

文法和语言

分为0, 1, 2, 3四种文法

其中1型文法又叫上下文有关文法, 2型文法为上下文无关文法, 3型文法为规范文法

一般以2型文法为主

3型文法只存在两种形式:

$A \rightarrow a$ 或 $A \rightarrow aA$

$A \rightarrow aB$ $A \rightarrow Ba$

右线性 左线性

上下文无关:

出现在左侧的一定是非终结符 仅出现在右侧的为终结符

一个文法只能对应产生一种语言 但一个语言可由不同文法构造

文法推导出的**含有终结符+非终结符**的称为句型 **仅含有非终结符**的称为句子

最左推导: 每步直接推导总是替换最左的非终结符

最右推导: 规范推导 规范句型: 右句型

最右推导的逆过程: 最左归约 (规范归约)

如果一个文法存在某个句子对应两棵不同的语法树, 则该文法是二义的

自顶向下: 推导 自底向上: 归约

子树的末端结点符号串是相对于子树根的**短语**

简单子树的末端结点组成的符号串是相对于简单子树根的**简单短语**

最左简单子树的末端结点组成的符号串是**句柄**

简单子树: 包括根结点只有两层

正规式与有穷自动机

正规式画状态转换图

左线性: (特殊记忆)

- 增设初态S 开始符号对应终态 每个非终结符为状态结点
- 对于 $A \rightarrow Ba$ 从B到A画弧 标为a
- 对 $A \rightarrow a$ 从初态S到A画弧 标为a

右线性：（正常思维）

- 增设终态Z 开始符号对应初态 每个非终结符为状态结点
- 对于 $A \rightarrow aB$ 从A到B画弧 标为a
- 对于 $A \rightarrow a$ 从A到终态Z画弧 标为a

五元组DFA： (K, Σ, M, S, F)

K：所有状态结点的集合

Σ ：弧上标号的集合

M：转换的集合，例如 $M(A, b) = B$ 表示从A到B有一条弧标为b

S：初态

F：终态集合

First集和Follow集

First集规则：相应字母在 \rightarrow 左边 查找 \rightarrow 右边的第一个符号

- $A \rightarrow aB$ 将a加入First(A)
- $A \rightarrow \epsilon$ 将 ϵ 加入First(A)
- $A \rightarrow Xa$ 把First(X)中 ϵ 去掉加入First(A)

Follow集规则：相应字母在 \rightarrow 右边 查找相应字母右边的符号

- 若A为开始符号，则Follow(A)有#
- $B \rightarrow Aa$ 将a加入Follow(A) 是终结符直接加入
- $B \rightarrow AC$ First(C)加入Follow(A) 不是终结符，加入其First集
- $B \rightarrow aA$ 或 $B \rightarrow aAC$, $C \rightarrow \epsilon$ Follow(B)加入Follow(A) 后面无符号，或有符号但可取空，将 \rightarrow 左边字母的Follow集加入

LL (1) 文法

前置知识-Select集合

- 求解方法：

1.如果 α 不能推出 ε ，则： $\text{SELECT}(A \rightarrow \alpha) = \text{FIRST}(\alpha)$

2.如果 α 推出 ε ，则： $\text{SELECT}(A \rightarrow \alpha) = (\text{FIRST}(\alpha) - \{\varepsilon\}) \cup \text{FOLLOW}(A)$

自顶向下的文法

第一个L：从左向右看

第二个L：最左推导

1:向前看一个字符

在LL(1)文法中，对于每个非终结符A的两个产生式 $A \rightarrow \alpha, A \rightarrow \beta$

满足： $\text{Select}(A \rightarrow \alpha) \cap \text{Select}(A \rightarrow \beta) = \emptyset$

(α 和 β 至多有一个能推导出 ε)

非LL(1)文法到LL(1)文法的等价变换

- 提取左公因子

- 左公因子：左前缀。例如 $A::=aAb|ab$ ，左公因子应为 a 而不是 ab

提取后，有 $A::=aA', A'::=Ab|b$

- 消除左递归

- 1.消除直接左递归（左递归变化成右递归）

考虑如下文法：

$$A \rightarrow A\alpha_1 | A\alpha_2 | \dots | A\alpha_m | \beta_1 | \beta_2 | \dots | \beta_n$$

