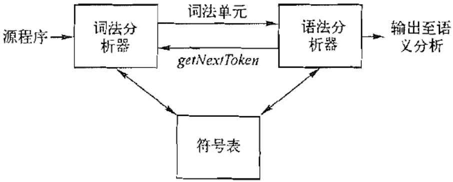
词法分析

魏恒峰

hfwei@nju.edu.cn

2020 年 11 月 4 日

输入: 程序文本/字符串 s & **词法单元 (token) 的规约**



输出: 词法单元流

$$\boxed{\text{token} : \langle \text{token-class}, \text{attribute-value} \rangle}$$

词法单元	非正式描述	词素示例
if	字符 i, f	if
else	字符 e, l, s, e	else
comparison	< 或 > 或 <= 或 >= 或 == 或 !=	<=, !=
id	字母开头的字母 / 数字串	pi, score, D2
number	任何数字常量	3.14159, 0, 6.02e23
literal	在两个 "之间，除" 以外的任何字符	"core dumped"

$$\text{token} : \langle \text{token-class}, \text{attribute-value} \rangle$$

词法单元	非正式描述	词素示例
if	字符 i, f	`if`
else	字符 e, l, s, e	`else`
comparison	< 或 > 或 <= 或 >= 或 == 或 !=	`<=`, `!=`
id	字母开头的字母/数字串	`pi`, `score`, `D2`
number	任何数字常量	`3.14159`, `0`, `6.02e23`
literal	在两个 "之间，除" 以外的任何字符	`"core dumped"`

int/if 关键词

ws 空格、制表符、换行符

comment “//” 开头的一行注释或者 “/* */” 包围的多行注释

```c
int main(void)
{
    printf("hello, world\n");
}
```

```c
int main(void)
{
    printf("hello, world\n");
}
```

int ws **main/id** **LP** **void** **RP** ws

LB ws

ws **id** **LP** **literal** **RP** **SC**ws

RB

```c
int main(void)
{
    printf("hello, world\n");
}
```

int ws **main/id** **LP** **void** **RP** ws

LB ws

ws **id** **LP** **literal** **RP** **SC**ws

RB

本质上, 这就是一个**字符串 (匹配/识别) 算法**

词法分析器的三种设计方法

手写词法分析器

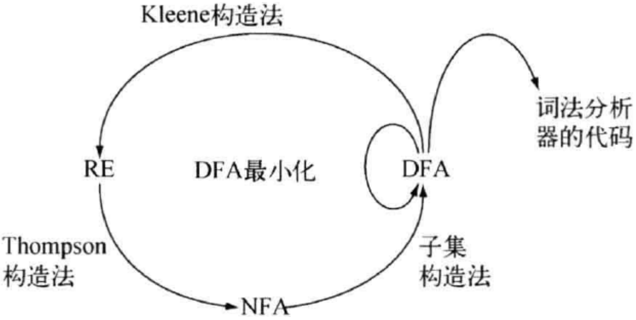
词法分析器的生成器

自动化词法分析器

生产环境下的编译器 (如 gcc) 通常选择**手写词法分析器**


creating lexical analyser
Prezi

目标：正则表达式 RE $\Longrightarrow$ 词法分析器



$$RE \Longrightarrow \epsilon\text{-}NFA \Longrightarrow NFA$$

终点固然令人向往，这一路上的风景更是美不胜收

Definition (NFA (Nondeteministic Finite Automaton))

非确定性有穷自动机 $\mathcal{A}$ 是一个五元组 $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$:

(1) 字母表 Σ ($\epsilon \notin \Sigma$)

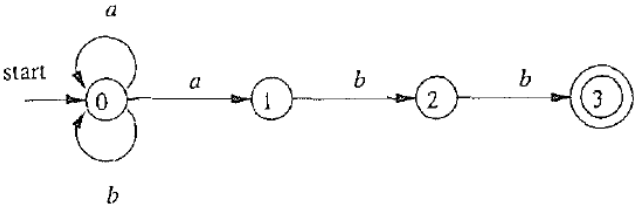
(2) **有穷**的状态集合 S

(3) 唯一的初始状态 $s_0 \in S$

(4) 状态转移**函数** δ

$$\delta : S \times (\Sigma \cup \{\epsilon\}) \to 2^S$$

(5) 接受状态集合 $F \subseteq S$



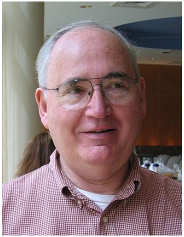
Michael O. Rabin (1931 ∼)

