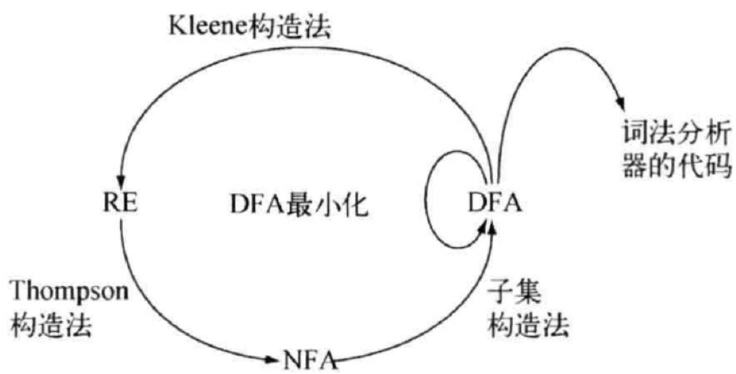
Chapter-3-REandAutomata

> main topic: 正则表达式与自动机理论

Course Notes

Outline

目标: 正则表达式 RE =⇒ 词法分析器



Basic

Definition (字母表)

- 字母表 Σ 是一个 **有限的符号集合**

Definition (串)

- 字母表 Σ 上的串 (s) 是由 Σ 中 **符号构成** 的一个 **有穷序列**

Definition (串上的 "连接" 运算)

- x = dog, y = house , xy = doghouse
- $s\epsilon = \epsilon s = s$

Definition (串上的 "指数" 运算)

- $s^0 \triangleq \epsilon$
- $s^i \triangleq ss^{i-1}$

Definition (语言)

- 语言是给定字母表 Σ 上一个任意的 *可数* 的 **串集合**
  - ∅
  - {ϵ}

- id : {a, b, c, a1, a2, . . . }
- ws : {blank, tab, newline}
- if : {if}

- **语言是串的集合**
- 因此, 我们可以通过 *集合操作构造新的语言*

语言的运算

1. L和M的并：L U M = {s | s属于L or s属于M}
2. L和M的连接：LM = {st | s属于L 且 t属于M}
3. L的Kleene闭包：L* = $U_{i=0}^{\inf} L^i$
4. L的正闭包：$L^+ = U_{i=1}^{\inf} L^i$

Regular Expression

- 每个正则表达式 r 对应一个正则语言 L(r)
- **正则表达式是语法, 正则语言是语义**
- eg:
    - ID: [a-zA-Z][a-zA-Z0-9]*
    - the language: {a1, a2, ab, . . .}

Definition (正则表达式)
给定字母表 Σ, Σ 上的正则表达式由且仅由以下规则定义:

- (1) ϵ 是正则表达式;
- (2) ∀a ∈ Σ, a 是正则表达式;
- (3) 如果 r 是正则表达式, 则 (r) 是正则表达式; (即：是否加括号不影响语言)
- (4) 如果 r 与 s 是正则表达式, 则 r|s, rs, r $*$ 也是正则表达式。

ps

- 运算优先级: () > $*$ > "连接" > |
    - eg: (a)|((b) $*$ (c)) ≡ a|b $*$ c

每个正则表达式 r 对应一个正则语言 L(r)

Definition (正则表达式对应的正则语言)

1. L(ϵ) = {ϵ}
2. L(a) = {a}, ∀a ∈ Σ
3. L((r)) = L(r)
4. 或 / 连接 / Kleene闭包
    - L(r|s) = L(r) ∪ L(s)
    - L(rs) = L(r)L(s)
    - L(r^*) = $(L(r))^*$

PS:

