

CH2 词法分析

2.1 scanning process 扫描处理

1.某些记号只有一个词义(lexeme): 保留字; 某些记号有无限多个语义: 标识符都由 ID 表示。

2.2 regular expression 正则表达式

- 1.*repetition+concatenation 连>alternation 选
- 2.相同的语言可以用不同的 RE 表示
- 3.R+: R(R*) R?: R|ε [abc]: a|b|c; [a-z]: (a|b|c|...|y|z); [^ab]除 a 或 b; [^a-z]除 a-z

4.RE 匹配优先匹配保留字; 最长字串优先

2.3 finite automata 有穷自动机

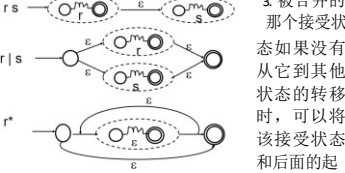
1.DFA: M 由字母表Σ、状态集 S、转换函数 T: S×Σ→S、初始状态 S₀ ∈ S 以及接受状态 Acc ⊆ S。

2.错误状态默认不画,但是存在: 错误状态下的任何转移均回到自身, 永远无法进入接受。

3.NFA: M 由字母表Σ、状态集 S、转换函数 T: S×(Σ ∪ {ε}) → P(S)、初始状态 S₀ 以及接受状态 A 的集合。

2.4 RE to DFAs 正则表达式到 DFA

2. Thompson 结构通过ε转移将 NFA glue together

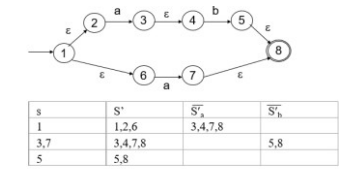


始状态合闭。

4.子集构造的过程:

首先列出所有状态的ε闭包: 然后将初始状态的ε闭包作为新的初始状态; 然后计算在每个新状态下的各个字符上的转移的闭包作为新的状态, 转移自然成为新的转移; 包含原接受状态的所有新状态都是接受状态

PS: ε闭包首先包含自身。下面步骤缺一个所有状态的ε闭包; S 代表是哪几个状态的闭包得到的 S'



4. DFA 状态数最小化: 最小状态数的 DFA 唯一。步骤: 创建两个集合, 一个包含所有接受, 另一个是剩余; 考虑每个字符上的转换, 如果所有的接受在 a 上都有到接受的转换, 或是都有到非接受的转换, 那么这就定义了一个从新接受到新非接受的转移; 如果两个接受有转移但是不在相同集合或是一个有转换, 一个没转换, 那么两个接受在该字符上被区分, 从而分割出了新的状态集合: 重复如此。

CH3 上下文无关文法与分析

3.2 CFG

1.左递归: 定义 A 的推导式的右边第一个出现的是 A; 右递归: 定义 A 的推导式邮编最后一个出现的是 A;

3.3 Parse tree and AST 分析数和抽象语法树

1. 同一个串存在多个推导即多个分析树

2. 分析树(concrete syntax tree)是一个作了标记

if (E) S1 else S2 => <code to evaluate E>; fip L1;

labeled 的树, 内部节点是非终结符, 树叶是终结符; 对于一个内部节点运用推导时, 推导结果从左到右依次成为该内部节点的子节点

3.最左推导和顺序编号对应, 最右推导后序

4.AST(syntax tree)去除了终结符和非终结符信息, 仅保留了语义信息: 一般用左孩子子兄弟

3.4 Ambiguity 二义性

- 1.定义: 带有两个不同的分析树的串的文法
- 2.解决方法①设置消歧规则 disambiguating rule. 在每个二义性情况下指出哪个是对的. 无需对文法进行修改, 但是语法结构就不是单纯依赖文法了, 还需要规则②修改文法.
- 4.修改文法时需要同时保证优先级和结合律

precedence and associativity

在语法规则中, 连接优先级, 越高, 优先级越低; 左递归导致左结合, 右递归导致右结合

6.将相同优先级的运算符分组叫做 precedence cascade 优先级联

7.通过最近嵌套规则 most closely nested rule 解决 else 悬挂问题; 另一种方案是 for else 语句使用一个括号关键字(end if {} fi 都可)