产生的句子为：

$$\beta_i \alpha_{j_1} \alpha_{j_2} \dots$$

所以，消除递归后文法为：

$$A \rightarrow (\beta_1 | \beta_2 | \dots | \beta_n) A'$$

$$A' \rightarrow (\alpha_1 | \alpha_2 | \dots | \alpha_m) A' | \epsilon$$

- 2.消除间接左递归

经过若干步推导，将间接左递归变成直接左递归，然后再使用消除直接左递归的方式

构造预测分析表

- 构造预测分析表的步骤

(1) 对每个终结符 $a \in \text{FIRST}(A)$ ，将 $A \rightarrow a$ 加到 $M[A, a]$ 中。

(2) 如果 $\epsilon \in \text{FIRST}(A)$ ，则对于任何 $b \in \text{FOLLOW}(A)$ ，将 $A \rightarrow \epsilon$ 加到 $M[A, b]$ 中。

例：

- 例解：

==注意：下文的测试用例中使用的文法如下：==

$E \rightarrow TE'$
 $E' \rightarrow +TE' | \epsilon$
 $T \rightarrow FT'$
 $T' \rightarrow *FT' | \epsilon$
 $F \rightarrow (E) | i$

(1) 求文法对应的FIRST和FOLLOW集合

求非终结符号的First集：

$\text{First}(E) = \{ (, i \}$
 $\text{First}(E') = \{ +, \epsilon \}$
 $\text{First}(T) = \{ (, i \}$
 $\text{First}(T') = \{ *, \epsilon \}$
 $\text{First}(F) = \{ (, i \}$

非终结符的Follow集：

$\text{Follow}(E) = \{), \# \}$
 $\text{Follow}(E') = \{), \# \}$
 $\text{Follow}(T) = \{ +,), \# \}$
 $\text{Follow}(T') = \{ +,), \# \}$
 $\text{Follow}(F) = \{ *, +,), \# \}$

(2) 构造预测分析表初步：

行（非终结符号）：E, E', T, T', F

列（终结符号）：i, +, *, (,), #

填写：非终结符号U+输入符号a->下一条执行的产生式

	i	+	*	()	#
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow i$			$F \rightarrow (E)$		

个人理解：

- 如果一个非终结符A的first集中有{a,b}，那么表格[A,a]和[A,b]会添加对应内容

- 如果一个非终结符A的first集合中有 ϵ ，那么在以上基础上观察其follow集合，如果follow集合中有{c,d}，那么表格[A,c]和[A,d]会添加对应内容
- 如果一个非终结符A的first集合中没有 ϵ ，那么不会关心其follow集合

算符优先分析

- 设有一文法G，如果G中没有形如 $A \rightarrow \dots BC \dots$ 的产生式，其中B和C为非终结符，则称G为算符文法
- 设有一不含空产生式的算符文法G，如果对任意两个终结符对a,b之间至多只有优先级高于、低于和等于三种关系中的一种成立，则称G是一个算符优先文法

FirstVT和LastVT

- 首先定义两个集合FIRSTVT(B)和LASTVT(B)

$\text{FIRSTVT}(B) = \{b \mid B \Rightarrow b \dots \text{或} B \Rightarrow Cb \dots\}$ ，其中 \dots 表示 V^* 中的符号串。

$\text{LASTVT}(B) = \{a \mid B \Rightarrow \dots a \text{或} B \Rightarrow \dots aC\}$

在以上的基础上：

FirstVT：能够推导出的第一个非终结符集合

- 如果有 $b \in \text{FirstVT}(B)$ 且 $A \rightarrow B \dots$ ，那么有 $b \in \text{FirstVT}(A)$

LastVT：能够推导出的最后一个终结符集合

- 如果有 $b \in \text{LastVT}(B)$ 且 $A \rightarrow \dots B$ ，那么有 $b \in \text{LastVT}(A)$

构造预测分析表

假定 G 是一个不含 ϵ 产生式的算符文法。对于任意一对终结符 a, b ，我们说：

- $a = b$ ，当且仅当文法 G 中含有形如 $P \rightarrow \dots ab \dots$ 或 $P \rightarrow \dots aQb \dots$ 的产生式。
 - $a < b$ ，当且仅当文法 G 中含有形如 $P \rightarrow \dots aR \dots$ 的产生式，而 $R \xRightarrow{+} b \dots$ 或 $R \xRightarrow{+} Qb \dots$ 。
 - $a > b$ ，当且仅当文法 G 中含有形如 $P \rightarrow \dots Rb \dots$ 的产生式，而 $R \xRightarrow{+} \dots a$ 或 $R \xRightarrow{+} \dots aQ$ 。
- 如果存在 $P \rightarrow \dots aR \dots$ ，那么a的优先级小于FirstVT(R)中的所有元素
 - 如果存在 $P \rightarrow \dots Rb \dots$ ，那么LastVT(R)中的所有元素优先级大于b

