

<https://github.com/lazyparser/becoming-a-compiler-engineer>

方舟·编译技术入门与实战

第六课：习题课（2）

吴伟 (@lazyparser)

2019-12-30

本课程所有资料都是开源免费的（更新中）

- 课程配套代码及幻灯片地址（有不明白的地方就开issues问）
 - <https://github.com/lazyparser/becoming-a-compiler-engineer>
 - <https://github.com/lazyparser/becoming-a-compiler-engineer-codes>
- 课程视频回看（包含所有直播及录播视频）
 - <https://space.bilibili.com/296494084>
- 课程直播地址（可以弹幕或评论区互动）
 - <https://live.bilibili.com/10339607>

志愿者征集：B站支持字幕了

- 用户可以自己编辑字幕并上传，有在线的字幕编辑器可以使用
- 需要UP主（我）和B站内容管理员进行审核
 - 我一般不出差的话当天会审核通过（或退回修改）
- 请大家有空时候为课程添加一点字幕，方便静音状态下学习观看
 - 这样可以在地铁或其他碎片时间观看视频了 ☺

本次课程将分四次视频介绍

- 已经学习的思维模式的回顾
- 讲解《现代编译原理》第二章词法分析课后习题
- 讲解《编译器设计》第二章词法分析课后习题
- 讲解《现代编译原理》第三章语法分析课后习题
- 讲解《编译器设计》第三章语法分析课后习题

复习：思考算法问题的几个思考模式

- 问题是如何定义的？
- 解空间规整么？
- 解的存在性证明了么？
- 构造性的方法来找到解？
- 穷举法？贪婪法？
- 是要找验证算法么？
- 是否可以构造等价映射？
- 能Google到答案么？

新增：几个使用的测试的思想

- 输入有哪些渠道？
- 信息是否足够重现问题？
- 空输入会发生什么？
- 极端大的输入会崩溃么？
- 输入的分类边界有哪些？
- 是否可以构造个fuzzer？
- 能Google到类似测试么？

<https://github.com/lazyparser/becoming-a-compiler-engineer/issues/12>

作业上传讨论：《编译器设计》第二章词法分析 #12

New issue

! Open

lazyparser opened this issue 2 days ago · 9 comments



lazyparser commented 2 days ago

Owner

...

Assignees

No one assigned

Hi all

现在作业和练习题已经逐渐变难，完全自学开始有点
课堂讨论。目前构思和尝试的几种方式，要面对大家时
布置作业 --> GitHub issues 汇总作业 --> 视频课针对某
些问题讲解

后续，《现代编译原理》和《编译器设计》的课后习题
提交可以是某个单独的习题的答案，最好做了一点就

也欢迎大家相互评论和提问:-)

具体答案请在issues里看，已经上传。
有疑问的题目欢迎新开issues讨论。
本次视频主要讲解解决问题的思路



pz9115 commented 2 days ago · edited

Milestone

No milestone

3 participants



2.2

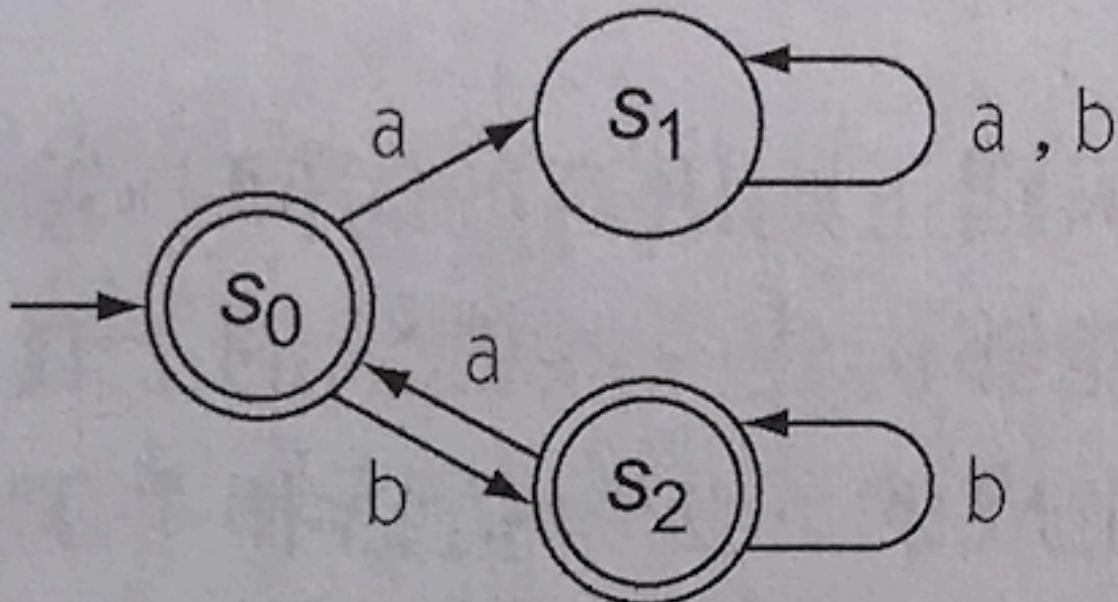
(1) (a) $S=\{s_0, s_1, s_2\}$ $\Sigma=\{a, b\}$ $\delta=\{(s_0, a) \rightarrow s_1, (s_0, b) \rightarrow s_2, (s_1, a) \rightarrow s_1, (s_1, b) \rightarrow s_1, (s_2, a) \rightarrow s_0, (s_2, b) \rightarrow s_2\}$
 $s_0=s_0$ $SA=\{s_0, s_2\}$

