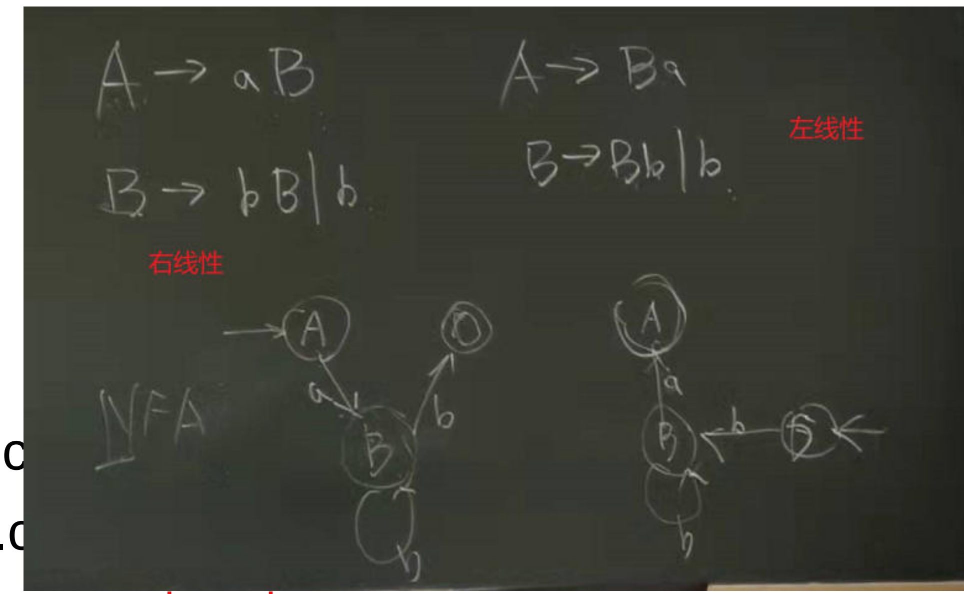
Principles of
Compiler Construction

空串：叶子节点均为‘空’
DFA识别空串：iff.开始状态即为终止状态

A->Ba L(B)多走a就到了A



Prof. Wen-jun LI

School of Computer Sc

lnslwj@mail.sysu.edu.c

有穷集变小，无穷集变有穷集：商集 eg.reduced DFA

lexical
英 [ˈleksɪk(ə)l] 美 [ˈleksɪkl]
adj. 词汇的；（与）词典（有关）的；词典编纂的

Lecture 3. Lexical Analysis

scanner

1. Introduction
2. Scanner Construction
3. Lexical Specification
4. Finite Automata
5. Transformation and Equivalence
6. Limits of Regular Languages
7. Lexical Analysis in Practice

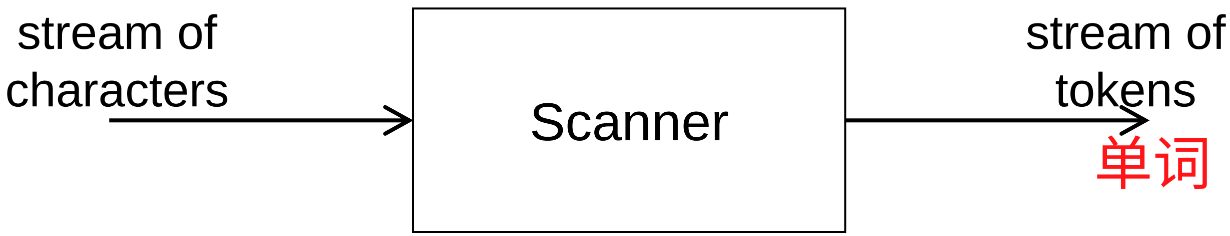
1. Introduction

n. 管子，管道

- Software Architecture: Pipes and Filters

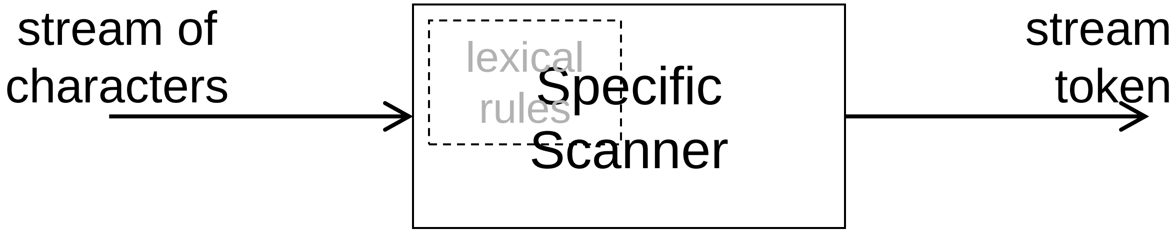
软件体系结构

思想：分而治之
缺点：拆分成过多的部分导致效率慢



Structure of a Scanner

- Implicit lexical rules

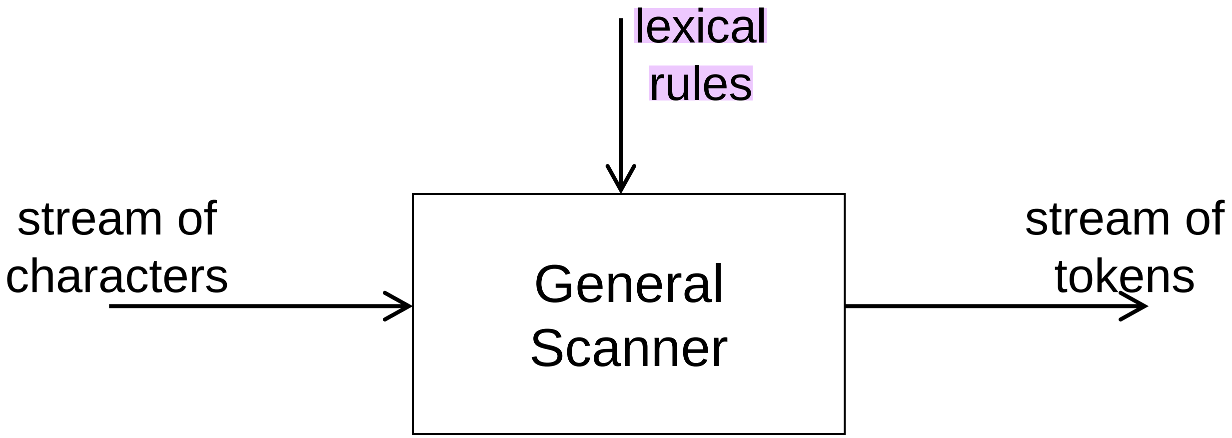


Specific to some predefined language.

interpretation
英 [ɪnˌtɜːprəˈteɪʃn] 美 [ɪnˌtɜːrprəˈteɪʃn]
n. 解释，理解；表演，演奏，艺术处理；口译

Structure of a Scanner (cont')