1. 常见的正则表达式文法

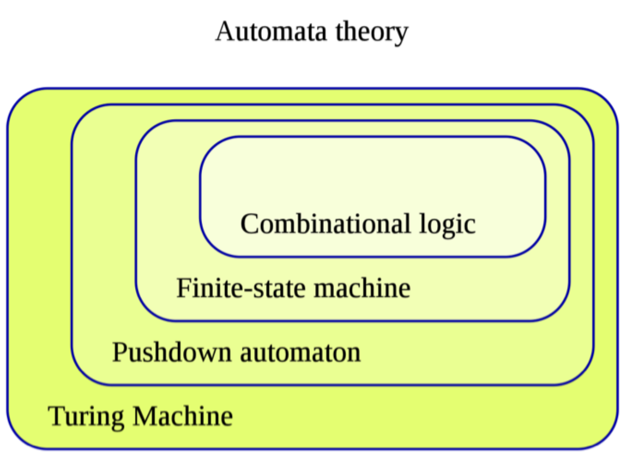
表达式	匹配	例子
c	单个非运算符字符 c	a
\c	字符 c 的字面值	*
"s"	串 s 的字面值	"**"
.	除换行符以外的任何字符	a.*b
^	一行的开始	^abc
$	行的结尾	abc$
[s]	字符串 s 中的任何一个字符	[abc]
[^s]	不在串 s 中的任何一个字符	[^abc]
$r*$	和 r 匹配的零个或多个串连接成的串	a*
$r+$	和 r 匹配的一个或多个串连接成的串	a+
$r?$	零个或一个 r	a?
$r\{m,n\}$	最少 m 个，最多 n 个 r 的重复出现	a{1,5}
r_1r_2	r_1 后加上 r_2	ab
$r_1 \mid r_2$	r_1 或 r_2	a|b
(r)	与 r 相同	(a|b)
r_1/r_2	后面跟有 r_2 时的 r_1	abc/123

2. 正则表达式转换网站：regex101

Automata

根据表达/计算能力的强弱, 自动机可以分为不同层次:

> Turing Machine > Pushdown Automaton > Finite-state machine > Combinational logic



NFA

Definition (NFA (Nondeteministic Finite Automaton))
非确定性有穷自动机 , A 是一个五元组 A = (Σ, S, s 0 , δ, F):

- (1) 字母表 Σ (ϵ ∈ / Σ)
- (2) 有穷的状态集合 S
- (3) 唯一的初始状态 s0
- (4) 状态转移函数 δ∈S
    - δ : S × (Σ ∪ {ϵ}) → 2^S
- (5) 接受状态集合 F ⊆ S

1. 约定: 所有没有对应出边的字符默认指向 “空状态” ∅
2. (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 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 ∈ F、标号为 x 的路径

因此, A 定义了一种语言 L(A): 它能接受的所有字符串构成的集合

关于自动机 A 的两个基本问题:

- Membership 问题: 给定字符串 x, x ∈ L(A)?
- L(A) 究竟是什么?

DFA

Definition (DFA (Deterministic Finite Automaton)) 确定性有穷自动机
A 是一个五元组 A = (Σ, S, s_0 , δ, F):

- (1) 字母表 Σ (ϵ ∈ / Σ)
- (2) 有穷的状态集合 S
- (3) 唯一的初始状态 s_0 ∈ S
- (4) 状态转移函数 δ
    - δ : S × Σ → S
- (5) 接受状态集合 F ⊆ S

1. 约定: 所有没有对应出边的字符默认指向一个 “死状态”

Design

1. NFA 简洁易于理解, 便于描述语言 L(A)

2. DFA 易于判断 x ∈ L(A), 适合产生词法分析器

3. 我们的策略：用 NFA 描述语言, 用 DFA 实现词法分析器 RE

    - ⇒ NFA ⇒ DFA ⇒ 词法分析器

从 RE 到 NFA: Thompson 构造法

- RE ⇒ NFA

- r ⇒ N(r)
- 要求 : L(N(r)) = L(r)

Thompson 构造法的基本思想: **按结构归纳**

1. ϵ 是正则表达式
2. a ∈ Σ 是正则表达式
3. 如果 s 是正则表达式, 则 (s) 是正则表达式
4. 如果 s, t 是正则表达式, 则 s|t 是正则表达式
5. 如果 s, t 是正则表达式, 则 st 是正则表达式
6. 如果 s 是正则表达式, 则 s^* 是正则表达式

