Chapter 3: Regular Languages and Regular Grammar

Dr. Yuan Yao

University of Nottingham Ningbo China (UNNC)

Learning Outcomes

Learning outcomes

At the conclusion of this chapter, the students are expected to be able to:

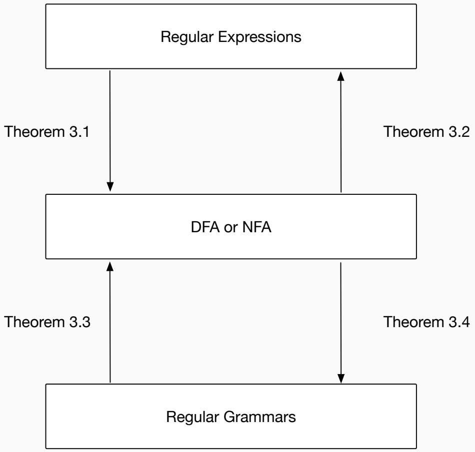
- Identify the language associated with a regular expression.
- Find a regular expression to describe a given language.
- Construct a nondeterministic finite automaton to accept the language denoted by a regular expression.
- Identify whether a particular grammar is regular.
- Construct regular grammars for simple languages.
- Construct an NFA that accepts the language generated by a regular grammar.
- Construct a regular grammar that generates the language accepted by a finite automaton.

Introduction

Introduction to Regular Languages 正则语言

- We defined regular languages as those that can be accepted by finite automata.
- In this chapter, we will discuss two other methods for describing regular languages:
  - Regular expressions.
  - Regular grammars.
- On the plus side, regular expressions are easy to learn because of their similarity to arithmetic expressions.
- On the negative side, there is no obvious way of extending them to the more complicated classes of languages we will discuss later.
- Regular grammars, on the other hand, are a special case of more general grammars which we will encounter again in this course.

- FAs, regular expressions, and regular grammars are equivalent.
- Therefore, we can choose whichever method is most appropriate for the situation at hand.
- We will discuss algorithms for converting any of these forms to another later in this chapter.



Regular Expressions

Regular Expressions

- Regular Expressions provide a concise way of describing some languages.
- Regular Expressions are defined recursively. For any alphabet Σ:
  - Primitive regular expressions:
    - the empty set $\emptyset$.
    - the empty string λ
    - any symbols $a \in \Sigma$
  - If r_1 and r_2 are regular expressions, then so are:
    - the union: $r_1 \cup r_2$
    - the concatenation: $r_1 r_2$
    - the star-closure: r_1^*
    - parenthesised expression: (r_1)
  - Any string resulting from a **finite** number of these operations on primitive regular expressions is also a regular expression.

Languages Associated with Regular Expressions

- A regular expression r denotes a language $L(r)$.
- Assuming that r_1 and r_2 are regular expressions, then:
  - The regular language $\emptyset$ denotes the empty set.
  - The regular language λ denotes the set $\{\lambda\}$.
  - For any a in the alphabet Σ, the regular expression a denotes the set $\{a\}$.
  - The regular expression $r_1 + r_2$ denotes $L(r_1) \cup L(r_2)$, e.g., $a + b$ means $\{a, b\}$.
  - The regular expression $r_1 r_2$ denotes $L(r_1)L(r_2)$, e.g., ab means $\{ab\}$.
  - The regular expression r_1^* denotes $(L(r_1))^*$, e.g., a^* means $\{\lambda, a, aa, \ldots\}$.
  - The regular expression (r_1) denotes $L(r_1)$.

Exercise: Regular Expression

- What is the language $L(ab^* + c)$?

Determining the Language Denoted by a Regular Expression

- By combining regular expressions using the given rules, arbitrarily complex expressions can be constructed.
- In applying operations, we observe the following **precedence rules**:
  - star closure precedes concatenation:
    - Example: ab^* should be interpreted as $a(b)^*$ rather than $(ab)^*$.
    - Thus, $L(ab^*) = \{a, ab, abb, \dots\}$.
  - concatenation precedes union:
    - Example: $ab + c$ should be interpreted as $(ab) + c$ rather than $a(b + c)$.
    - Thus, $L(ab^* + c) = \{c, a, ab, abb, \dots\}$.
  - Parentheses are used to override the normal precedence of operators.
    - Hence, the language $\{ab, ac\}$ is generated by the regular expression $a(b + c)$.