- Explicit lexical rules (interpretation model)
  解释执行

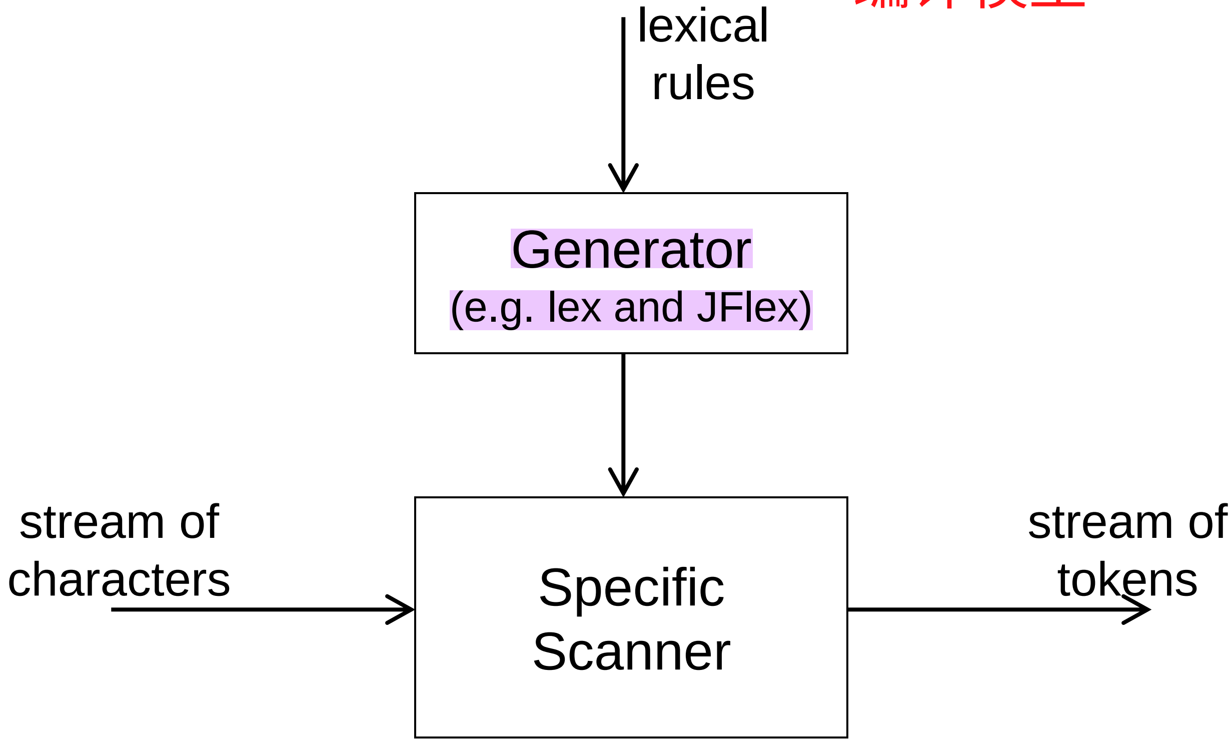


每次都要查询rules，效率低

No hard-coding of language-specific code.

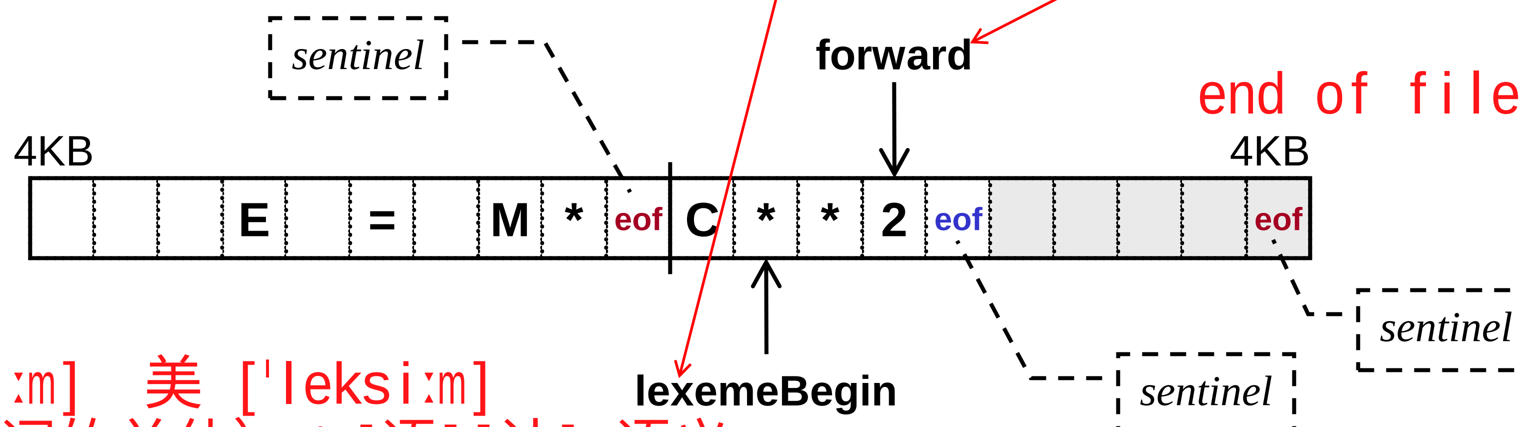
Structure of a Scanner (cont')

- Explicit lexical rules (compilation model)
  编译模型



Input Buffering

- Most runtime of the front end of a compiler is spent on scanning.
  - But the most time consuming stage is optimization. 不同编译器差别大的地方
- Buffer Pairs
  - 2-buffer scheme: each buffer is of size N.
  - Avoid overwriting: $|lexeme| + |lookaheads| \leq N$



lexeme
英 [ˈleksiːm] 美 [ˈleksiːm]
n. 词位（词的单位）；[语][计] 语义

3.2.1 缓冲区对

由于在编译一个大型源程序时需要处理大量的字符，处理这些字符需要很多的时间，因此开发了一些特殊的缓冲技术来减少用于处理单个输入字符的时间开销。一种重要的机制就是利用两个交替读入的缓冲区，如图 3-3 所示。

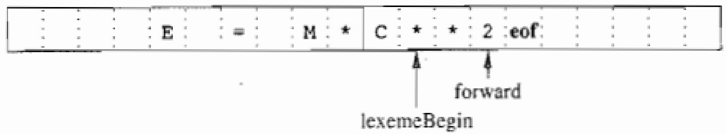


图 3-3 使用一对输入缓冲区

每个缓冲区的容量都是 N 个字符，通常 N 是一个磁盘块的大小，如 4096 字节。我们可以使用系统读取命令一次将 N 个字符读入到缓冲区中，而不是每读入一个字符调用一次系统读取命令。如果输入文件中的剩余字符不足 N 个，那么就会有一个特殊字符(用 **eof** 表示)来标记源文件的结束。这个特殊字符不同于任何可能出现在源程序中的字符。

程序为输入维护了两个指针：

1) `lexemeBegin` 指针：该指针指向当前词素的开始处。当前我们正试图确定这个词素的结尾。

2) `forward` 指针：它一直向前扫描，直到发现某个模式被匹配为止。做出这个决定所依据的策略将在本章的其余部分中讨论。

一旦确定了下一个词素，`forward` 指针将指向该词素结尾的字符。词法分析器将这个词素作为某个返回给语法分析器的词法单元的属性值记录下来。然后使 `lexemeBegin` 指针指向刚刚找到的词素之后的第一个字符。在图 3-3 中，我们看到，`forward` 指针已经越过下一个词素 ** (Fortran 的指数运算符)。在处理完这个词素后，它将会被左移一个位置。

将 `forward` 指针前移要求我们首先检查是否已经到达某个缓冲区的末尾。如果是，我们必须将 N 个新字符读到另一个缓冲区中，且将 `forward` 指针指向这个新载入字符的缓冲区的头部。只要我们从不需要越过实际的词素向前看很远，以至于这个词素的长度加上我们向前看的距离大于 N，我们就决不会在识别这个词素之前覆盖掉这个尚在缓冲区中的词素。

Output Tokens

类型

- 5 kinds for most modern programming languages
  - Identifiers: getBalance, weight, ...
  - Reserved words: IF, ELSE, WHILE, ...
  - Constants: 10, 3.14, -1.26E-5, 'a', "abc", ...
  - Operators: +, -, *, /, <<, ...
  - Punctuation: (,), ;, :, ...

key words可以重定义
reserved words不可以

- Output
  - A Pair of <kind, associatedAttributeValues>
    - Some values are pointers to the symbol table.
  - Symbol (string) table
    - Array, LinkedList, HashSet, TreeSet, ...

scanner生产者，parser消费者

callee caller

Interaction with Parser

- Aim at reducing passes
- 2 levels of abstraction
  - Logically
    - A pipe through which tokens are transferred.
  - Physically
    - A disk file of the token sequence 低效
    - Concurrent threads or co-routines 难，并发
    - Method invocation 方法调用 caller&callee

invocation

英 [ˌɪnvəˈkeɪʃn] 美 [ˌɪnvəˈkeɪʃn]

n. （向神或权威人士的）求助，祈祷；咒语；（仪式或集会开始时的）发言祷文；（法院对另案的）文件调取；（计算机）调用，启用；（法权的）行使

2. Scanner Construction

- Definition of lexical rules: 3 equivalent notations 三种方式等价 L(RE)=L(RG)
  - Expressions describe
    - Regular expression
    - Regular definition
  - Grammars generate
    - Regular grammar
    - Left/right linear grammar
  - Finite automata 有限自动机 recognize
    - Deterministic Finite Automata (DFA) 状态转换唯一
    - Nondeterministic Finite Automata (NFA)