例. $G[E]: E \rightarrow E+T | T$
 $T \rightarrow T * F | F$
 $F \rightarrow (E) | i$

算符优先级表

F	$+$	$*$	$($	$)$	i
E	✓	✓	✓		✓
T		✓	✓		✓
F			✓		✓

L	$+$	$*$	$($	$)$	i
E	✓	✓		✓	✓
T		✓		✓	✓
F				✓	✓

	$+$	$*$	$($	$)$	i
$+$	$>$	$<$	$<$	$>$	$<$
$*$	$>$	$>$	$<$	$>$	$<$
$($	$<$	$<$	$<$	$=$	$<$
$)$	$>$	$>$		$>$	
i	$>$	$>$		$>$	

素短语

- 至少包含一个终结符
- 该短语不再包含满足第一个条件的更小的短语

最左边的素短语称为最左素短语

算符优先分析过程

对比栈顶符号和输入串最左边符号的优先级，如果栈顶归约后为终结符，那么比较栈顶的第二个符号。

如果栈顶符号优先级>输入串最左边符号，那么归约

如果栈顶符号优先级<输入串最左边符号，那么移进

如果栈顶符号优先级=输入串最左边符号，那么先移进后归约

LR (0) 分析

自底向上语法分析

LR (0) 项目集规范族的构造

- 构造识别活前缀的DFA（框图）
- 构成识别一个文法活前缀的DFA项目集（状态）的全体称为这个文法的LR (0) 项目集规范族（列表）

LR (0) 项目：在文法G中产生式的右部适当位置添加一个圆点构成项目

对于空产生式： $A \rightarrow \epsilon$ 仅有一个项目 $A \rightarrow \cdot$

-
- 移进项：圆点后面为终结符
 - 待约项：圆点后面为非终结符
 - 归约项：圆点在最右端
 - 接受项（特殊归约项）：如果一个项呈 $Z \rightarrow u \cdot$ 形，且Z是识别符号