Finite Automata and Their Decision Problems‡

Abstract: Finite automata are considered in this paper as instruments for classifying finite tapes. Each one-tape automaton defines a set of tapes, a two-tape automaton defines a set of pairs of tapes, et cetera. The structure of the defined sets is studied. Various generalizations of the notion of an automaton are introduced and their relation to the classical automata is determined. Some decision problems concerning automata are shown to be solvable by effective algorithms; others turn out to be unsolvable by algorithms.

发表于 1959 年;

1976 年, 共享图灵奖



Dana Scott (1932 ∼)

"which introduced the idea of ***nondeterministic machines****,*

which has proved to be an enormously valuable concept."

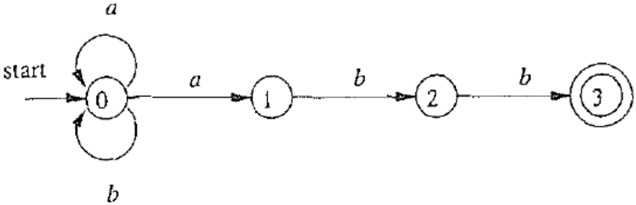
(非确定性) 有穷自动机是一类极其简单的**计算**装置

它可以**识别** (接受/拒绝) Σ 上的字符串

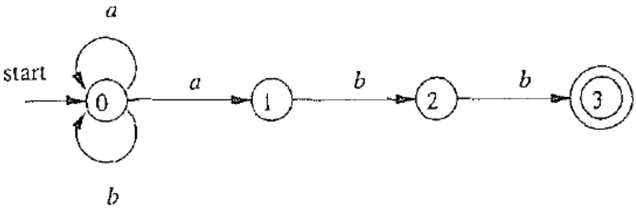
Definition (接受 (Accept))

(非确定性) 有穷自动机 $\mathcal{A}$ 接受字符串 x, 当且仅当**存在**一条从开始状态 s_0 到**某个**接受状态 $f \in F$、标号为 x 的路径。

因此, $\mathcal{A}$ 定义了一种**语言** $L(\mathcal{A})$: 它能接受的所有字符串构成的集合



$$aabb \in L(\mathcal{A}) \qquad ababab \notin L(\mathcal{A})$$



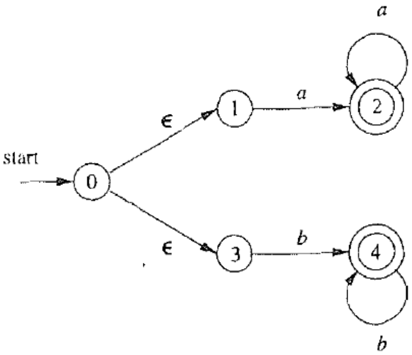
$$aabb \in L(\mathcal{A}) \qquad ababab \notin L(\mathcal{A})$$

$$L(\mathcal{A}) = L((a|b)^* abb)$$

关于语言与自动机 $\mathcal{A}$ 的两个最基本的问题:

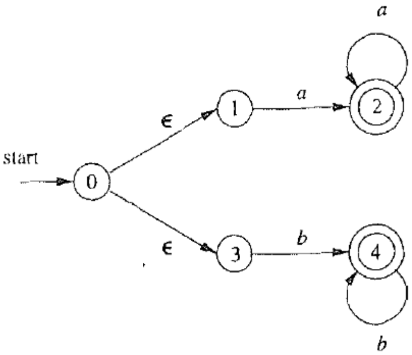
给定字符串 $x, x \in L(\mathcal{A})$?