表达能力一样

Begin with Formal Specification

- Specification of tokens 便于人读写
  - Regular definition
    - Hierarchical and brief, suitable for **humans**.
    - Then transferred to regular expressions.
    - Then transferred to finite automata.
  - Regular grammar
    - Compliant with the spec. of syntax rules.
    - Then transferred to finite automata.
- Transferring finite automata
  - 确定化
  - NFA → DFA → Reduced DFA
  - Suitable for implementation on a **machine**.

compliant
英 [kəm'plaɪənt] 美 [kəm'plaɪənt]
adj. 服从的，顺从的；（与规定或标准）符合的，一致的；（物理，医）有可塑性的，柔量的

有穷的集合必然是Regular Language eg.A^n B^n (0<=n<5)
有限状态机实现

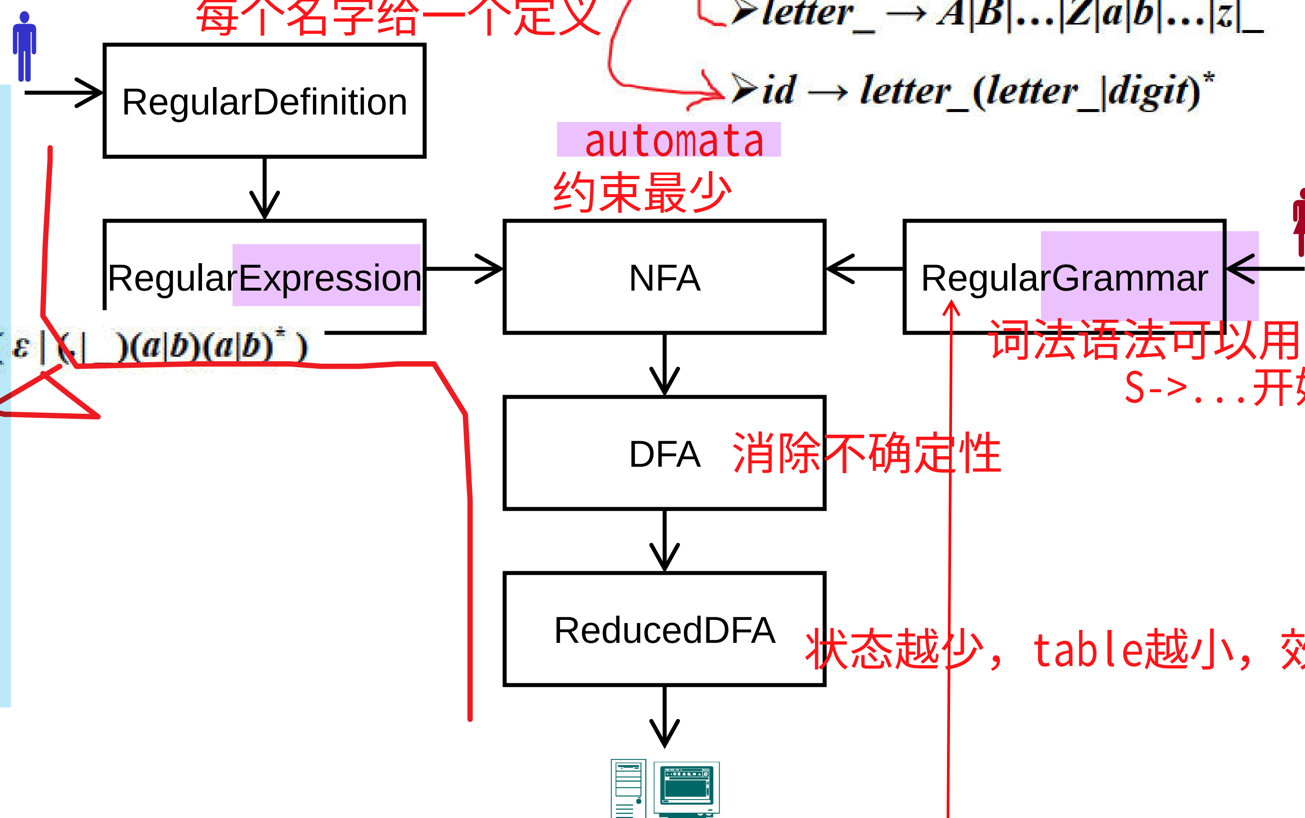
Steps of Construction

前面定义过的后面才能用

- $digit \rightarrow 0|1|2|\ldots|9$
- $letter\_ \rightarrow A|B|\ldots|Z|a|b|\ldots|z|\_$
- $id \rightarrow letter\_(letter\_|digit)^*$

每个名字给一个定义

automata
约束最少



- 例： $r = a(a|b)^*(\varepsilon \mid (.|\_)(a|b)(a|b)^*)$

词法语法可以用同一套规则
S->...开始符号开始

消除不确定性

状态越少，table越小，效率越高

- **正则文法** (*Regular Grammar, RG*)
  - **右线性**(*Right Linear*)**文法**： $A \rightarrow wB$ 或 $A \rightarrow w$
  - **左线性**(*Left Linear*) **文法**： $A \rightarrow Bw$ 或 $A \rightarrow w$

种分类

右侧最多1个非终结符，在同一侧

Programming a Scanner

- Implicit transition diagram
  - A manual approach
  - State transitions are hard-coded in the program.
- Explicit transition diagram
  - A table-driven approach
  - No hard-coding of specific lexical rules
  - Lead to automatic scanner generation.

3. Lexical Specification

- Regular Expression
- Regular Definition
- Regular Grammar
- Transformation and Equivalence

Regular Expression

- Regular expression: constructively defined
  - Basis: L:正则表达式映射成语言
    - $\boldsymbol{\varepsilon}$ is a regular expr; $L(\boldsymbol{\varepsilon}) = \{\varepsilon\}$.
    - **a** is a regular expr if $a \in \Sigma$; $L(\mathbf{a}) = \{a\}$.
  - Induction: if **r** and **s** are regular expressions,
    - $\mathbf{r} \mid \mathbf{s}$ is a regular expr; $L(\mathbf{r} \mid \mathbf{s}) = L(\mathbf{r}) \cup L(\mathbf{s})$.
    - $\mathbf{r}\ \mathbf{s}$ is a regular expr; $L(\mathbf{r}\ \mathbf{s}) = L(\mathbf{r})\ L(\mathbf{s})$.
    - $\mathbf{r}^*$ is a regular expr; $L(\mathbf{r}^*) = (L(\mathbf{r}))^*$.
    - $(\mathbf{r})$ is a regular expr; $L((\mathbf{r})) = L(\mathbf{r})$.
  - Extensions: 仅为书写方便
    - $\mathbf{r}^+ = \mathbf{r}\ \mathbf{r}^* = \mathbf{r}^*\ \mathbf{r}$ $\quad$ $\mathbf{r}^* = \mathbf{r}^+ \mid \varepsilon$ optional的表示
    - $\mathbf{r}^? = \mathbf{r} \mid \varepsilon$
    - $\mathbf{[ace]} = \mathbf{a} \mid \mathbf{c} \mid \mathbf{e}$ $\quad$ $\mathbf{[A–Z]} = \mathbf{A} \mid \mathbf{B} \mid \ldots \mid \mathbf{Z}$

这些都不是Σ里的符号，是为了表达RE

Regular Expression Language

- Regular expression: a language
  - Syntax
    - **a (a | b) b**
    - **a (a | b)***
  - Semantics
    - {aab, abb}
    - {a, aa, ab, aaa, aab, aba, abb, ...}
- Language is an alternative approach to problem solving.

Discussions

- Signature of an operator
  - true: $\rightarrow$ bool
  - +: int $\times$ int $\rightarrow$ int
  - +: real $\times$ real $\rightarrow$ real (overloading)
- What is the signature of the semantic function L in the previous slides ?
  - Syntactic category
  - Semantic category
  - Mapping

Alphabet: Lexical vs. Syntax

- Both scanner and parser are based over an alphabet Σ.
- But the meaning (elements) of Σ is quite different.
  - 词法分析 Scanner: elements in Σ are characters in source programs.
    - ASCII, EBCDIC, Unicode, ...
  - 语法分析 Parser: elements in Σ are tokens generated and passed by the scanner.