Sample Regular Expressions and Associated Languages

- $(ab)^*$
- $a + b$
- $(a + b)^*$
- $a(bb)^*$
- $a^*(a + b)$
- $(aa)^*(bb)^*b$
- $(0 + 1)^*00(0 + 1)^*$

Equivalence of Regular Expressions

- Two regular expressions are equivalent if they denote the same language.
- For example, $(a+b)^*$ and $(a^*b^*)^*$ are equivalent, because:
  - $L((a+b)^*) = L((a^*b^*)^*) = \{a, b\}^*$
- Another interesting case, what are the languages for the following regular expression?
  - $r_1 = (1^*011^*)^*(0+\lambda) + 1^*(0+\lambda)$
  - $r_2 = (1+01)^*(0+\lambda)$

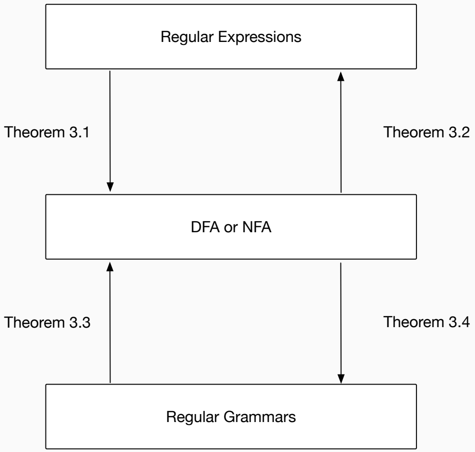
Application

- One of the most widely accessible applications of regular expressions is in search and pattern matching.
- Any non-trivial editor provides search using regular expressions, e.g., Emacs, Vim...

Connections Between Regular Expression and Regular Language

Regular Expression and NFA

- **Theorem 3.1**: For any regular expression r, there is a nondeterministic finite automaton that accepts the language denoted by r.
- Since NFA and DFA are equivalent, for any regular expression r, the language $L(r)$ is also regular.
- How to prove this theorem?



A Constructive Proof

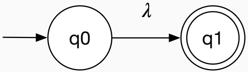
- Construction of an NFA to accept a language $L(r)$ where r is a regular expression.
- We start with **primitive regular expressions**:
- Draw a NFA with two states for the following regular expressions:
  - the empty set.
  - the empty string.
  - Any individual symbol $a \in \Sigma$.

A Constructive Proof

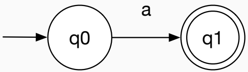
- Construction of an NFA to accept a language $L(r)$ where r is a regular expression.
- We start with **primitive regular expressions**:
- Draw a NFA with two states for the following regular expressions:
  - (a) the empty set.
  - (b) the empty string.
  - (c) any individual symbol $a \in \Sigma$.

- **(a)** 空集（没有任何有效路径）。
- **(b)** 空字符串（从初始状态 q_0 通过 λ-转换到终态 q_1）。
- **(c)** 任意单个符号 $a \in \Sigma$(从 q_0 通过输入 a 转换到终态 q_1)。

(a)



(b)



(c)

A Constructive Proof

- Before going forward, we need to prove the following claim:
- **Claim:** for every NFA with arbitrary number of final states, there is an equivalent NFA with only one final state.

通常的构造方法是：

1. **添加一个新的终态** q_f，使其成为唯一的终态。
2. **对于原 NFA 中的所有终态**，增加一条 λ**-转换（ε转换）**到 q_f。
3. 这样，所有原来的终态仍然是可达的，但最终都归结到唯一的终态 q_f。

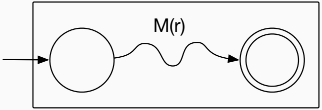
A Constructive Proof

- Before going forward, we need to prove the following claim:
- **Claim:** for every NFA with arbitrary number of final states, there is an equivalent NFA with only one final state.

- **Hint:** Introduce a new final state p_f. For every state $q \in F$, add a λ-transition from q to q_f, i.e., $\delta(q, \lambda) = \{p_f\}$. Make p_f the only final state. Then prove that $\delta^*(q_0, w) \in F$ in the original NFA if and only if $\delta^*(q_0, w) = \{p_f\}$ after the modification. So, these two NFAs are equivalent.
- How about the same claim but for DFA? is it true?