N(r) 的 性质 以及 Thompson 构造法 复杂度分析

1. N(r) 的开始状态与接受状态均唯一
2. 开始状态没有入边, 接受状态没有出边
3. N(r) 的状态数 |S| ≤ 2 × |r| (|r| : r 中运算符与运算分量的总和)
4. 每个状态最多有两个 ϵ-入边与两个 ϵ-出边
5. ∀a ∈ Σ, 每个状态最多有一个 a-入边 与一个 a-出边

从 NFA 到 DFA: 子集构造法

子集构造法：Subset / Powerset Construction

- NFA ⇒ DFA
- N ⇒ D
- 要求 : L(D) = L(N)

子集构造法的思想: 用 DFA 模拟 NFA

构造方法：

1. 从状态 s 开始, **只** 通过 ϵ-转移 可达的状态集合
    - ϵ-closure(s) = {t ∈ S_N |s − ϵ^*→ t}
2. ϵ-closure(T) = $\cup_{s \in T}$ ϵ-closure(s)
3. move(T, a) = $\cup_{s \in T}$ δ(s, a)

子集构造法 (N =⇒ D) 的原理:

- N : (Σ_N , S_N , n_0 , δ_N , F_N)
- D : (Σ_D , S_D , d_0 , δ_D , F_D)
- ΣD =ΣN
- S D ⊆ 2^{S_N} (∀s D ∈ S D : s D ⊆ S N)

1. 初始状态 d 0 = ϵ-closure(n 0)
2. 转移函数 ∀a ∈ Σ D : δ D (s D , a) = ϵ-closure(move(s D , a))
3. 接受状态集 F D = {s D ∈ S D | ∃ f ∈ F N . f ∈ s D }

子集构造法的复杂度分析: (|S N | = n)

- $|S_D|$ = Θ (2^n) = O(2^n) ∩ Ω(2^n)

闭包:

1. 闭包 (Closure): f-closure(T)
2. ϵ-closure(T)
3. T ⇒ f(T) ⇒ f(f(T)) ⇒ f(f(f(T))) ⇒ . . . 直到找到 x 使得 f(x) = x (x 称为 f 的 *不动点*)

DFA 最小化算法

基本思想: 等价的状态可以合并

如何定义等价状态?

- “等价”的符号表示: s ∼ t
- 定义“等价”: s ∼ t ⇐⇒ ∀a ∈ Σ. (s --a→ s ′) ∧ (t --a→ t ′) ⇒ (s ′ ∼ t ′)

基于该定义, 不断合并等价的状态, 直到无法合并为止!

- s ∼ t ⇐⇒ ∀a ∈ Σ. (s --a→ s ′) ∧ (t --a→ t ′) =⇒ (s ′ ∼ t ′).
  - 缺少基础情况, 不知从何下手
- s ≁ t ⇐⇒ ∃a ∈ Σ. (s --a→ s ′) ∧ (t --a→ t ′) ∧ (s ′ ≁ t ′)
  - 划分, 而非合并!

接受状态 与 *非接受状态* 必定不等价 ⇒ 空串 ϵ 区分了这两类状态

DFA 最小化等价状态划分方法:

1. Π = {F, S \ F}
2. 直到再也无法划分为止 (不动点!)
3. 然后, 将同一等价类里的状态合并
4. 要注意处理 “死状态”

PS:

- 不适用于 NFA 最小化;
- NFA 最小化问题是 PSPACE-complete 的

Definition (可区分的 (Distinguishable); 等价的 (Equivalent))

- 如果存在某个能区分状态 s 与 t 的字符串, 则称 s 与 t 是可区分的;
- 否 则, 称 s 与 t 是等价的

Definition (字符串 x 区分状态 s 与 t)