Algebraic Laws for Regular Expressions

- $r \mid s = s \mid r$
- $r \mid (s \mid t) = (r \mid s) \mid t$
- $r \mid r = r$
- $r(st) = (rs)t$
- $r(s \mid t) = rs \mid rt$
- $(s \mid t)r = sr \mid tr$
- $r\varepsilon = \varepsilon r = r$
- $r^*r^* = r^*$
- $r^* = \varepsilon \mid r \mid rr \mid \ldots$
- $(r^*)^* = r^*$
- $rr^* = r^*r$
- $(r^* \mid s^*)^* = (r^*s^*)^*$
- $(r^*s^*)^* = (r \mid s)^*$
- $(rs)^*r = r(sr)^*$
- $(r \mid s)^* = (r^*s)^*r^*$

Discussion: what does it mean that two regular expressions are equivalent ?

Binding to Practice

- Identifiers in Pascal

$(a \mid b \mid \ldots \mid z \mid A \mid B \mid \ldots \mid Z)\ (\ (a \mid b \mid \ldots \mid z \mid A \mid B \mid \ldots \mid Z) \mid (0 \mid 1 \mid \ldots \mid 9)\ )^*$

- Hard to read and write

Regular Definition

- In the form of
  - $d_1 \rightarrow r_1$
  - $d_2 \rightarrow r_2$
  - ...
  - $d_n \rightarrow r_n$
    - where $d_i \notin \Sigma \wedge (d_i = d_j \Rightarrow i = j)$, and
    - r_i is a regular expr over $\Sigma \cup \{d_1, d_2, \ldots, d_{i-1}\}$

Binding to Practice

- Identifiers in Pascal (extended)
  - letter_ $\rightarrow$ A | B | … | Z | a | b | … | z | _
  - digit $\rightarrow$ 0 | 1 | … | 9
  - id $\rightarrow$ letter_ (letter_ | digit)*
- Real numbers in Pascal
  - digit $\rightarrow$ 0 | 1 | … | 9
  - digits $\rightarrow$ digit digit*
  - optionalFraction $\rightarrow$. digits | ε
  - optionalExponent $\rightarrow$ (E (+ | - | ε) digits) | ε
  - real $\rightarrow$ digits optionalFraction optionalExponent
- Hierarchical representation of regular expression

Binding to Practice (cont')

- Writing a regular definition with extended operators
  - digit $\rightarrow$ [0–9]
  - digits $\rightarrow$ digit^{+}
  - real $\rightarrow$ digits (. digits)$^{?}$ (E [+-]$^{?}$ digits)$^{?}$

？表示可有可无

Note the difference between '–' and '-'

Transforming between Regular Definitions and Regular Expressions

- Regular definition → regular expression
  - Top-down, stepwise substitution
  - **Discussion**: does the substitution process always terminate ?
- Regular expression → regular definition
  - Simply add a new symbol and "→".
- Equivalence
  - Defined by the language they denote
  - Bi-direction

在表达式基础上给个名字

Transforming Regular Definitions to Regular Grammars

正则表达式：更直观

词法语法同个风格

- Difficulties arise from the closure operator
  - Transform to a left/right recursive production.
- In regular definition
  - id $\rightarrow$ letter (letter | digit)*
- Introduce a right part
  - id $\rightarrow$ letter rid
  - rid $\rightarrow$ rid (letter | digit) | ε (not permitted in Grammar)
    that is, rid $\rightarrow$ rid letter | rid digit | ε
  - or right-linear, rid $\rightarrow$ letter rid | digit rid | ε

Class 3 Grammars

- Right Linear (Regular) Grammar
  - $A \rightarrow aB$ or $A \rightarrow a$, where $A, B \in N \wedge a \in \Sigma \cup \{\varepsilon\}$.
- Left Linear Grammar
  - $A \rightarrow Ba$ or $A \rightarrow a$, where $A, B \in N \wedge a \in \Sigma \cup \{\varepsilon\}$.
- Extended Forms
  - Right Linear (Regular) Grammar
    $A \rightarrow \alpha B$ or $A \rightarrow \alpha$, where $A, B \in N \wedge \alpha \in \Sigma^*$.
  - Left Linear Grammar
    $A \rightarrow B\alpha$ or $A \rightarrow \alpha$, where $A, B \in N \wedge \alpha \in \Sigma^*$.

4. Finite Automata recognize

- 3 types of finite automata
  - DFA (Deterministic Finite Automata)
  - NFA (Nondeterministic Finite Automata)
  - ε-NFA (Nondeterministic Finite Automata with empty transitions)
- Languages defined by a finite automaton
- Representation of finite automata
  - For humans: Transition Diagrams
  - For computers/machines: Transition Tables

Formal Definition

- A finite automaton is a 5-tuple: $M = (\Sigma, S, \Delta, s_0, F)$, where
  - Σ is the input alphabet. 小写
  - 字符不同 $S \cap \Sigma = \varnothing$ is a **finite** set of states. 大写
  - Δ is a transition function.
  - $s_0 \in S$ is the start (initial) state.
  - $F \subseteq S$ is a set of final (accepting) states.
- DFA, NFA, and ε-NFA only differ in
  - For a DFA, $\delta: S \times \Sigma \to S$ 去到唯一状态
  - For an NFA, $\delta: S \times \Sigma \to 2^S$ 去到多个状态
  - For an ε-NFA, $\delta: S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$

一个输入

可以没有输入就转换

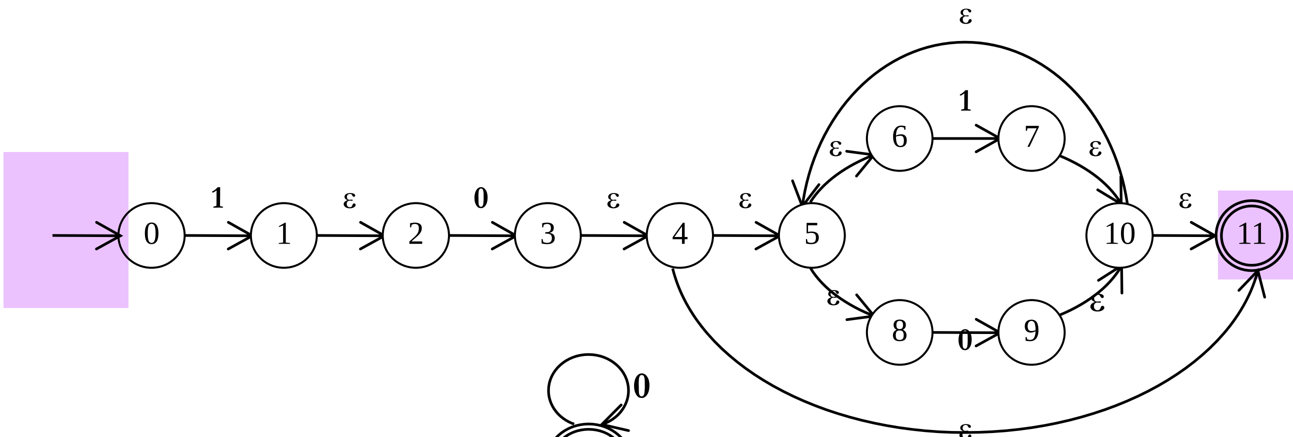
Language Defined by an FA

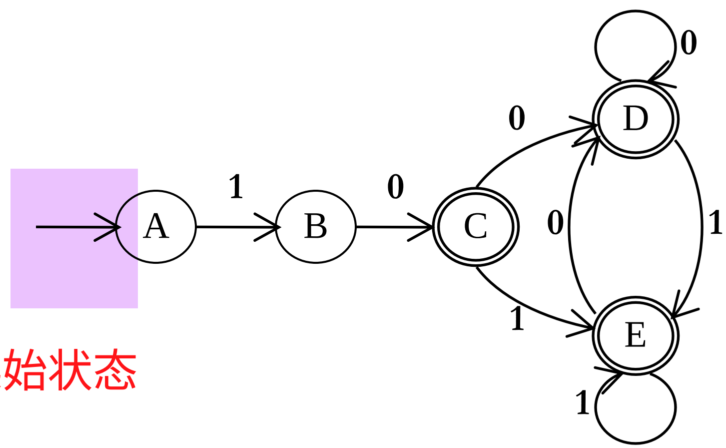
- **Overloading** the transition function
  - For any $a \in \Sigma$, $s \in S$ and $\omega \in \Sigma^*$,
    - $\delta(s, \varepsilon) = s$
    - $\delta(s, \omega a) = \delta(\delta(s, \omega), a)$
- Given a finite automaton M,
  - $L(M) = \{\omega \mid \omega \in \Sigma^* \wedge \exists s \in F.\ \delta(s_0, \omega) = s\}$

