编译原理与技术

计算机科学与技术学院
李诚

USTC, CHINA
ADSLAB

3.16 用文法

S-> (L) | a

L-> L, S | S

(a) 构造(a, (a, a))的最右推导,

(b) 给出对应(a)的最左推导的移进归约分析器的步骤。

Assignment 6 —— 3.16b

栈	输入	动作
$	(a,(a,a))$	移进
$(	a,(a,a))$	移进
$(a	,(a,a)$	归约
$(s	,(a,a)$	归约
$(L	,(a,a)$	移进
$(L,	(a,a)$	移进
$(L,(	a,a)$	移进
…	…	…
…	…	…
$(L)	$	归约
$S	$	

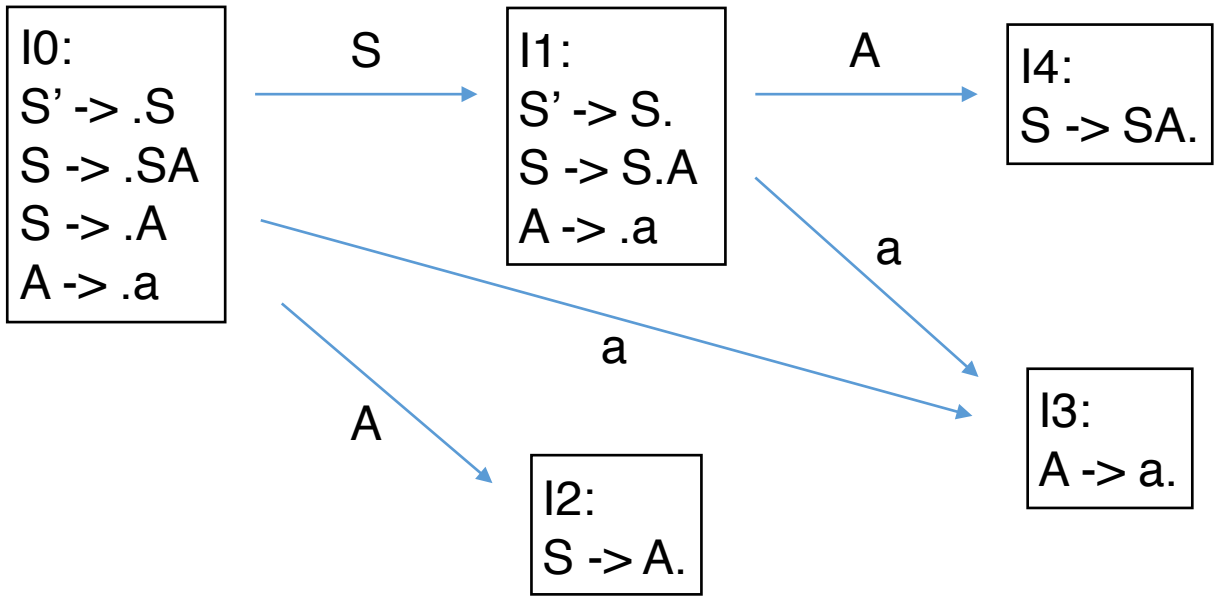
3.20 证明下面的文法

S --> S A | A

A --> a

是SLR(1)文法，但不是LL(1)文法。

活前缀DFA：



SLR分析表：

状态	动作		转移	
	a	$	S	A
0	S3		1	2
1	S3	acc		4
2	r1	r1		
3	r2	r2		
4	r1	r1		

没有冲突，故该文法为SLR(1)文法

First(SA) = First(A) = {a}

故该文法不是LL(1)文法

3.27 文法G的产生式如下：

S --> I | R　　I --> d | I d　　R --> WpF

W --> Wd | \epsilon　　F --> F d | d

(a) 令d表示任意数字，p表示十进制小数点，那么非终结符S, I, R, W和F在编程语言中分别表示什么？

(b) 该文法是LR(1)文法吗？为什么？

(a)

S: 整数或浮点数

I：整数

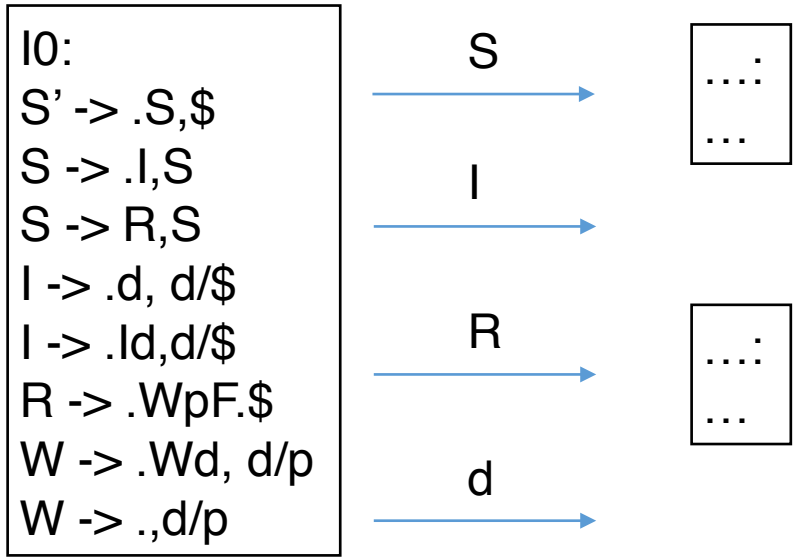
R：浮点数

W：浮点整数部分

F：浮点数小数部分

Assignment 7 —— 3.27

(b)