$L(\mathcal{A})$ 究竟是什么?



$$aaa \in \mathcal{A}?$$

$$L(\mathcal{A}) =$$



$$aaa \in \mathcal{A}?$$

$$L(\mathcal{A}) = L((aa^*|bb^*))$$

Definition (DFA (Deterministic Finite Automaton))

确定性有穷自动机 $\mathcal{A}$ 是一个五元组 $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$:

(1) 字母表 Σ ($\epsilon \notin \Sigma$)

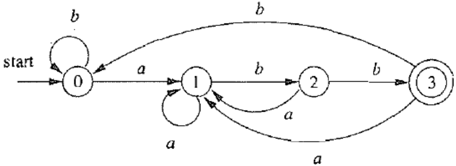
(2) **有穷**的状态集合 S

(3) 唯一的初始状态 $s_0 \in S$

(4) 状态转移**函数** δ

$$\delta : S \times \Sigma \to S$$

(5) 接受状态集合 $F \subseteq S$



Definition (DFA (Deterministic Finite Automaton))

确定性有穷自动机 $\mathcal{A}$ 是一个五元组 $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$:

(1) 字母表 Σ $(\epsilon \notin \Sigma)$

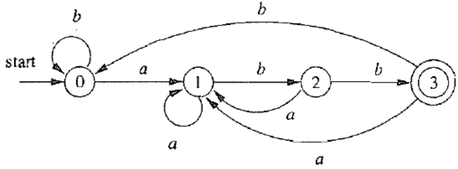
(2) **有穷**的状态集合 S

(3) 唯一的初始状态 $s_0 \in S$

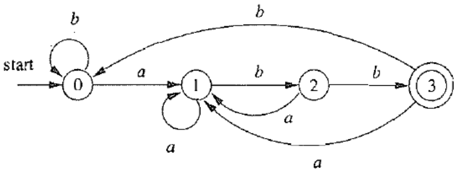
(4) 状态转移**函数** δ

$$\delta : S \times \Sigma \to S$$

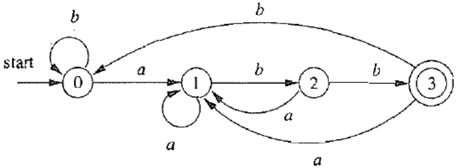
(5) 接受状态集合 $F \subseteq S$



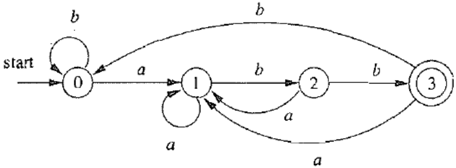
约定: 所有没有对应出边的字符默认指向一个不存在的 **"死状态"**



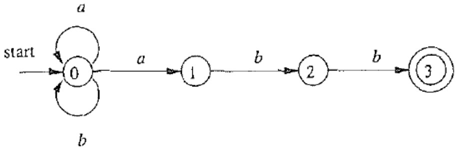
$$L(\mathcal{A}) =$$



$$L(\mathcal{A}) = L((a|b)^* abb)$$



$$L(\mathcal{A}) = L((a|b)^* abb)$$



NFA 与 DFA 的**优缺点**比较:

$x \in L(\mathcal{A})$: DFA 易于实现; NFA 不易实现

$L(\mathcal{A})$: NFA 简洁易于理解; DFA 状态多转移多

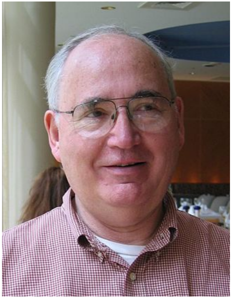
取长补短:

用 NFA 描述, 用 NFA 实现;

从 NFA 到 DFA 的转化自动完成

从 NFA 到 DFA 的转换: **子集构造法** (Subset/Powerset Construction)

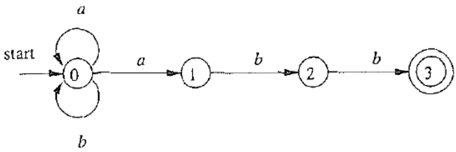
Michael O. Rabin (1931 ∼)