增广文法

如果G是一个以S为开始符号的文法，则G的增广文法 G' 就是在G中加上新开始符号 S' 和产生式 $S' \rightarrow S$ 而得到的文法

后继项目

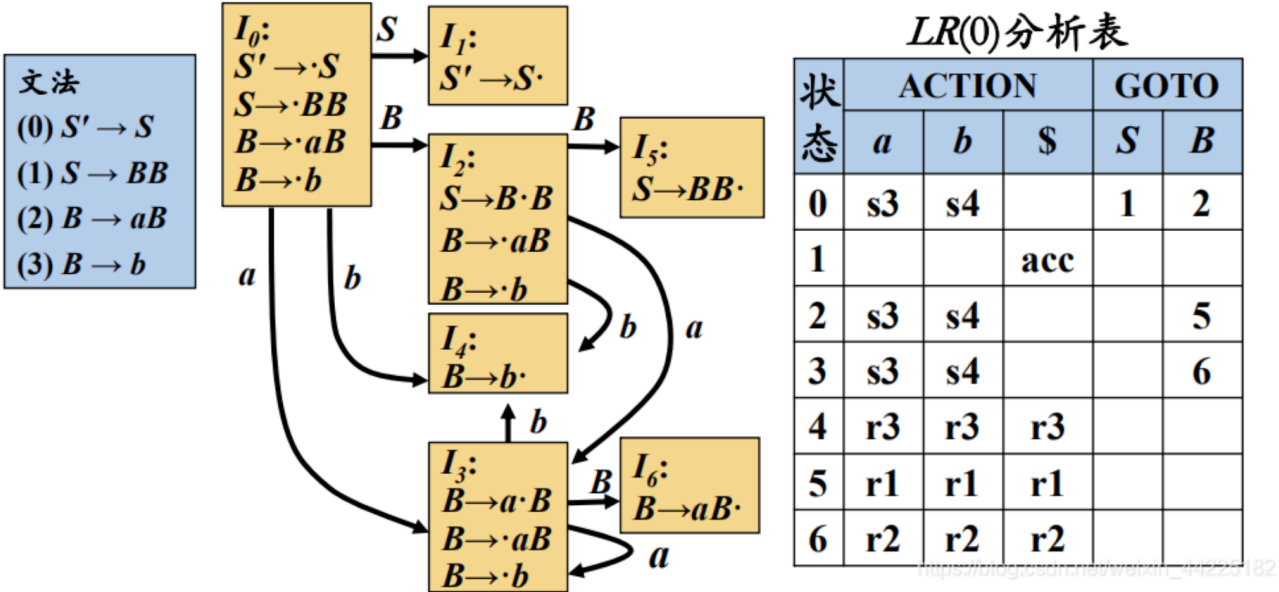
同属于一个产生式的项目，但圆点的位置只相差一个符号

$A \rightarrow a \cdot Xb$ 的后继项目是 $A \rightarrow aX \cdot b$

LR (0) 自动机

LR(0)自动机

举例说明



- $I_0 \rightarrow I_1$
输入符号S，观察当前状态中所有圆点后的符号，恰好为S的入栈
- $I_0 \rightarrow I_3$
输入符号a，其中 $B \rightarrow \cdot aB$ 符合条件，入栈后变为 $B \rightarrow a \cdot B$ ，由于圆点后面的是非终结符号B，因此要重新对该非终结符进行展开（即将 I_0 中左部为该符号的产生式入栈）

分析表中，如果下一个字符为终结符，则为ACTION；如果为非终结符，则为GOTO

其中ACTION可能为移进、归约、ACC

- 移进S：圆点后还有字符，接受该字符后进入什么状态，则为 S_i
- 归约R：圆点后无字符，该字符对应文法中的哪一条，即为 R_i
 - 此时 I_4 已经是归约项目了，无论下一个输入什么字符，都采取归约操作。通过文法发现， I_4 中的 $B \rightarrow b \cdot$ 对应于文法中的 (3) $B \rightarrow b$ ，所以 (4, a/b/\$) ==> r3 (r:归约 3: 文法中第三个式子)【这里归约的时候，不用看后面GOTO，因为GOTO中填的只能是数字】
- ACC：如 $I_1: S' \rightarrow S \cdot$ ，此时下一个字符为\$时即为ACC

GOTO仅可能出现数字，表示当前状态接受下一个字符为当前非终结符时，将进入什么状态

分析表中可能含有移进-归约冲突或归约-归约冲突。如果LR (0) 分析表中没有语法分析动作冲突，那么给定的文法就称为LR (0) 文法

LR (0) 分析过程

若对文法 G' 的产生式编号如下：

- | | |
|------------------------|------------------------|
| (0) $S' \rightarrow E$ | (4) $A \rightarrow d$ |
| (1) $E \rightarrow aA$ | (5) $B \rightarrow cB$ |
| (2) $E \rightarrow bB$ | (6) $B \rightarrow d$ |
| (3) $A \rightarrow cA$ | |

表 6.3 LR(0)分析表

状 态	ACTION					GOTO		
	a	b	c	d	#	E	A	B
0	S_2	S_3				1		
1					acc			
2			S_4	S_{10}			6	
3			S_5	S_{11}				7
4			S_4	S_{10}			8	
5			S_5	S_{11}				9
6	r_1	r_1	r_1	r_1	r_1			
7	r_2	r_2	r_2	r_2	r_2			
8	r_3	r_3	r_3	r_3	r_3			
9	r_5	r_5	r_5	r_5	r_5			
10	r_4	r_4	r_4	r_4	r_4			
11	r_6	r_6	r_6	r_6	r_6			

表 6.4 对输入串 $bccd$ # 的 LR(0)分析过程

步 骤	状态栈	符号栈	输入串	ACTION	GOTO
(1)	0	#	$bccd$ #	S_3	
(2)	03	# b	ccd #	S_5	
(3)	035	# bc	cd #	S_5	
(4)	0355	# bcc	d #	S_{11}	
(5)	0355(11)	# $bccd$	#	r_6	9
(6)	03559	# $bccB$	#	r_5	9
(7)	0359	# bcB	#	r_5	7
(8)	037	# bB	#	r_2	1
(9)	01	# E	#	acc	

- 步骤1时，状态栈为0，符号栈为#
- 每次观察状态栈栈顶和输入串左侧符号，在分析表中找到对应的动作
- 进入下一步骤时，如果是移进，则ACTION动作中的下标进入状态栈，终结符进入符号栈；如果是归约，则GOTO动作中的数字进入状态栈，非终结符进入符号栈

- 归约时，需要从状态栈和符号栈中弹出对应字符

SLR (1) 分析

基于容许LR (0) 规范族中有冲突的项目集，用向前查看一个符号的方法来解决冲突。由于仅对有冲突的状态才向前查看一个符号，因此是简单的LR (1) 分析法，即SLR (1)