A Constructive Proof

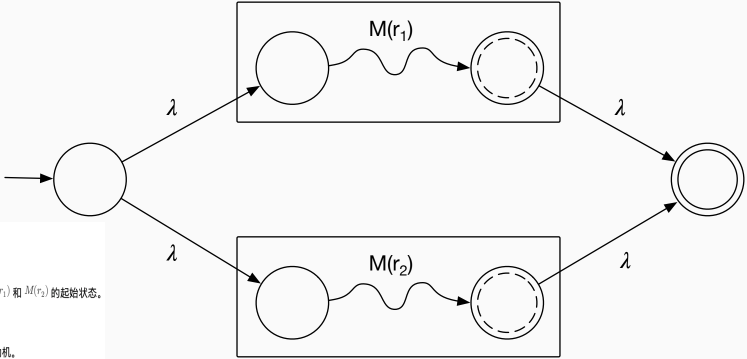
- We could therefore use the following representation for automata $M(r)$ that accept the language $L(r)$ denoted by a regular expression r.



- Assume r_1 and r_2 are two regular expressions, then how to construct an automaton that accept $L(r_1 + r_2)$?

A Constructive Proof

- Assume r_1 and r_2 are two regular expressions, then how to construct an automaton that accept $L(r_1 + r_2)$?



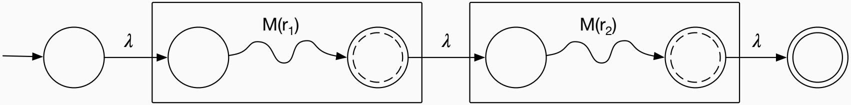
1. **创建新的起始状态** (左侧的圆圈)。
2. **使用 ε-转换**：
   - 让新起始状态通过 **ε-转换** (λ-转换) 分别连接到 $M(r_1)$ 和 $M(r_2)$ 的起始状态。
3. **两个正则表达式各自独立运行**：
   - $M(r_1)$ 和 $M(r_2)$ 代表能够识别 $L(r_1)$ 和 $L(r_2)$ 的自动机。
   - 它们的终态也通过 ε-转换连接到 **新的唯一终态** (右侧的圆圈)。
4. **等价性**：
   - 任何属于 $L(r_1)$ 或 $L(r_2)$ 的字符串都能被这个 NFA 接受。

A Constructive Proof

- Assume r_1 and r_2 are two regular expressions, then how to construct an automaton that accept $L(r_1 r_2)$, i.e., the concatenation?

A Constructive Proof

- Assume r_1 and r_2 are two regular expressions, then how to construct an automaton that accept $L(r_1 r_2)$, i.e., the concatenation?



1. **创建新的起始状态**（左侧的圆圈）。
2. **使用 ε-转换**：
   - 新起始状态通过 **ε-转换**（λ-转换）指向 $M(r_1)$ 的起始状态。
   - $M(r_1)$ 的终态通过 **ε-转换** 指向 $M(r_2)$ 的起始状态。
   - $M(r_2)$ 的终态通过 **ε-转换** 指向最终的唯一终态（右侧的圆圈）。
3. **工作原理**：
   - 只有当 **一个字符串能够依次通过 $M(r_1)$ 和 $M(r_2)$ 的自动机** 时，它才会被接受。
   - 这符合 **正则表达式连接运算的定义。**

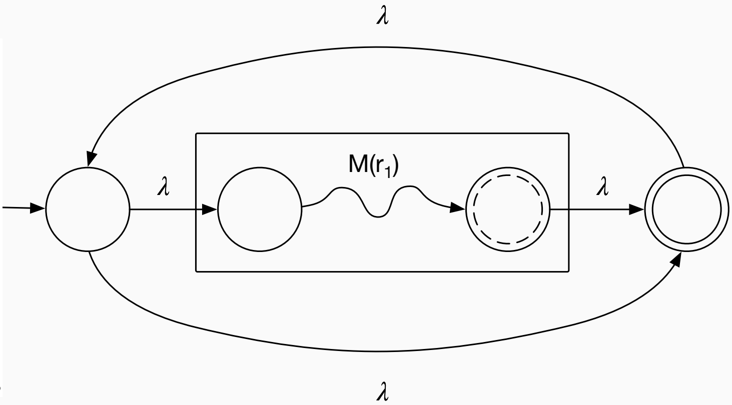
A Constructive Proof

- Assume r_1 is a regular expressions, then how to construct an automaton that accept $L(r_1^*)$?