(b) SLR分析表：

状态	动作			转移				
	d	p	$	S	I	R	F	W
0	s4/r8	r8		1	2	3		5
…								
…								

有冲突，故该文法不是SLR(1)文法

3.30 描述文法

S —> aSbS | a S | \epsilon

产生的语言，并为此语言写一个LR（1）文法。

b的数量小于a的数量的ab组成的串。

S —> aLbL

L —> aT

T —> S | \epsilon

3.24 证明下面文法

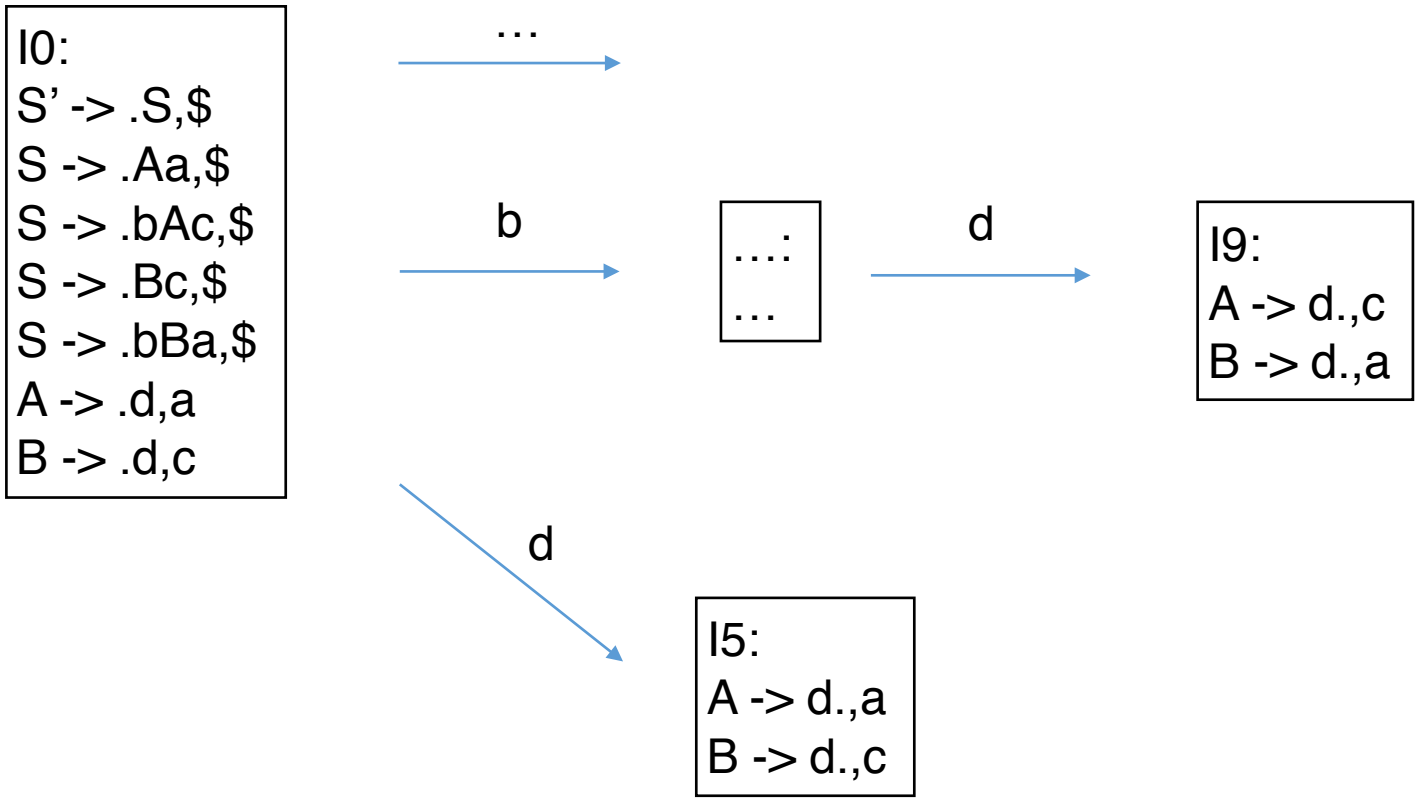
S —> Aa | bAc | Bc | bBa

A —> d

B —> d

是LR(1)文法，但不是LALR(1)文法。

Assignment 8 —— 3.24



Assignment 8 —— 3.24

状态	动作					转移		
	a	b	c	d	$	S	A	B
0		s3		s5		1	2	4
1					acc			
…								
5	r5		r6					
…								
9	r9		r5					
…								