假定一个 LR(0)规范族中含有如下的项目集(状态) I :

$$I = \{ X \rightarrow \alpha \cdot b\beta, A \rightarrow \gamma \cdot, B \rightarrow \delta \cdot \}$$

也就是在该项目集中含有移进-归约冲突和归约-归约冲突。其中 $\alpha, \beta, \gamma, \delta$ 为文法符号串， b 为终结符。那么只要在所有含有 A 或 B 的句型中，直接跟在 A 或 B 后的可能终结符的集合即 FOLLOW(A)和 FOLLOW(B)互不相交，且都不包含 b ，也就是只要满足

$$\text{FOLLOW}(A) \cap \text{FOLLOW}(B) = \emptyset$$

$$\text{FOLLOW}(A) \cap \{b\} = \emptyset$$

$$\text{FOLLOW}(B) \cap \{b\} = \emptyset$$

那么，当在状态 I 时，如果面临某输入符号为 a ，则动作可按以下规定决策：

- (1) 若 $a=b$ ，则移进。
- (2) 若 $a \in \text{FOLLOW}(A)$ ，则用产生式 $A \rightarrow \gamma$ 进行归约。
- (3) 若 $a \in \text{FOLLOW}(B)$ ，则用产生式 $B \rightarrow \delta$ 进行归约。
- (4) 此外，报错。

处理一个有冲突的项目集时，其中的每一个句型都会产生一个集合：

- 对于归约句，等式左部非终结符的Follow集合
- 对于移进句，圆点后面的非终结符

需要保证这些集合两两不相交

如果对于一个文法的LR (0) 项目集规范族的某些项目集或LR (0) 分析表中所含有的动作冲突都能用上述方法解决，则这个文法是SLR (1) 文法

SLR (1) 分析表

在LR (0) 分析表里，ACTION中如果某一状态对应的一个非终结符为归约，那么该行一定均为归约操作，原因如下：

- LR (0) 认为，当前语句为归约型时，不论下一个符号是什么，都采取归约操作
- 而SLR (1) 中，对于 $A \rightarrow \alpha$ ，仅当下一个符号属于 Follow(A) 时，才会在SLR (1) 分析表中写入归约操作

LR (1) 分析

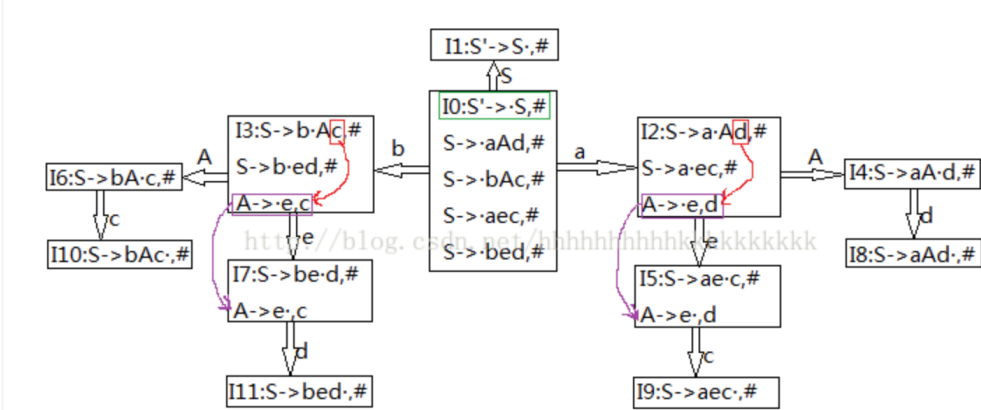
在SLR (1) 中，只凭Follow集合判断是否采用归约不合适，SLR (1) 依然可能存在语法冲突。

LR (1) 项目由两部分组成，一部分为LR (0) 项目，一部分为向前搜索符集合

初始 S_0 状态的项目集为: $[S' \rightarrow S \cdot, \#]$

向前搜索符:

课本的答案是对的, 但是写法很是让我们一头雾水, 下面让我们来看看答案是怎么出来的:



- 一开始, 绿框处 S' 是整个句子, 所以后面理所当然跟句子结束符 $\#$ 。然后 S 后面是 ϵ (就是什么都没有) 所以 $\beta = \epsilon$, 接着逗号后面是 $\#$ 即 $a = \#$, 这样 $FIRST(\beta a) = FIRST(\epsilon \#) = \{\#\}$ 。这就是 I_0 中 S 后面 $\#$ 号的来历。
- 在 I_2 中, 如红线所示, 在 $I_2: S \rightarrow a \cdot A d, \#$ 中 A 的后面是 d , 所以 $FIRST(d \#) = \{d\}$ (就是“ $d \#$ ”的第一个终结符 d), 所以接下来 A 的后面跟的是 d 。