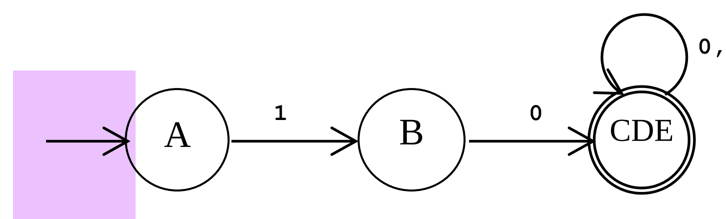
终态

Transition Diagrams

- Graphical representation of FA



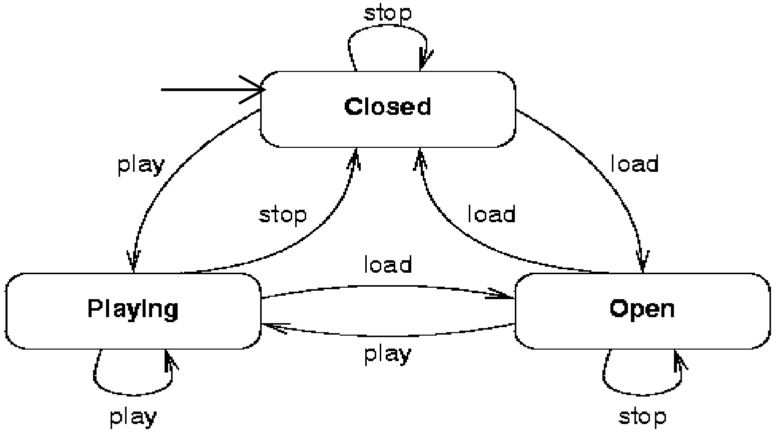




标注起始状态

Commonly Used in Practice

- Description of the state transition of a stateful object



- UML (Unified Modeling Language)
  - De facto standard for object-oriented modeling and design.
  - Statechart of a class.

Internal Representation

- Transition table
  - Rows
    - DFA, NFA, and ε-NFA: State S
  - Columns
    - DFA and NFA: Alphabet Σ
    - ε-NFA: $\Sigma \cup \{\varepsilon\}$
  - Cells
    - DFA: an element of S
    - NFA and ε-NFA: a subset of S
- That is why NFA & ε-NFA are not suitable for implementation.

5. Transformation and Equivalence

- Regular Grammar to ε-NFA
- Regular Expression to ε-NFA
- Determination (of ε-NFA)
- Reduction (of DFA)

Regular Grammar to ε-NFA

- Algorithm
  - Add a new final state *f*
  - For any $a \in \Sigma \cup \{\varepsilon\}$ and $A, B \in N$,
    - If $A \to a$ where $a \in \Sigma \cup \{\varepsilon\}$, let $\delta(A, a) = f$
    - If $A \to aB$, let $\delta(A, a) = B$ 用弧表示

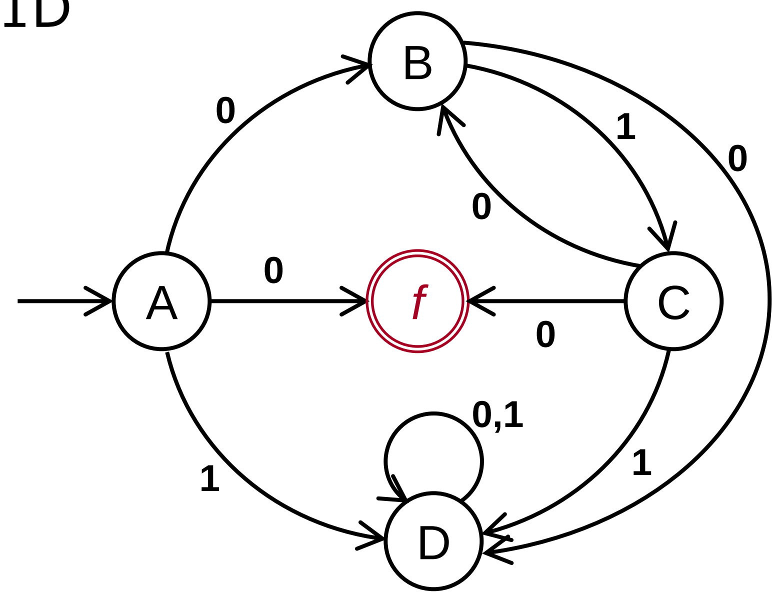
An Example

- Given a regular grammar

$$A \to 0 \mid 0B \mid 1D$$
$$B \to 0D \mid 1C$$
$$C \to 0 \mid 0B \mid 1D$$
$$D \to 0D \mid 1D$$

右线性！！

- We have an equivalent ε-NFA



非终结符作为状态
再加上终结状态

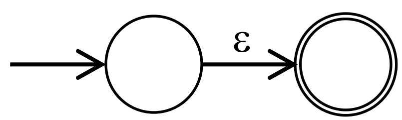
左线性？

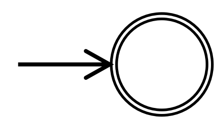
Regular Expression to ε-NFA

- Thompson's construction (McNaughton-Yamada-Thompson algorithm)
  - A constructive approach corresponding with the constructive definition of regular expressions
  - Basis:

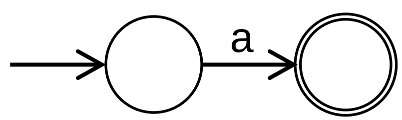
按照算法执行

    - ε



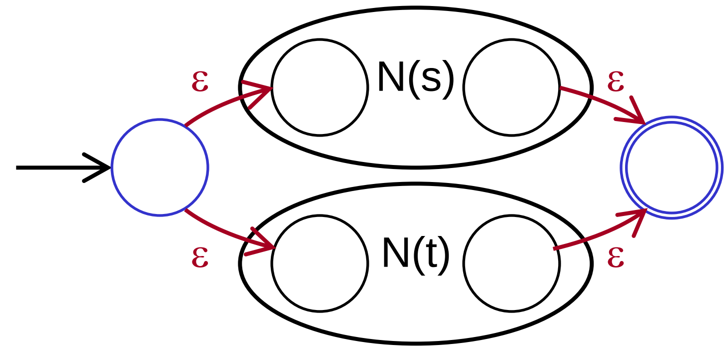


    - a $\in \Sigma$

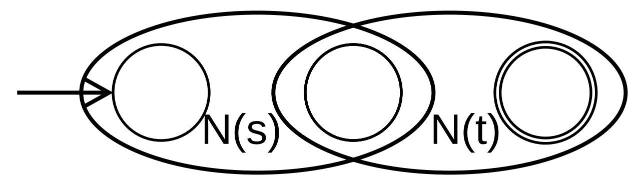


严格按照算法，不要优化

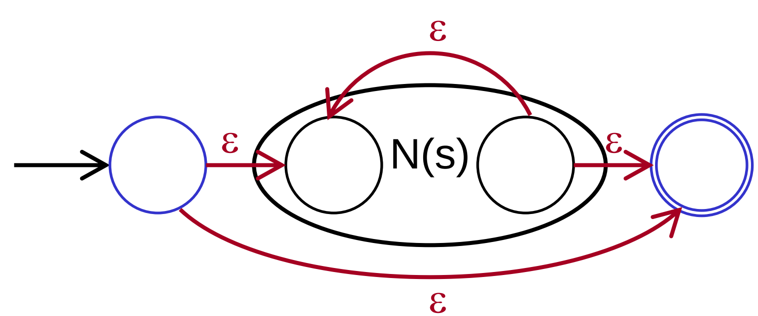
- Induction:
  - r = s | t
    2 new states
    4 new ε transitions