8.essential ambiguity 是无关紧要的二义性, 虽然语法规则各不相同, 但是语义相同, 例如算术加法虽然可结合但是结合顺序无关紧要

3.5 EBNF

1.A→a(b)a b 可重复, 花括号在右是左递归 A→a(b)a b 可选

CH4 自顶向下分析

第一个 L 是从左到右处理, 第二个 L 是最左推导, 1 代表仅使用一个字符预测分析方向

4.1 recursive-descent 递归下降

1.将一个非终结符 A 的文法规则看作将识别 A 的一个过程。递归下降需要使用 EBNF: 将可选[]翻译成 if, 将重复{}翻译成 while 循环

4.2 LL(1)

- 2.第一列标号: 第二列为分析站内容, 底座在左, 栈底标注\$; 第三列显示了输入, 从左到右, \$表示输入结束; 第四列为动作
- 3.动作: ①生成, 利用文法将栈顶的 N 替换成串, 串反向进栈②匹配: 将栈顶的记号和下一个输入记号匹配③错过

4.Definition of LL(1) Grammar: A grammar is and LL(1) grammar if the associated LL(1) parsing table has at most one production in each table entry. 分析表中的每个项目中至多只有一个产生式。LL(1)文法是无二义性的

6.LL(1)面对重复和选择的解决方法: 消除左递归 left recursion removal 和提取左因子 left factoring。

7.简单直接左递归: A → Aα|β, αβ ∈ N, 且 β 不以 A 开头。A → βA', A' → αA'|ε

8.普遍直接左递归:

A → Aα₁|Aα₂ ... |Aα_n|β₁|β₂ ... |β_m
A → β₁A'|β₂A' ... |β_mA'
A → α₁A'|α₂A' ... |α_nA'|ε
9.一般的左递归, 不能带有ε产生式和循环 for i:=1 to m do
for j:=1 to i-1 do
replace each grammar rule choice of the form A_i→A_jβ by the rule
A_i→ α₁|α₂ ... |α_i is the current rule for A_j
Remove if necessary immediate left recursion

involving A_i 其中 m 是 N 的个数

10.提取左因子
A → αβ|αγ. A → αA', A' → β|γ

4.3 first and follow sets

- 1.First 定义: 令 X 为一个 T 或 N 或 ε, First(X)由 T 或 ε 组成。①若 X 为 T 或 ε, First(X)={X}②若 X 为 N, 对于每个产生式 X→X₁X₂...X_n, First(X)都包含了 First(X₁)-{ε}。若对于某个 i<n, 所有的 First(X_i) ... First(X_n)都含有 ε, 则 First(X)也包括了 First(X_n)-{ε}。若所有 First(X_i) ... First(X_n)都含有 ε, 则 First(X)也包含 ε。
- 2.定理: A non-terminal A is nullable if and only if First(A) contains ε

3.Follow 定义: 若 A 是一个 N, 那么 Follow(A)由 T 和 \$ 组成。①若 A 是 S, 直接进入 Follow(A) ②若存在产生式 B→αAγ, 则 First(γ)-{ε}在 Follow(A)中 ③若存在产生式 B→αAβ, 且 ε 在 First(γ)中, 则 Follow(A)包括 Follow(B)

PS: ③更常见的情况是 B→αA, 那么 Follow(A)包括 Follow(B)

4.First 关注点在产生式左边, Follow 在右边

5.LL(1)分析表 M[N T]的构造算法: 为每个非终结符 A 和产生式 A→α 重复以下两个步骤: ①对于 First(α)中的每个记号 a, 都将 A→α 添加到项目 M[A a] 中 ②若 ε 在 First(α)中, 则对于 Follow(A)

中的每个元素 a (包括 \$), 都将 A→α 添加到项目 M[A a] 中

S→ES' S'→ε|+S E→num|(S)

	num	+	()	\$
S	S→ES'		S→ES'		
S'	S'→+S		S'→ε		S'→ε
E	E→num		E→(S)		

6.LL(1)文法的判别: A grammar in BNF is LL(1) if the following conditions are satisfied. ①For every production A_i→α₁|α₂ ... |α_n First(α_i)∩First(α_j) is empty for all i and j 1 ≤ i, j ≤ n, i ≠ j ②For every non-terminal A such that First(A) contains ε First(A)∩Follow(A) is empty.