- 如果分别从 s 与 t 出发, 沿着标号为 x 的路径到达的两个状态中只有一个是接受状态, 则称 x 区分了状态 s 与 t

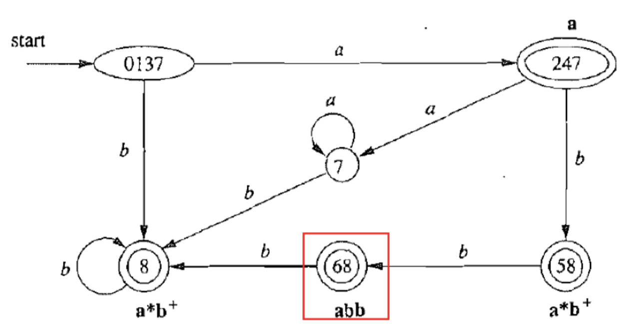
从DFA到词法分析器

原则：

- 最前优先匹配: abb (比如 关键字)
- 最长优先匹配: aabbb

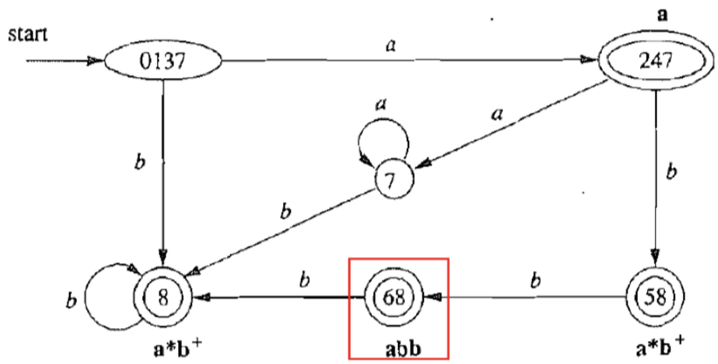
方式：

1. 根据正则表达式构造相应的 NFA
2. 如有需要先合并NFA（要保留各个 NFA 的接受状态信息, 并采用最前优先匹配原则）
3. 使用子集构造法将 NFA 转化为等价的 DFA (需要消除 “死状态”, 避免词法分析器徒劳消耗输入流)
4. 模拟运行该 DFA, 直到无法继续为止 (输入结束或状态无转移)
5. 假设此时状态为 s
    1. 若 s 为接受状态, 则识别成功
    2. 否则, 回溯 (包括状态与输入流) 至 *最近一次* 经过的 **接受状态, 识别成功!**
    3. 若没有经过任何接受状态, 则报错 (忽略第一个字符)
    4. 无论成功还是失败, 都从 **初始状态**开始继续识别下一个词法单元



$$\begin{aligned} x = a &\qquad \text{输入结束; 接受; 识别出 } a \\ x = abba &\qquad \text{状态无转移; 回溯成功; 识别出 } abb \\ x = aaaa &\qquad \text{输入结束; 回溯成功; 识别出 } a \\ x = cabb &\qquad \text{状态无转移; 回溯失败; 报错 } c \end{aligned}$$

特定于词法分析器的 DFA 最小化方法



初始划分需要考虑不同的**词法单元**

$$\Pi_0 = \{\{0137, 7\}, \{247\}, \{8, 58\}, \{68\}, \{\emptyset\}\}$$

$$\Pi_1 = \{\{0137\}, \{7\}, \{247\}, \{8\}, \{58\}, \{68\}\}$$

从DFA到RE：Kleene构造法

- DFA ⇒ RE
- D ⇒ r
- 要求 : L(r) = L(D)

这部分不做要求，故略!

Learn More about Regular Expressions

Web Ref: RE

Here we offer some basic examples:

- https://regex101.com/r/jucEtW/1 (regex/bg-color)
- https://regex101.com/r/jchuZs/1 (regex/date)
- https://regex101.com/r/fWJkCF/1 (regex/dollar)
- https://regex101.com/r/K5MCMZ/1 (regex/cat)
- https://regex101.com/r/PUsCwP/1 (regex/html-head)
- https://regex101.com/r/eXue43/1 (regex/html-head-lookaround)
- https://regex101.com/r/I07Gpu/1 (regex/html-a-img)

Slide

03/04-REandAutonoma

[计算理论导引](https://ocw.mit.edu/courses/ 18-404j-theory-of-computation-fall-2020/)