(b) $S=\{s_0, s_1, s_2, s_3\}$ $\Sigma=\{0, 1\}$ $\delta=\{(s_0, 0) \rightarrow s_1, (s_0, 1) \rightarrow s_2, (s_1, 0) \rightarrow s_3, (s_1, 1) \rightarrow s_0, (s_2, 0) \rightarrow s_1, (s_2, 1) \rightarrow s_3,$
 $(s_3, 0) \rightarrow s_3, (s_3, 1) \rightarrow s_3\}$ $s_0=s_0$ $SA=s_3$

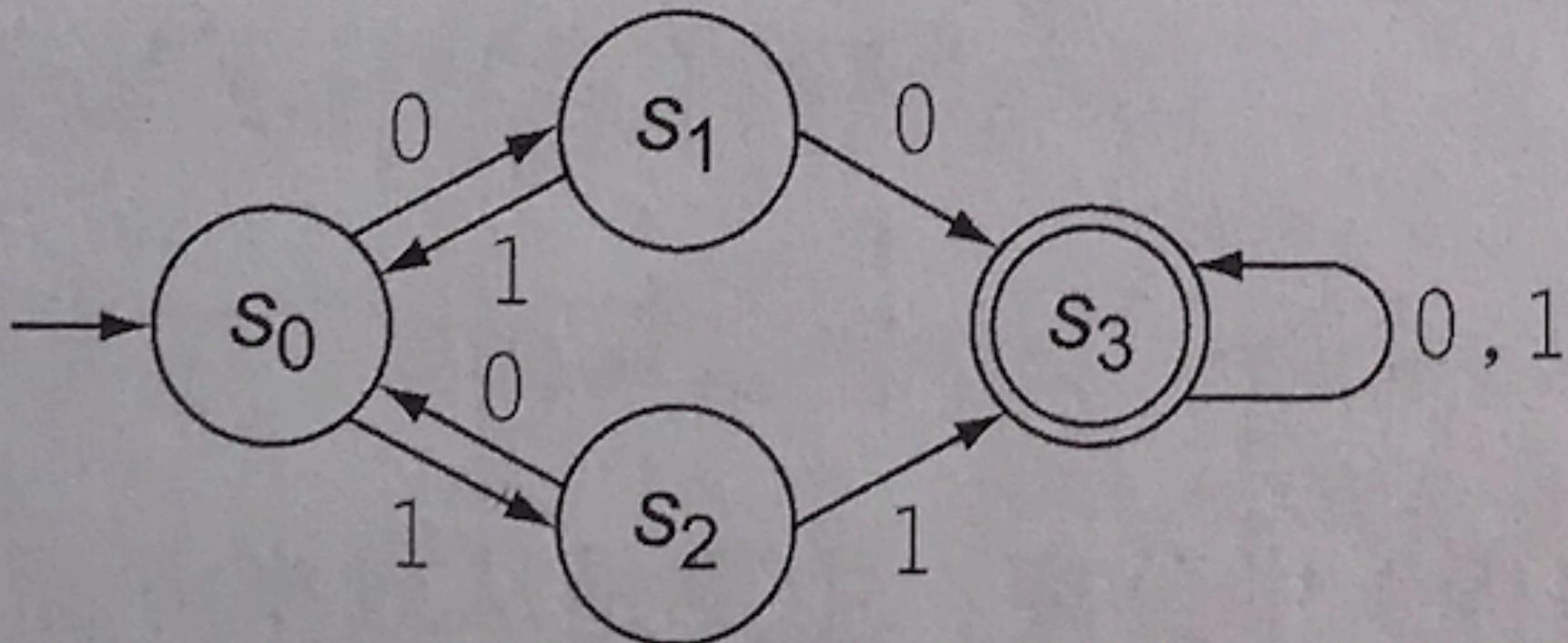
2.2节

(1) 非正式描述下列FA接受的语言：

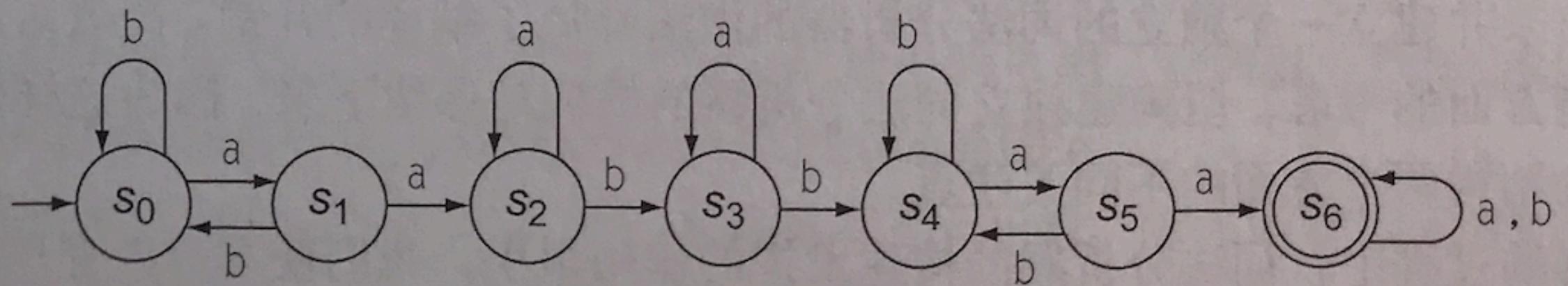
(a)



(b)



(c)



(2) 分别构建一个FA，识别下述的各种语言：

- (a) $\{w \in \{a, b\}^* \mid w \text{以'a'开头, 包含子串'baba'}\}$
- (b) $\{w \in \{0, 1\}^* \mid w \text{包含子串'111', 不包含子串'00'}\}$
- (c) $\{w \in \{a, b, c\}^* \mid w \text{中, 'a'的数目对2取模等于'b'的数目对3取模}\}$

(c) $\{w \in \{a, b, c\}^* \mid w \text{ 以 } a \text{ 的数目对 } 2 \text{ 取模等于 } b \text{ 的数目对 } 3 \text{ 取模}\}$