4.5 error recovery

- 1.遇错后的不同层次反应: 给出一个错误信息后 ①尽可能准确定位②尝试进行错误矫正 error repair③分析程序从错误程序中推断 infer 出正确程序
- 2.some important considerations: ①尽快判断出错误的发生②错误发生后, 必须挑选一个位置恢复 resume 分析, 尽可能找到多的真的错误 ③避免出现错误级联(一个错牵出数个假错)④避免错误的无限循环
- 3.panic mode 应急模式, 递归下降中的错误矫正。基本机制为每个递归过程提供一个额外的由同步记号组成的参数。遇到错误是, 就向前扫描, 并且一直丢弃忘记号知道遇到一个同步记号, 从这里恢复分析。Follow 集合是同步记号中的重要一员。First 集合可以避免跳过开始新的主要结构的重要记号, 也可以在更早时候检测错误。同步记号随着递归不断传递并增加新值。
- 4.LL(1)中没有递归, 因此额外增加一个栈存同步记号, 算法生成每个动作前, 都调用 checkpoint; 或者在分析表中的空格中补全错误处理, 共有三种可能①若当前输入为\$或是

在 Follow(A)中, 将 A 从栈中弹出, 记作 pop②当输入不是\$或不在 First(A)∪Follow(A)中, 看到一个为了它可以重新开始分析的记号后, 再弹出该记号, 记作 scan③特殊情况下压入一个新的 N

CH5 自底向上分析

Yacc 基于 LALR(1): 使用显示栈完成分析。

5.1 概览

- 1.动作为①shift, 将 T 从输入开头移到栈顶②reduce 使用产生式 A→α 将栈顶的 α 规约成 A ③accept 分析栈为开始符号, 输入栈为空时的动作 ④error
- 2.注意统一额外加一个 S'作为新的开始符号
- 3.推导中的 N 和 T 的每个中间串都称作右句型 right sentential form, 这样的句型都被分析栈和输入分隔开(即使某一边空了也 OK)。在每一种情况下, 分析栈的符号序列都被称为右句型的可行前缀 viable prefix分析栈空时, 可行前缀为ε)
- 3.若语法无二义性, 则句柄唯一

5.2 FN of LR(0) items and LR(0)

1.LR(0)的项就是在右边带有区分位置的产生式, 同时就是 LR(0)的 FA 中的一个状态

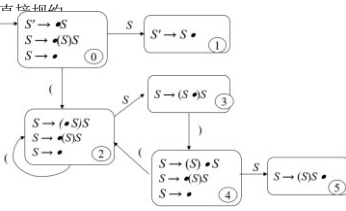
2. NFA 构造算法: 按照上述规则, 从添加的 S'→S 作为起始状态进行构造。



•If X is a nonterminal
•X will never appear as an input symbol. (such a transition will still correspond to the pushing of X onto the stack during a parse, but this can only occur during a reduction by a production X→β)

3. DFA 构造算法: 每个新状态都是一个产生式的ε闭包。其中在闭包步骤中通过ε添加到状态中的项目与引起状态的项目, 前者叫闭包项 closure item. 后者叫做核心项 kernel item. 若有一个文法, 核心项唯一判断出状态以及转换, 那么只需要指出和心想就可以完整地表示出 DFA。

4. LR(0) 分析算法的定义: Let s be the current state (at the top of the parsing stack). Then actions are defined as follow: ①If state s contains any item of the form A → α · Xβ (X is a T). Then the action is to shift the current input token onto to the stack. ② If state s contains any complete item (A → α ·) then the action is to reduce by the rule A → α. ③如果为 T 直接移进; 如果包含完整项,



5.s-r conflict: 包含了完整项的状态不能包含任何其他项目, 否则 s-r 冲突

6.r-r conflict: 两个完整项共存则出现 r-r 冲突

7. LR(0)文法不可能是二义的

8. A grammar is LR(0) if and only if ①Each state is

State	Input	Goto
	(a) A	
0	s3 s2	1
1	r1 r1 r1	
2	r2 r2 r2	
3	s3 s2	4
4		s5
5	r3 r3 r3	

a shift state (a state containing only shift items) or a reduce state (containing a single complete item). 这里的 r1 都是用 S'→S 规约, 应该写成 accept

