# 第6章 LR分析：一种自下而上的分析方法

移进-归约法：从输入串 → 归约到开始符

需要一个栈：没有一个可归约的串则压栈，若栈顶可归约则归约成产生式的左部；

最终串扫描完而栈内只剩一个开始符S则接受

输入串从左向右扫描

自下而上的语法分析方法相当于从下往上构建一棵语法树

关键：定义可归约的串

识别可归约串

算符优先分析方法：最左素短语：比LR快。

LR：活前缀

LR：对文法的要求比较宽泛，采用较多

分析表：

LR(k)：k：向右看k个字符

~~算符~~ 分析表：某个状态，遇到某个输入终结符，转向另一个状态：放在ACTION部分，对于$S_n$.

S表示压栈，转到n状态，$r_m$表示用第m条产生式归约。

acc表示接收。

空白表示为错

前缀：将符号串的任意合有头符号的子串，空串ε为任何串的前缀。

栈内的符号称为活前缀

句柄：最左边的直接短语，一定含对应的右边

识别活前缀：

一般扩广文法：S'→S 变成唯一接收。

活前缀怎么推导：根据产生式，一定根据某个非终结符推导

$S' \Rightarrow S \quad \varepsilon . S$

$\Downarrow$

$aAcBe$

只有的栈内容（产生式的右边）可构成一个有穷自动机

$S' \Rightarrow S \Rightarrow aAcBe \Rightarrow aAcde \Rightarrow abcde$

$\searrow aAbcde$

$\Downarrow$

$abbcde$

活前缀没有包含句柄：移进

活前缀包含句柄：归约

关心活前缀与当前句型的句柄，知道了关系就可以知道下一步的动作

文法的LR(0)项目：可以用来表示活前缀。

~~[illegible]~~（相对于某一个句柄而言）

$S \rightarrow a \cdot AcBe$：句柄为 aAcBe，当前活前缀为a

△注：空规则 $A \rightarrow \varepsilon$ 对应的项目 $A \rightarrow \cdot$

圆点右边：S终结符：移进项目

非终结符：待约项目

·在最右边：归约项目

S·：接收项目

Mo Tu We Th Fr Sa Su

MEMO NO.

DATE / /

前面构造的识别活前缀的自动机中每个状态可以对应一个LR(0)项目 (NFA)

将有空边相连的状态合并 → DFA

$S \overset{*}{\Rightarrow} \alpha A w \Rightarrow \alpha\beta_1\beta_2 w$

终结符

则称 LR(0)项目 $A \to \beta_1 \cdot \beta_2$ 对活前缀 $\alpha\beta_1$ 有效.

DFA中一个状态:
对相同的活前缀有效的项目集

定义了LR(0)项目集I的 closure:设I是文法G的LR(0)项目集,则closure(I)定义如下:

(1) $I \subseteq$ closure(I)

(2) $\{B \to \cdot\gamma \mid A \to \alpha \cdot B\beta \in \text{closure}(I)\} \subseteq$ closure(I)

LR(0)项目集的GO函数

设I是文法G的LR(0)项目集,则GO(I,X)定义如下:

$GO(I,X) = \text{closure}(\{A \to \alpha X \cdot \beta \mid A \to \alpha \cdot X\beta \in I\})$

LR(0)识别活前缀DFA.M构造方法:

文法 $G=(V_N, V_T, P, S)$ 也已扩广成文法 $G'=\{V_N \cup \{S'\}, V_T, P \cup \{S' \to S\}, S'\}$ 且 $V_N \cap \{S'\} = \emptyset$

则识别活前缀DFA $M=(K, \Sigma, f, S, Z)$.

(1) $K \subseteq$ ρ(LR(0)项目集)

(2) $\Sigma = V_N \cup V_T$

(3) $f(I,X) = GO(I,X)$, $I \in K$, $X \in \Sigma$

(4) $S = \text{closure}(S' \to \cdot S)$

(5) $Z = K$

文法G的识别活前缀DFA M的状态集.

称为文法G的LR(0)项目集规范族。

由有穷自动机可以构造出分析表:

① 移进项目: $S_n$ → 移进,并转到状态n

② 待约项目: 填写GOTO表. → GOTO表中

③ 归约项目: $r_n$ → 归约,用第n条产生式

④ 接收项目: acc (没有冲突)

构造出来的分析表中每格中只有1种,则称文法是一种LR(0)文法,或者说可用LR(0)分析方法来分析

某个项中B可能跟在A之后的终结符. Follow(A)

SLR(1)分析:简单LR(1)分析:

当有冲突时:某些情况可以消除冲突

{ 移进-归约冲突
{ 归约-归约冲突

假设文法LR(0)项目集规范族有一个上述两种冲突的项目集 $I_k$, $I_k = \{A \to \alpha \cdot a\beta, B_1 \to \gamma_1 \cdot, B_2 \to \gamma_2 \cdot, \cdots\}$

{ $\text{Follow}(B_1) \cap \text{follow}(B_2) = \emptyset$
{ $\text{Follow}(B_1) \cap \{a\} = \emptyset$ → SLR(1)分析法
{ $\text{follow}(B_2) \cap \{a\} = \emptyset$

扩展 $I_k = \{A_1 \to \alpha_1 \cdot a_1\beta_1, A_2 \to \alpha_2 \cdot a_2\beta_2 \cdots$
$B_1 \to \gamma_1 \cdot, B_2 \to \gamma_2 \cdot \cdots\}$

$\{a_1, a_2 \cdots a_n\}$ $\text{Follow}(B_i)$ 两两不相交

Mo Tu We Th Fr Sa Su

MEMO NO.

DATE / /

SLR(1)分析能力强于LR(0)：要一个LR(0)的文法一定也要一个SLR(1)文法。

非终结符报告字符号串，只有一个LR(0)项目，此为归约项目

分析过程由通过分析表来执行

LR(1)要规范的分析方法：可以保证栈内一定要活前缀。

LR(1)项目：LR(0)项目+搜索符构成

$[S' \to \cdot S, \#]$ → S之后可能跟的符号

定义LR(1)项目的闭包：

I是文法G的LR(1)项目集，closure(I)定义如下：

(1) $I \subseteq closure(I)$

(2) $\{[B \to \cdot\gamma, b] \mid [A \to \alpha \cdot B\beta, a] \in closure(I), b \in First(\beta a)\} \subseteq closure(I)$

(3) 重复(2)直至 closure 不再变大。

LR(1)项目的GO函数：

设I是文法G的LR(1)项目集，则$GO(I, X)$定义如下：

$GO(I,X) = closure(\{[A \to \alpha X \cdot \beta, a] \mid [A \to \alpha \cdot X\beta, a] \in I\})$

LR(1)的分析能力比SLR(1)和LR(0)都强

LALR(1)方法：合并同心项目集

可能会带来新的归约-归约冲突

不会带来新的移进-归约冲突

$LR(0) \subseteq SLR(1) \subseteq$ LALR(1) $\subseteq LR(1)$

LR(k)：k变大，能力变强

合并