(3) 分别创建FA，识别(a)表示复数的单词；(b)表示十进制数的单词，其格式为科学记数法。

2.2 节

2.3节

(4) 不同程序设计语言使用不同的符号表示法来表示整数。分别为下列各种整数构造一个对应的正则表达式。

(a) C语言中的非负整数，基数可能为10或16。

(b) VHDL中的非负整数，其中可能包含下划线（不能作为第一个或最后一个字符）。

(c) 货币，按美元计算，表示为正的十进制数，小数点后有效数字两位，其余数字四舍五入。

这种数字以字符\$开头，小数点左侧每三个数字用逗号分隔为一组，小数点右侧保留两位数字，例如\$8,937.43和\$7,777,777.77。

(5) 分别编写一个正则表达式，表示下述各种语言。

- (a) 给定字母表 $\Sigma = \{0, 1\}$, L 是 0 和 1 交替出现构成的所有字符串的集合。
- (b) 给定字母表 $\Sigma = \{0, 1\}$, L 是包含偶数个 0 或偶数个 1 的所有字符串的集合。
- (c) 给定英语的小写字母表, L 是字母按词典顺序升序出现的所有字符串的集合。
- (d) 给定字母表 $\Sigma = \{a, b, c, d\}$, L 是字符串 $xyzwy$ 的集合, 其中 x 和 w 是 Σ 中一个或多个字符构成的字符串, y 是 Σ 中任一字符, z 是字符 z (取自字母表 Σ 之外)。(每个字符串 $xyzwy$ 包含两个单词 xy 和 wy , 二者均由 Σ 中的字母构建而成。两个单词以同一字母 y 结束。二者通过字符 z 分隔。)
- (e) 给定字母表 $\Sigma = \{+, -, \times, \div, (), \text{id}\}$, L 是对 id 使用加减乘除和括号得到的代数表达式的集合。