A Constructive Proof

- Assume r_1 is a regular expressions, then how to construct an automaton that accept $L(r_1^*)$?

1. **创建新的起始状态和终态**：
   - 让新的起始状态通过 **ε-转换** (λ-转换) 直接指向：
     1. $M(r_1)$ 的起始状态 (表示至少执行一次 r_1)。
     2. 终态 (表示可以直接接受空字符串)。
2. **循环结构**：
   - $M(r_1)$ 的终态通过 **ε-转换** 指向：
     1. 终态 (表示执行一次 r_1 后可以接受)。
     2. $M(r_1)$ 的起始状态 (表示可以重复执行 r_1)。
3. **等价性**：
   - 这样构造的 NFA 可以接受 **任意数量的** r_1**(包括零次)**，符合 r_1^* 的定义。



Exercise: Regular Expression to NFA

- Given the following regular expression

$$r = (a + bb)^*(ba * +\lambda)$$

- Construct a DFA that accept $L(r)$.

Regular Expressions for Regular Languages

- So far, we have learnt that, for any regular expression r, the language $L(r)$ is a regular language.
- What about the other direction?
- **Question:** Is it true that for any regular language L, there exists a regular expression r such that $L = L(r)$? Yes

- **Theorem 3.2** Let L be a regular language. Then there exists a regular expression r such that $L = L(r)$.
- How to prove it?

Regular Expressions for Regular Languages

- So far, we have learnt that, for any regular expression r, the language $L(r)$ is a regular language.
- What about the other direction?
- **Question:** Is it true that for any regular language L, there exists a regular expression r such that $L = L(r)$?

- **Theorem 3.2** Let L be a regular language. Then there exists a regular expression r such that $L = L(r)$.
- How to prove it?
- To prove theorem 3.2, we need an algorithm which, given a finite automaton M, returns a regular expression r such that $L(r) = L(M)$.
- We are not going to introduce the details, please refer to the textbook. This algorithm is also implemented in JFLAP.

Regular Grammar

Regular Grammar

- The production rule for the following two languages are as follows:
  - $L_1 = \{a^n b^n | n \geq 0\}$

$$S \rightarrow aSb \quad | \quad \lambda$$

  - $L_2 = \{a^n b^m | n, m \geq 0\}$

$$\begin{array}{lcl} S & \rightarrow & aS \quad | \quad A \\ A & \rightarrow & bA \quad | \quad \lambda \end{array}$$

- **Question:** Can we identify the type of grammars that generates **regular language**?
- What's the difference between the production rules above?

Regular Grammar

- The production rule for the following two languages are as follows:
  - $L_1 = \{a^n b^n | n \geq 0\}$
    $$S \rightarrow aSb \quad | \quad \lambda$$
  - $L_2 = \{a^n b^m | n, m \geq 0\}$
    $$\begin{aligned} S &\rightarrow aS \quad | \quad A \\ A &\rightarrow bA \quad | \quad \lambda \end{aligned}$$
- What's the difference between the production rules above?
  - in the production $S \rightarrow aSb$, symbols appear on both sides of S.
  - whereas in the production rules of the second grammar, symbols are added only to one side (in this case, the left side) of the variable.

Right-Linear and Left-Linear Grammar

- **Right-linear Grammar:** a grammar $G = (V, T, S, P)$ is said to be right-linear if all productions in P are of the form:
  $$\begin{aligned} A &\rightarrow xB \\ A &\rightarrow x \end{aligned}$$
  where $A, B \in V$ and $x \in T^*$
- **Left-linear Grammar:** a grammar $G = (V, T, S, P)$ is said to be left-linear if all productions in P are of the form:
  $$\begin{aligned} A &\rightarrow Bx \\ A &\rightarrow x \end{aligned}$$
  where $A, B \in V$ and $x \in T^*$
- **We say that a grammar is regular if it is either right-linear or left-linear.**

Exercise: Regular Grammar

- Given the following grammar G:

$$G = (\{S\}, \{a, b\}, S, P)$$

  where P is defined as follows:

$$S \to abS \quad | \quad a$$

- Is the above grammar left-linear or righ-linear or neither? right-linear
- Can you give a regular expression for the regular language generated by the above grammar?

