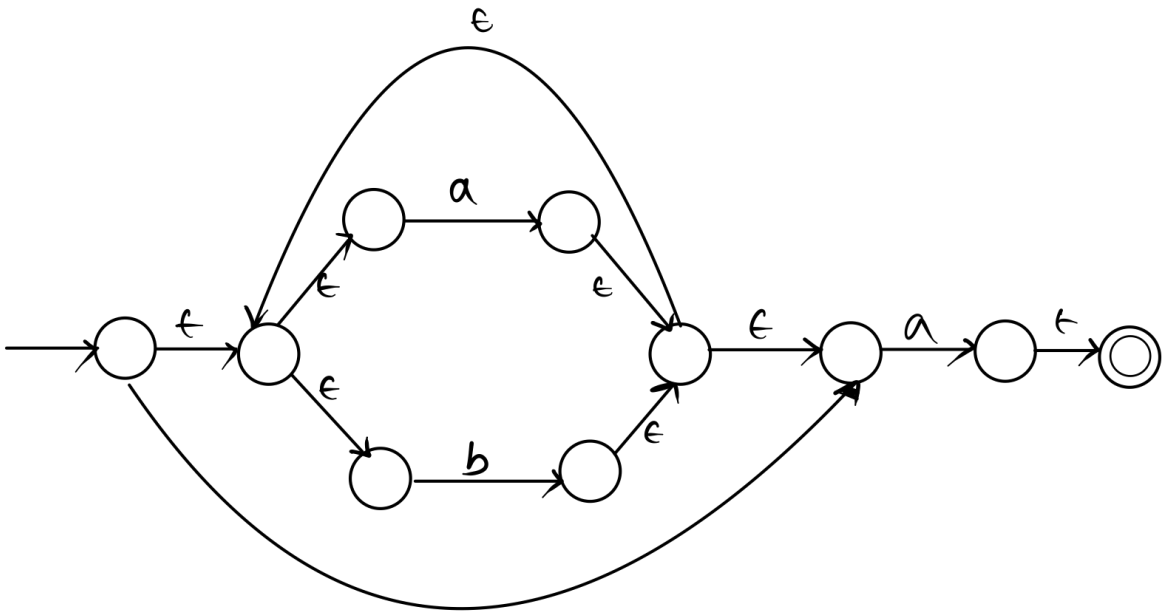


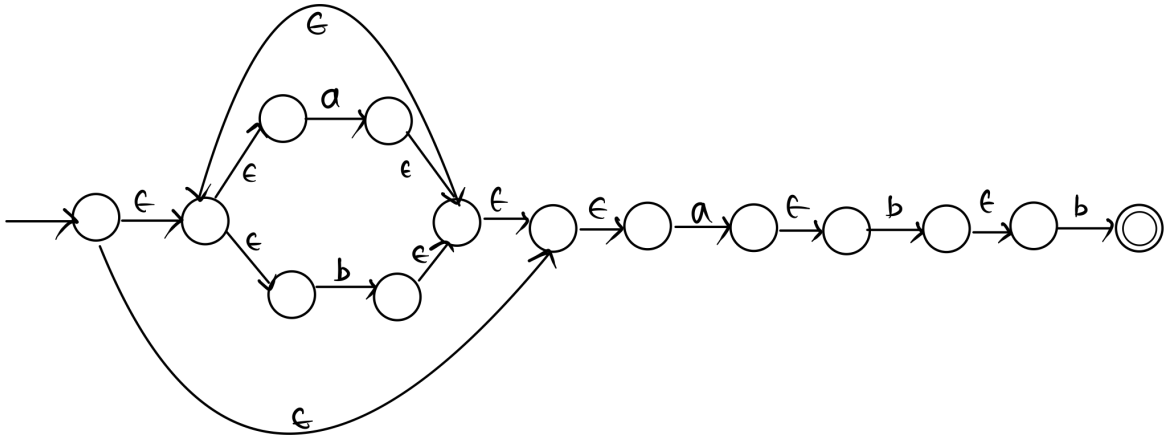
Figure 3.18: Automata constructed for (0+1)*1(0+1)