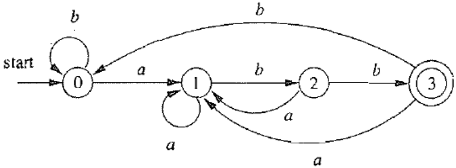
Dana Scott (1932 ∼)

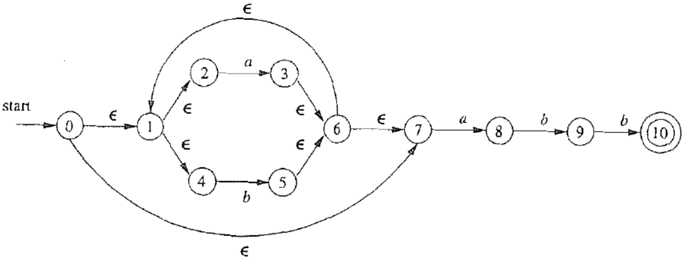
思想: 用 DFA 模拟 NFA

用 DFA 模拟 NFA

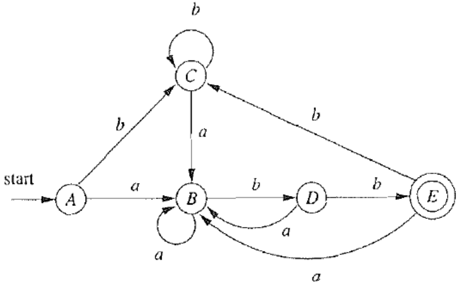


$$L(\mathcal{A}) = L((a|b)^* abb)$$





$$L(\mathcal{A}) = L((a|b)^* abb)$$



ϵ-closure(s)

$$\epsilon\text{-closure}(s)$$

$$\epsilon\text{-closure}(T) = \bigcup_{s \in T} \epsilon\text{-closure}(s)$$

$$\epsilon\text{-closure}(s)$$

$$\epsilon\text{-closure}(T) = \bigcup_{s \in T} \epsilon\text{-closure}(s)$$

$$\text{move}(T, a) = \bigcup_{s \in T} \delta(s, a)$$

子集构造法 ($\mathcal{N} \Longrightarrow \mathcal{D}$) 实现时采用**标记搜索**过程

```
一开始, ε-closure(s_0) 是 Dstates 中的唯一状态, 且它未加标记;
while ( 在 Dstates 中有一个未标记状态 T ) {
        给 T 加上标记;
        for ( 每个输入符号 a ) {
                U = ε-closure(move(T, a));
                if ( U 不在 Dstates 中 )
                        将 U 加入到 Dstates 中, 且不加标记;
                Dtran[T, a] = U;
        }
}
```

接受状态集 $F_{\mathcal{D}} = \{s \in S_{\mathcal{D}} \mid \exists f \in F_{\mathcal{N}}.\ f \in s\}$

子集构造法的**复杂度分析**:

$$(|S_{\mathcal{N}}| = n)$$

$$\Theta(2^n)$$

子集构造法的**复杂度分析**:

$(|S_{\mathcal{N}}| = n)$

$$\Theta(2^n)$$

最坏情况下, 复杂度为 $\Omega(2^n)$

子集构造法的**复杂度分析**:

$(|S_{\mathcal{N}}| = n)$

$$\Theta(2^n)$$

最坏情况下, 复杂度为 $\Omega(2^n)$

"长度为 $m \geq n$ 个字符的 0,1 串, 且倒数第 n 个字符是 1"

子集构造法的**复杂度分析**:
$(|S_{\mathcal{N}}| = n)$

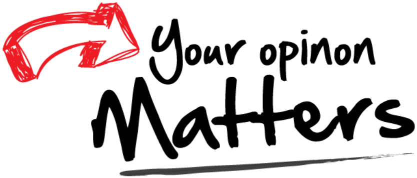
$$\Theta(2^n)$$

最坏情况下, 复杂度为 $\Omega(2^n)$

"长度为 $m \geq n$ 个字符的 0,1 串, 且倒数第 n 个字符是 1"

作业: $m = n = 3$


Thank
You!



Office 926

hfwei@nju.edu.cn