在 I_2 中, 有项目1: $S \rightarrow a \cdot A d$, 项目2: $A \rightarrow \cdot e$, 其中项目2中的 A 是从项目1中得来的, 因此需要看项目1中 A 后面的字符串: d , 以及项目1的向前搜索符 $\#$, 两个拼接后得到 $\{d \#\}$ 的First集即位项目2的向前搜索符

LALR分析

将LR (1) 分析中的同心 (除展望符外, 两个LR (1) 项目集是相同的) 项目进行简化

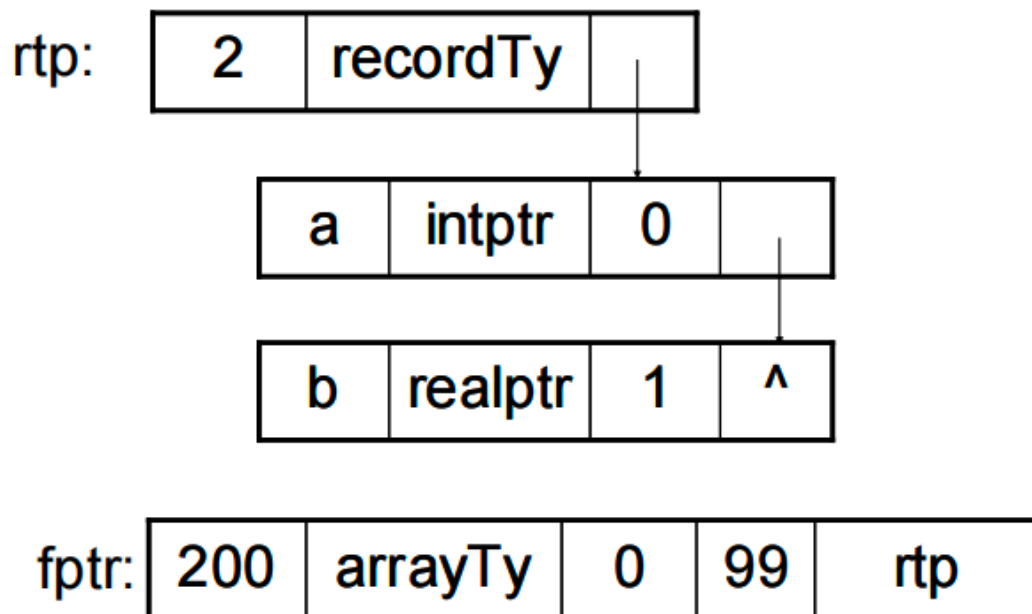
没有归约-归约冲突

符号表与类型表

三、(10 分) 设当前层数为 L , 可用偏移量值为 $off1$, 且有下面 C 程序段, 试完成相关的符号表和类型表。

```
typedef struct {
    int a;
    float b;
} CELL;
CELL f1[100];
```

CELL	rtp	typekind			
f1	fptr	varkind	dir	L	off1



源程序的中间表示

四元式中间代码

- 算术法

四元式：（操作符，左操作数，右操作数，结果）

三元式：（操作符，运算对象1，运算对象2）

四、（10 分）试写出下列表达式的四元式（或三地址代码）中间代码，假设运算符优先级由高到低依次为-（负号）、*、/、+、-，且均为左结合。

$$x = -(a*b) + (c+d) - (a*b-c)$$

- (1) $(*, a, b, t1)$
- (2) $(-, t1, _, t2)$
- (3) $(+, c, d, t3)$
- (4) $(+, t2, t3, t4)$
- (5) $(*, a, b, t5)$
- (6) $(-, t5, c, t6)$
- (7) $(-, t4, t6, t7)$
- (8) $(=, t7, _, x)$

- 程序法

3、设有 C 语言部分程序：

```
{
    int  i;
    int  max;
    i=1;      max=1;
    while (i<=10)
    {
        max=max*i;
        i=i+1;
    }
}
```

将其中的语句翻译成四元式序列。

(MOV,1,i)	(MOV,1,max)
(LABEL, inL)	(<=,i,10,T1)
(JUMP0,outL)	(* ,max,i,T2)
(MOV,T2,max)	(+,i,1,T3)
(MOV,T3,i)	(JUMP,inL)
(LABEL,outL)	