Converting Regular Expressions to Automata



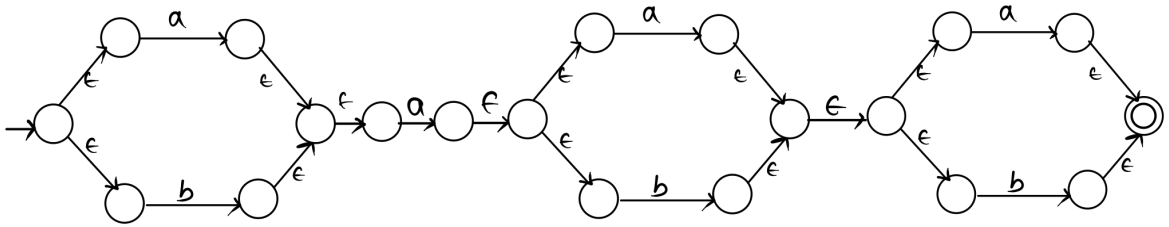
(a+b)*a

Converting Regular Expressions to Automata



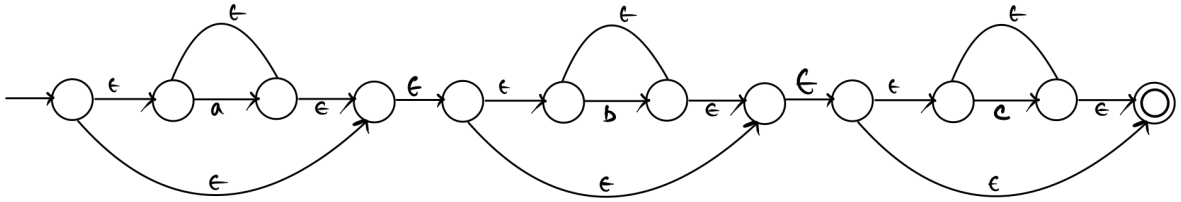
(a+b)*abb

Converting Regular Expressions to Automata



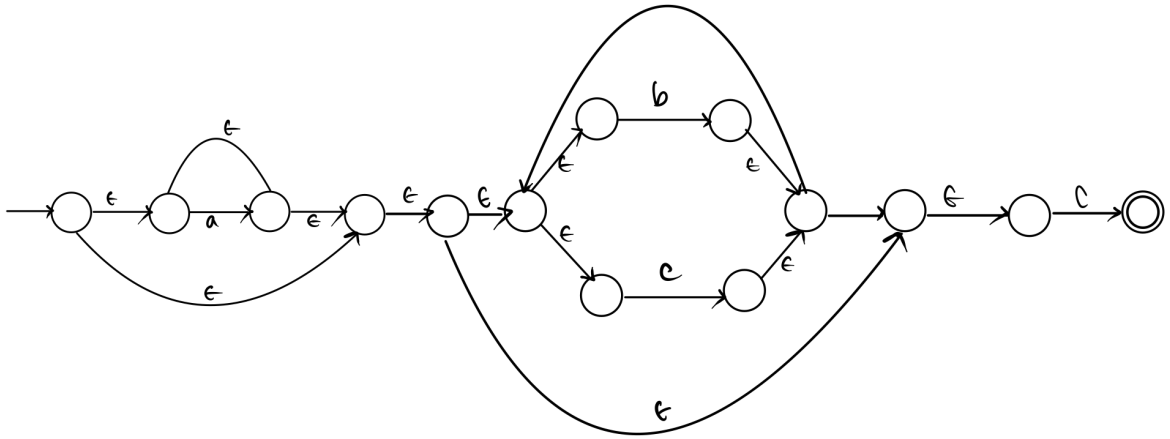
(a+b)a(a+b)(a+b)

Converting Regular Expressions to Automata



a*b*c*

Converting Regular Expressions to Automata



a*(b+c)*c

Exercise

- ab*+c
- b*(a+c)*de
- ab(a+b)*cd*
- (a+b)(b+c)*
- abc(b+c)*d
- 01*
- (0+1)01
- 00(0+1)*

Applications of Regular Expressions

1. **Text Search and Extraction:**
    - **Search for patterns:** Regex can be used to find specific patterns, like phone numbers, email addresses, or specific words within large bodies of text.
    - **Extract data:** Regex allows extraction of structured information, such as dates, credit card numbers, or URLs from unstructured text.
2. **Data Validation:**
    - **Email validation**: Regex is commonly used to ensure an email address is correctly formatted (e.g., someone@example.com).
    - **Phone number validation**: Regex is also used to verify if phone numbers follow a correct format, accounting for different country codes.
    - **Password validation**: You can use regex to ensure passwords meet complexity rules (e.g., minimum length, special characters, and numbers).

Applications of Regular Expressions

Text Search in Databases:

- **Pattern-based search**: Many databases support regex for querying, allowing more sophisticated search patterns compared to simple string matching.
- **Full-text search**: Regex can be used in SQL queries to search through large amounts of text stored in databases.

Security and Malware Detection:

- **Detecting malicious patterns**: Regular expressions can be used to find signatures of known malware or suspicious URLs in logs or web traffic.
- **Intrusion detection**: Regex can be used to identify unusual patterns in network traffic or logs that may indicate an attempted breach