5.3 SLR(1)

1.SLR(1)算法定义: ①If state s contains any item of form A → α · Xβ (X is T) then the action is to shift the current input token onto the stack and the new state to be pushed on the stack is the state containing the item A → αX · ②If state s contains the complete item A → α · and the next token in the input string is in Follow(A) then the action is to reduce by the rule A → α. 移进规则不变; 规约时要求输入必须在 follow 中

2.SLR(1)不可能是二义性

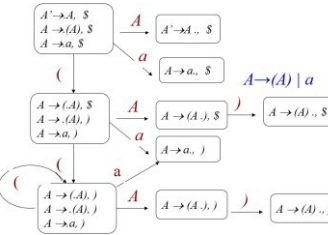
3. A grammar is SLR(1) if and only if for any state s the following two conditions are satisfied: ①For any item A → α · Xβ (X is T) there is no complete item B → γ · in s with X in Follow(B). ②For any two complete item A → α · and B → β · in s Follow(A) ∩ Follow(B) is empty. 待移进的终结符不能是完整项的 Follow 元素; 两个完整项的 Follow 集不相交

4. 自底向上分析中右递归可能引起栈溢出, 需要避免

5. SLR(1)中的两种冲突, sr 冲突使用消歧规则: 优先移进; rr 冲突基本是设计出问题

5.4 LR(1) and LALR(1)

- 1. LR(1) items: [A → α · β, a] 前面是 LR(0)项, 后面是 lookahead token
- 2. LR(1)的起始状态[S'→ · S, \$]的闭包
- 3. LR(1)转移的定义: ①非空转移 Given an LR(1) item [A → α · Xy, a] X is T or N there is a transition on X to the item [A → αX · y, a] ②空转移 Given an LR(1) item [A → α · By, a] B is a N there are ε-transitions to item [B→ · β, b] for every production B→β and for every token b in First(ya). 第一条规则永远不会创建新的先行; 实际情况中, 往往是y本身就是ε, 此时从格式 [A → α · B, a]到[B→ · β, a]可得到ε转移



4. LR(1) definition: let s be the current state (at the top of the paring stack). Then actions are defined as follows: ①If state s contains LR(1) item of the form [A → α · Xβ, a] X is T and X is the next token in the input string. ②If state s contains LR(1) item [A → α ·, a] the next token in the input stream is a. ③If the next input token is such that neither of the above two cases applies error.

5. LR(1)文法不可能二义性

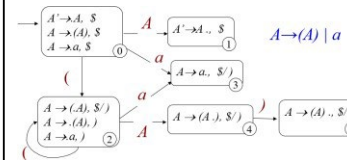
6. A grammar is LR(1) if and only if for any state s.

The following two conditions are satisfied: ①For any item [A → α · Xβ, a] X is T. There is no item in s of the form [B→ γ · X], 否则 sr 冲突 ②There are no two items in s of the form [A → α ·, a] and [B → β ·, a], 否则 rr 冲突

7. (1) A→(A), (2) A→a 的分析表

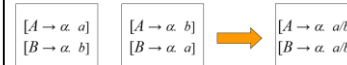
State	Input	Goto
	(A) \$	\$ S
0	s2 s3	1
1		accept
2	s5 s6	4
3		r2
4		s7
5	s5 s6	8
6		r2
7		r1
8		s9
9		r1

8. LALR(1)将先行合并



9. A grammar is an LALR(1) grammar if no parsing conflicts arise in the LALR(1) parsing algorithm.

10. 如果文法是 LR(1), 那么 LALR(1)中必然没有 sr 冲突, 但是可能有 rr 冲突。



- 11. 如果文法是 SLR(1) 那么必然是 LALR(1)。
- 12. 通过传播先行 propagating lookahead 的处理从 LR(0)项目的 DFA 直接计算出 LRAL(1)的 DFA 是可能的。

5.7 Error recovery

- 1. LR(1)比 LALR(1)或 SLR(1)更早检测出错误: LALR(1)和 SLR(1)都比 LR(0)更早
- 2. There are three possible alternative actions: ①Pop a state from the stack. ②Successively pop tokens from the input until a token is seen for which we can restart the parse. ③Push a new state onto the stack.
- 3. When an error occurs is as follows: ①Pop states from the parsing stack until a state is found with nonempty Goto entries. ②If there is a legal action on the current input token from one of the Goto states push that state onto the stack and restart the parse. ③ If there is no legal action on the current input token from one of the Goto states advance the input.