- (6) 分别编写一个正则表达式，来描述下列各种程序设计语言结构：
- (a) 制表符和空格的任意序列
 - (b) C语言中的注释
 - (c) 字符串常数（没有转义字符）
 - (d) 浮点数

2.4节

(7) 考虑以下的三个正则表达式：

$$(ab \mid ac)^*$$

$$(0 \mid 1)^* 1100 \quad 1^*$$

$$(01 \mid 10 \mid 00)^* \quad 11$$

(a) 使用Thompson构造法，分别为每个RE构建一个NFA。

(b) 将这些NFA转换为DFA。

(c) 最小化DFA。

(8) 证明两个RE等价的一种方法是分别为两者构建对应的最小化DFA，然后比较得到的DFA。如果两个DFA只有状态名不同，那么对应的RE是等价的。使用这种技巧，来检查下列各对RE是否等价。

(a) $(0 \mid 1)^*$ 和 $(0^* \mid 10^*)^*$

(b) $(ba)^+ (a^* b^* \mid a^*)$ 和 $(ba)^* ba^+ (b^* \mid \epsilon)$

(9) 有时候，通过 ϵ 转移连接的两个状态是可以合并的。

(a) 在哪些条件下，通过 ϵ 转移连接的两个状态是可以合并的？

(b) 给出用于消除 ϵ 转移的算法。

(c) 你的算法与用于实现子集构造法的 ϵ -closure函数有何关联？

- (10) 请说明，正则语言的集合在交集操作下是封闭的。
- (11) 图2-9中给出的DFA最小化算法，是通过while循环的各次迭代来枚举 P 的所有元素和 Σ 中所有的字符。
- (a) 重做算法，使用WorkList来保存还需要考察的各个集合。
 - (b) 重做Split函数，使之围绕 Σ 中所有的字符来划分集合。
 - (c) 比较修改后算法和原始算法的预期复杂度，结果如何？

2.5节

(12) 为下述的每种C语言结构分别构造一个DFA，然后为各个DFA构建表驱动实现所需的表：

- (a) 整数常数
- (b) 标识符
- (c) 注释

(13) 对前一习题中的每一个DFA，分别构建一个直接编码的词法分析器。

(14) 本章描述了DFA实现的几种风格。实现词法分析器的另一种方法是使用相互递归函数
(mutually recursive functions)。讨论这种实现的优点和缺点。

- 。对比这种实现的优点和缺点。
- (15) 为减小转移表，词法分析器生成器可以使用一种字符分类方案，但生成分类器表似乎代价颇高。看起来比较直接的算法将需要 $O(|\Sigma|^2 \cdot |states|)$ 时间。推导一个渐近复杂度较低的算法，来得到同样的转移表。
- (16) 图2-15给出了一种方案，可以在通过模拟DFA而构建的词法分析器中避免二次方量级的回滚行为。遗憾的是，这种方案要求词法分析器预先知道输入流的长度，且必须维护一个比特矩阵 Failed，其规模为 $|states| \times |input|$ 。设计一种方案，使得无需预先了解输入流的长度。是否可以使用同样的方案，在不出现最坏输入的情况下，来减小 Failed 表？

编译技术课程的讨论注意事项

- 就像是RE到NFA有很多种转换方法，**答案不唯一**
- 很多时候**需要**讨论来确定自己的答案和想法是否正确
- 不要盲信：包括我和所有助教/学生的答案可能都是错误的
- 遇到以上情况，**有任何不确定的感觉，直接issues里贴出来问**
 - 说不定还能帮助到同样有问题却不好意思发帖的人 ☺

新同学：可以加入本课程的微信学习群

- 群有人数限制，请在HelloGCC微信公众号输入「**旁听**」



方舟·编译技术入门与实战

视频可以碎片时间看，编程作业一定要沉下心来写才行的

有不明白的地方就及时开issues提问😊

<https://github.com/lazyparser/becoming-a-compiler-engineer>