  - r = s t
    No new states
    or transitions



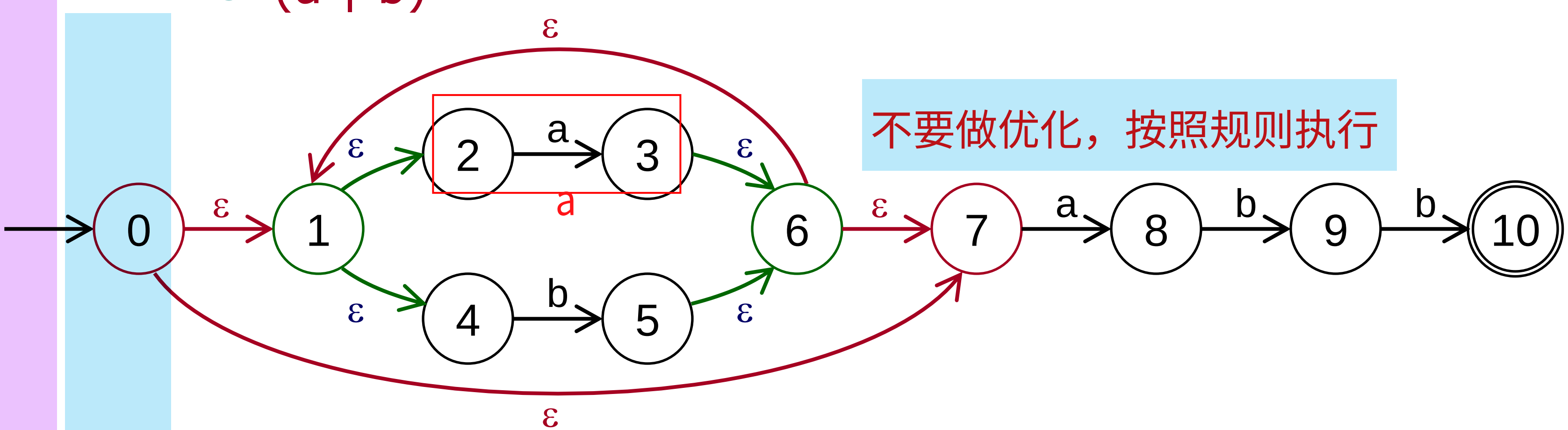
  - r = s*
    2 new states
    4 new ε transitions



An Example

- Given regular expression
  - (a | b)*abb
- We have
  - a | b
  - (a | b)*

带ε-边的NFA



词法必考

ε-NFA to DFA

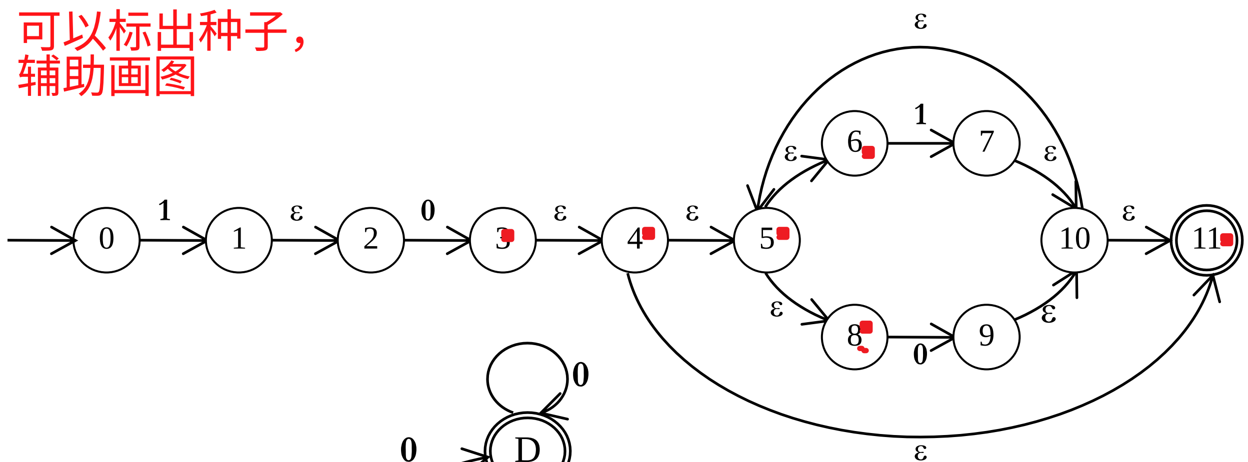
- Both 2 approaches are applied
  - Subset construction
  - ε-closure construction
- Subset
  - A state in the new DFA corresponds to a subset of states in the original NFA.
- ε-closure
  - The empty transition does not consume any input.

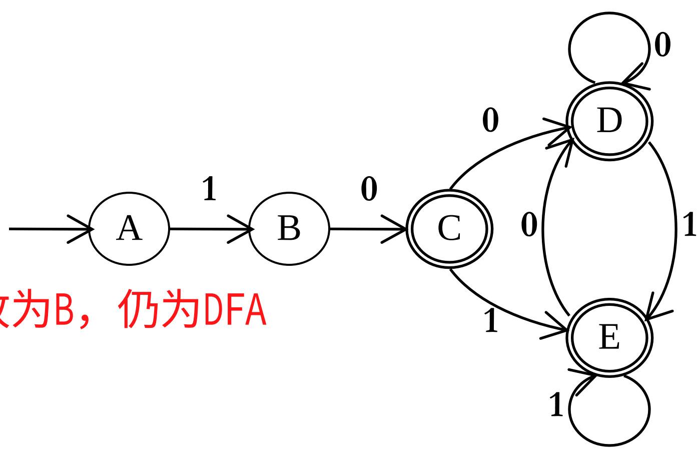
An Example

A = {0}, B = {1, 2}, C = {3, 4, 5, 6, 8, **11**}

D = {9, 10, **11**, 5, 6, 8}, E = {7, 10, **11**, 5, 6, 8}

可以标出种子，
辅助画图

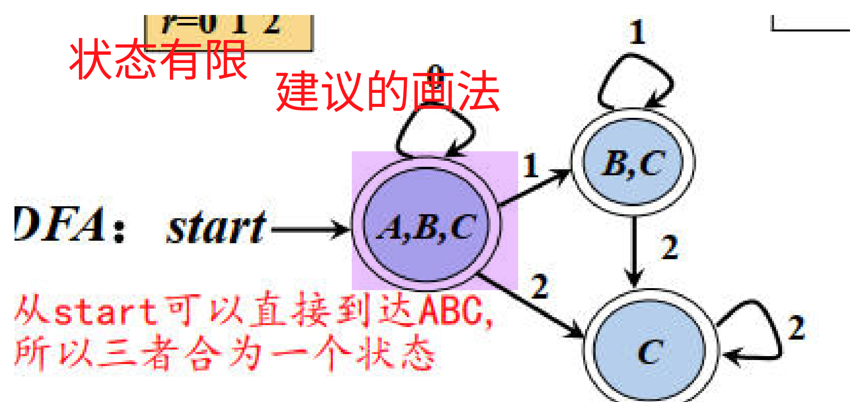




把开始符号改为B，仍为DFA

状态有限

建议的画法



从start可以直接到达ABC，
所以三者合为一个状态

Calculate the ε-closure

- Let ε-closure({0}) = {0} = A, then
  - δ(A, 1) = ε-closure({1}) = {1, 2} = B
  - δ(B, 0) = ε-closure({3}) = {3, 4, 5, 6, 8, 11} = C
  - δ(C, 0) = ε-closure({9}) = {5, 6, 8, 9, 10, 11} = D
  - δ(C, 1) = ε-closure({7}) = {5, 6, 7, 8, 10, 11} = E
  - δ(D, 0) = ε-closure({9}) = D
  - δ(D, 1) = ε-closure({7}) = E
  - δ(E, 0) = ε-closure({9}) = D
  - δ(E, 1) = ε-closure({7}) = E

对于DFA的确定化、优化

Reduction of DFA

极小化：状态变少，合并等价状态

- Find out equivalent classes of states
  - Approach 1:
    - Use a table to represent the equivalent relationship
    - Only a triangle is needed.
  - Approach 2:
    - Use partitions of the set of states