Right-Linear Grammars Generate Regular Languages

- **Theorem 3.3** Let $G = (V, T, SP)$ be a right-linear grammar. Then $L(G)$ is a regular language.
- We could prove this theorem constructively.
- In fact, there is an algorithm for constructing an NFA to accept the language generated by a given right-linear grammar G.

Right-Linear Grammars Generate Regular Languages

- How to construct an NFA to accept the language generated by a given right-linear grammar G:
  - Label the NFA start state with S and a final state V_f.
  - For every variable symbols V_i, create an NFA state and label it V_i.
  - For each production of the form $A \to aB$, label a transition from state A to B with symbol a.
  - For each production of the form $A \to a$, label a transition from state A to V_f with symbol a.
  - Note: you need to add intermediate states for productions with more than one terminal on the right-hand side.
- **Question:** Can you see why in general we get an NFA, rather than a DFA?

Exercise: Construct an NFA to accept the given regular grammar

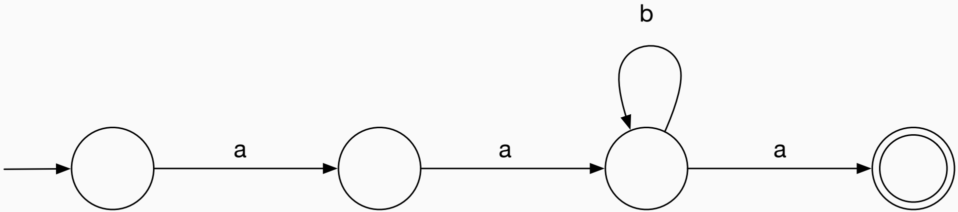
- Given the regular grammar $G = (\{V_0, V_1\}, \{a, b\}, V_0, P)$, where P is defined as follows:

$$
\begin{array}{lcl}
V_0 & \rightarrow & aV_1 \\
V_1 & \rightarrow & abV_0 \quad | \quad b
\end{array}
$$

Right-Linear Grammars for Regular Languages

- **Theorem 3.4:** If L is a regular language on the alphabet Σ, then there exists a right-linear grammar $G = (V, T, S, P)$ such that $L = L(G)$.
- **How to prove it?**
  - There is an algorithm that, given any DFA M accepting a regular language L, constructs a right-linear grammar G which generates the same language:
    - Each state in the DFA corresponds to a variable symbol in G.
    - For each DFA transition from state A to state B labeled with symbol a, there is a production of the form $A \to aB$ in G.
    - For each final state F_i in the DFA, there is a corresponding production $F_i \to \lambda$ in G.

- Given the following finite automaton M, write a regular grammar G such that $L(M) = L(G)$.



Notes: Regular Grammars

- Remember that a grammar is said to be regular if it is either left-linear or right-linear.
- For simplicity, we stated all our results in terms of right-linear grammars, but very similar arguments prove the corresponding results for left-linear grammars.
- **Caution:** A grammar is regular if either all its productions are left-linear, or they are all right-linear, but not a combination of both.
- For example, consider the grammar $G = (\{S, A, B\}, \{a, b\}, S, P)$ with productions:

$$\begin{aligned} S &\rightarrow A \\ A &\rightarrow aB \mid \lambda \\ B &\rightarrow Ab \end{aligned}$$

- **Question:** What is the language $L(G)$?

Application: Text Editing and Pattern Matching

- We have already mentioned that, almost any sophisticated text editor (such as Emacs, Vim, Netbeans, etc.) allows search by regular expressions.

- **Question:** How is it possible to perform search using a regular expression as input?

- The text editor goes through the following steps:
  1. Convert the regular expression into an equivalent NFA (Theorem 3.1).
  2. Convert the NFA to an equivalent DFA.
  3. Minimize the DFA (which we do not discuss in this module).
  4. Finally, run the DFA over the input for pattern matching.

Application: Compilation and Pattern Matching

- **Question:** How is it possible for a C compiler to check whether a string of symbols is a valid identifier?

- The designers of the compiler must go through the following steps:
  1. Convert the regular grammar to an equivalent NFA (Theorem 3.3).
  2. Convert the NFA to an equivalent DFA.
  3. Minimize the DFA (which we do not discuss in this module).
  4. Finally, incorporate the DFA into the compiler for pattern matching.
- **As can be seen, the relatively simple results that we have discussed so far are used in practice for crucial applications (e.g., pattern matching in text editors and compilers).**