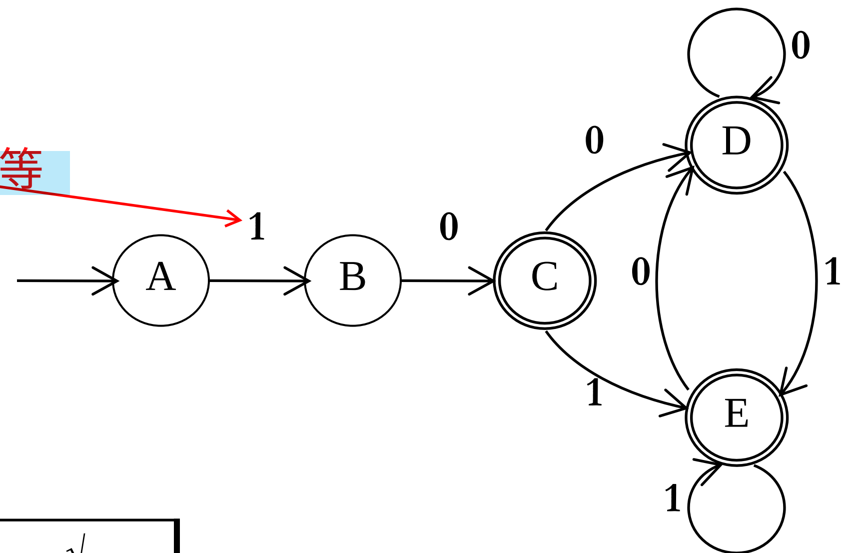
方法1

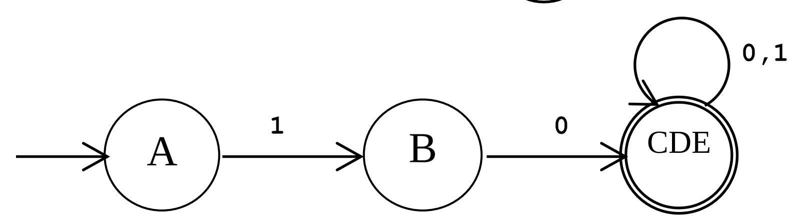
An Example

核心找出等价关系

B	$\times_2$ $\delta(A, 1) = B$ $\delta(B, 1) = \perp$ 不存在，因此不相等			
C	$\times_1$ Final	$\times_1$ Final		
D	$\times_1$ Final	$\times_1$ Final	$\surd_3$ $\delta(C, 0) = D$ $= \delta(D, 0)$ $\delta(C, 1) = E$ $= \delta(D, 1)$	
E	$\times_1$ Final	$\times_1$ Final	$\surd_4$ $\delta(C, 0) = D$ $= \delta(E, 0)$ $\delta(C, 1) = E$ $= \delta(E, 1)$	$\surd_5$ $\delta(D, 0) = D$ $= \delta(E, 0)$ $\delta(D, 1) = E$ $= \delta(E, 1)$
	A	B	C	D

CDE可识别
ifuxilou
AB不可





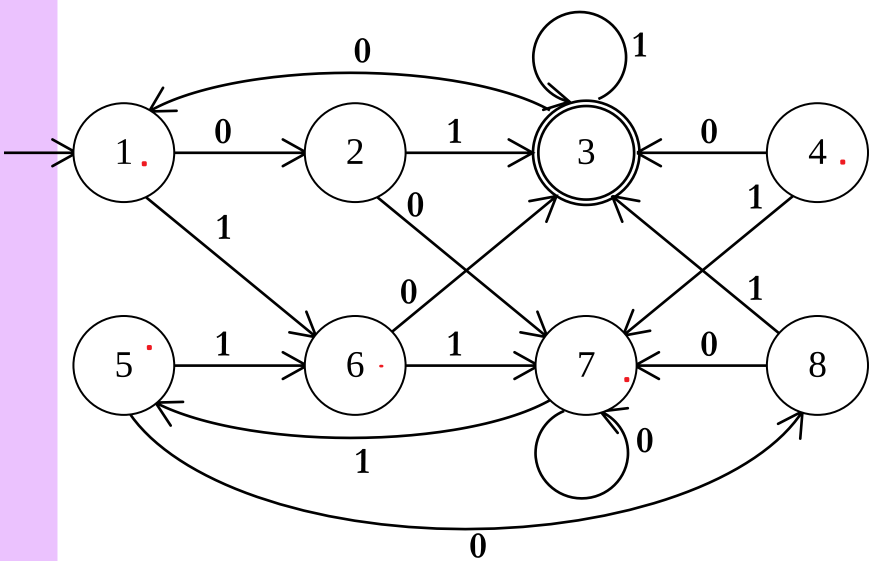
根据自反性删除重复的内容，一半的表就够了
只有终结状态可以识别yifuxilou

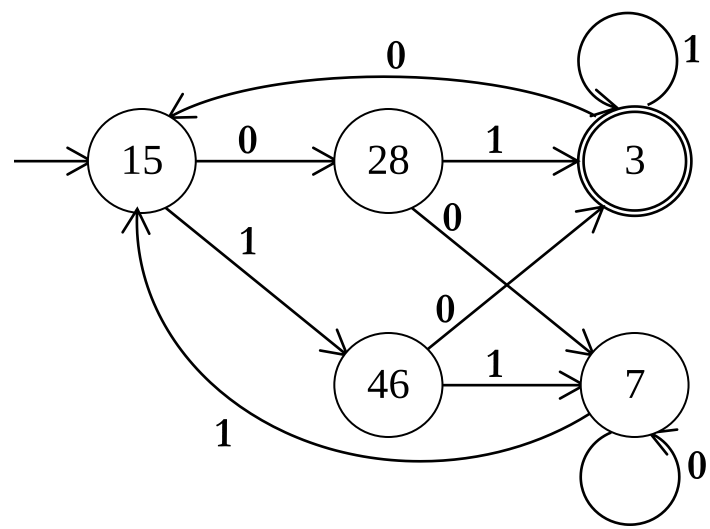
方法2

More Example

考试的难度

- {1, 2, 4, 5, 6, 7, 8} {3} (final and non-final, that is from ε) 互不等价
- {1, 4, 5, 6, 7} {2, 8} {3} (from 1 to final and non-final) 通过1进入终结状态
- {1, 5, 7} {4, 6} {2, 8} {3} (from 0 to final and non-final)
- {1, 5} {7} {4, 6} {2, 8} {3} (from 0 to state {2, 8} and {7})
  {1, 5} {7} {4, 6} {2, 8} {3} (or from 1 to state {6} and {5})
- No difference found, terminate.





1 5是否等价取决于2 8是否等价
执行完必须复核一次

6. Limits of Regular Languages

scanning可以识别词法，不能识别语法

- Regular languages are not able to handle infinite counting and matching
  - e.g. $\{a^n\, b\, c^n \mid n \geq 1\}$ is NOT a regular set.

    begin end a、c个数要求一样

不同文法要求不同但是大小未必更大或者更小

- The matching of parentheses in most programming languages
  - Can not be described by regular expressions
  - Can not be generated by regular grammars
  - Can not be recognized by finite automata

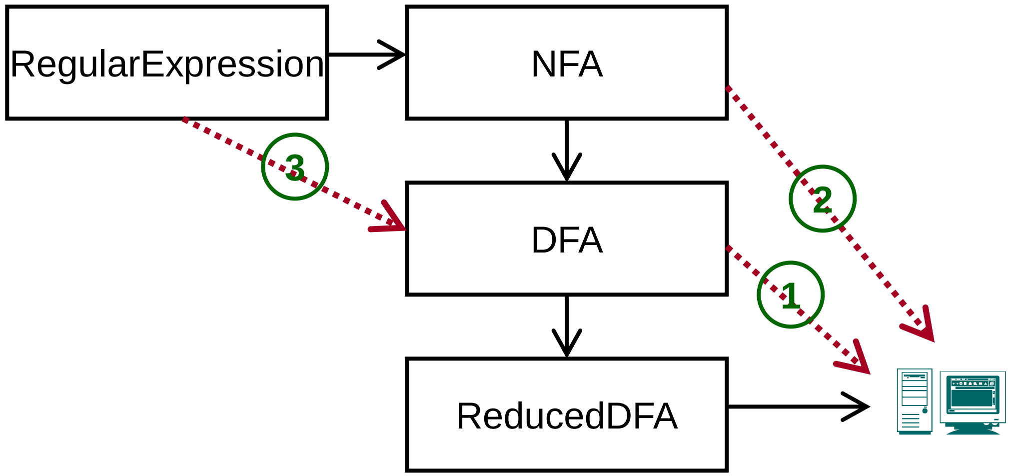
7. Lexical Analysis in Practice

- Scanner in practice: trained in our lab-time
  - Construct a scanner manually
    - Define the lexical rules
    - Construct a finite automaton
    - Write a scanner based on the blueprint of the finite automaton
  - Construct a scanner with a scanner generator
    - lex, Flex, and JFlex

      C Java

String Processing

- Regular expression is a common way to describe a pattern in a string.
  - Trade-off and consequence in the implementation of regular expressions
    - Recognizing with an NFA
    - Regular expression to DFA directly



Exercise 3.1

- Give the recognized tokens of the following program in Pascal.

```
function max(i, j: integer): integer;
{return the maximum of integers i and j}
begin
  if i > j then max := i else max := j
end;
```

Exercise 3.2

- (DBv2, Ch.3, pp.125, ex.3.3.2) Describe the languages denoted by the following regular expressions:
  - $a\ (a \mid b)^*\ a$
  - $a^*\ b\ a^*\ b\ a^*\ b\ a^*$

自然语言描述

Exercise 3.3

- (DBv2, Ch.3, pp.125, ex.3.3.4) Most Languages are case sensitive, so keywords can be written only one way, and the regular expressions describing their lexemes are very simple.
- However, some languages, like Pascal and SQL, are case insensitive. For example, the SQL keyword **`SELECT`** can also be written **`select`**, **`Select`**, or **`sELEcT`**.
- Show how to write a regular expression for a keyword in a case insensitive language. Illustrate your idea by writing the expression for **`SELECT`** in SQL.

Exercise 3.4

- Given the following regular expression

$$1^*(0 \mid 01)^*$$

(1) Transform it to an equivalent finite automaton.

(2) Construct an equivalent DFA for the result of exercise (1).

(3) Reduce the result of (2) and get a reduced DFA.

有难度

Exercise 3.5**

c注释　　字母只是抽象

content　　*

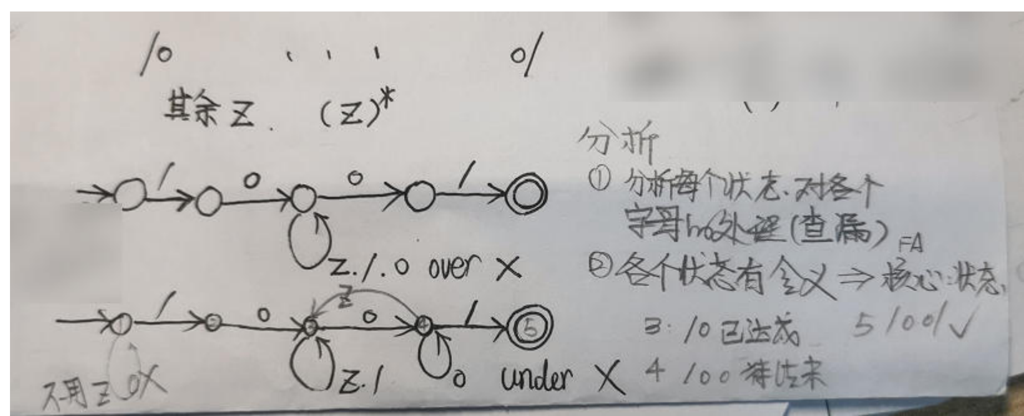
- Given the alphabet Σ = { z, o, / }, a comment in a program over Σ begins with "/o" and ends with "o/". Embedded comments are not permitted.

  (1) Draw a DFA that recognizes nothing but all the comments in the source programs.

  (2) Write a single regular expression that exactly describes all the comments in the source programs.

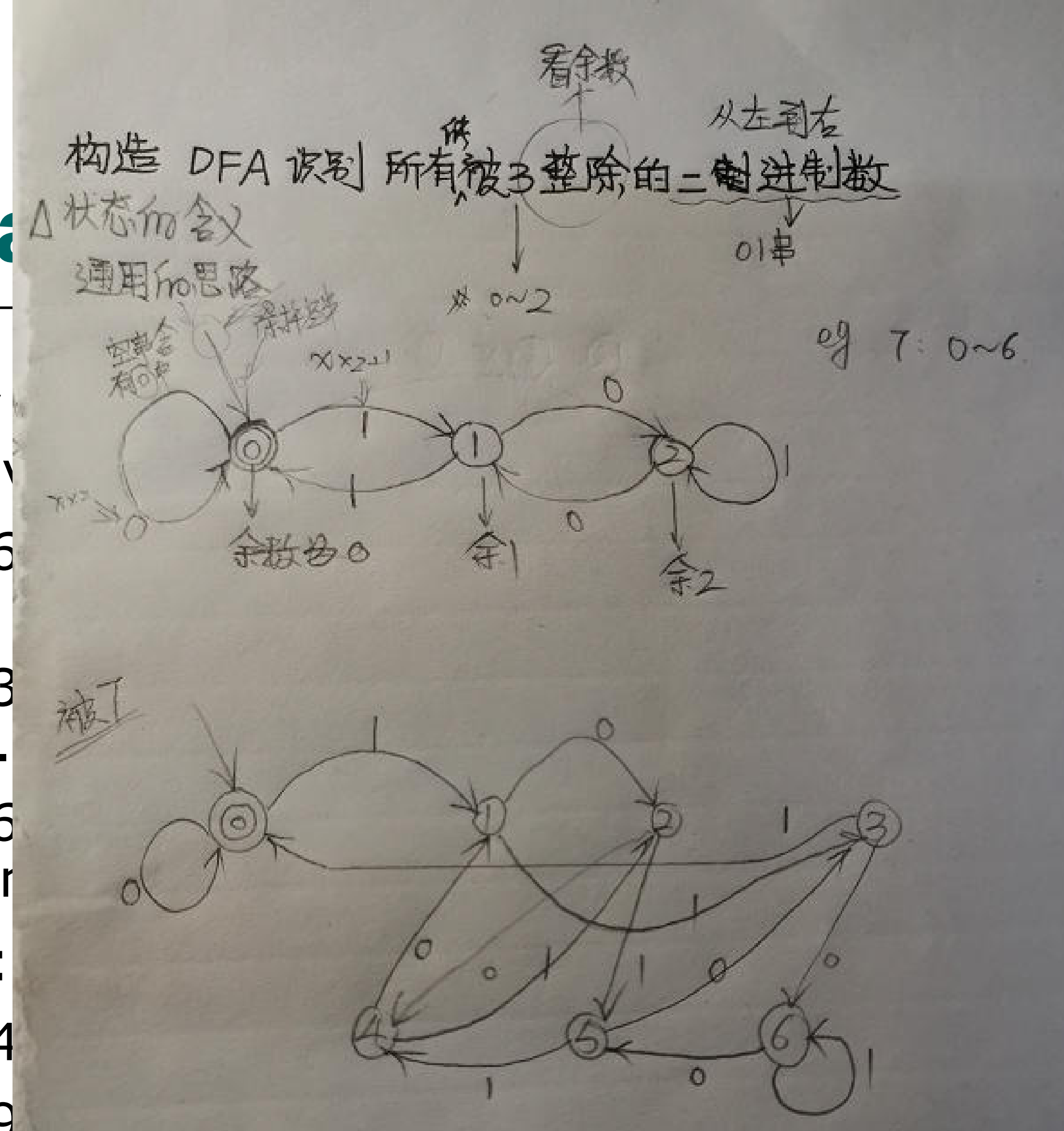
直接用表达式

over specification
under ... :



Further Rea

- Dragon Book,
  - Comprehensi
    - Section 2.6
      scanner.
    - Section 3.3
      definitions.
    - Section 3.6
      related tran
  - Skip reading:
    - Section 3.4
    - Section 3.9
      directly to DFAs.



Enjoy the Course!