没有冲突，故该文法为LR(1)文法

合并状态5，9时有冲突，故该文法不是LALR(1)文法

3.37 下面是一个二义文法

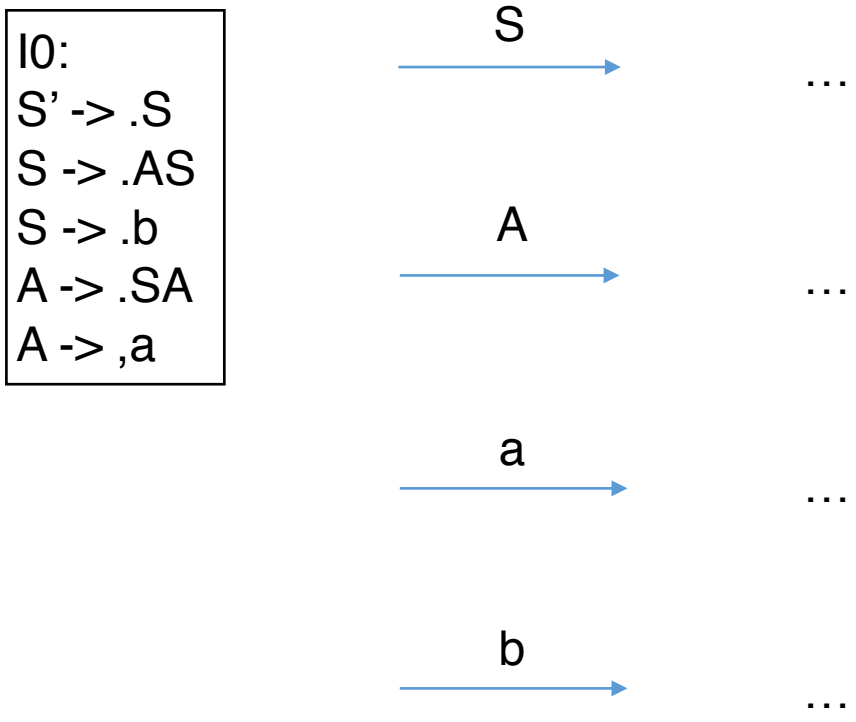
S --> AS | b

A --> S A | a

如果为该文法构造LR分析表，则一定存在某些有分析动作冲突的条目，它们是哪些?

假定分析表这样来使用：出现冲突时，不确定地选择一个可能的动作。给出对于输入abab所有可能的动作序列。

Assignment 8 —— 3.37



Assignment 8 —— 3.37

状态	动作			转移	
	a	b	$	S	A
0	s4	s3		1	2
1	s4	s3	acc	6	5
2	s4	s3		7	2
3	r2	r2	r2		
4	r4	r4			
5	s3,r3	s3,r3		7	2
6	s4	s3		6	5
7	s4,r1	s3,r1	r1	6	5

abab:

(1) r4 -> s3 -> r2 -> s4 -> r4 -> s3 -> r2 -> r1

(2) r4 -> s3 -> r2 -> s4 -> r4 -> r3 -> s3 -> r2 -> r1

(3) r4 -> s3 -> r2 -> r1 -> s4 -> s3 -> r2 -> r1

(4) r4 -> s3 -> r2 -> r1 -> s4 -> r3 -> s3 -> t2 -> r1 -> acc

4.3 为文法

S --> (L) | a

L --> L , S | S

(a) 写一个语法制导定义，它输出括号的对数。

(b) 写一个语法制导定义，它输出括号嵌套的最大深度。

(a)

S’ —> S	print S.num
S —> (L)	S.num = L.num + 1
S —> a	S.num = 0
L —> L’,S	L.num = L’.num + S.num
L —> S	L.num = S.num

(b)

S’ —> S	print S.depth
S —> (L)	S.depth = L.depth + 1
S —> a	S.depth = 0
L —> L’,S	L.depth = max(L.depth + S.depth)
L —> S	L.depth = L.depth

4.14 程序的文法如下：

P —> D

D —> D;D | id : T | proc id;D;S

(a) 写一个语法制导定义，打印该程序一共声明了多少个id。

(b) 写一个翻译方案，打印该程序每个变量id的嵌套深度。

(a)

P’ —> P	print(P.num)
P —> D	P.num = D.num
D —> D1;D2	D.num = D1.num + D2.num
D —> id:T	D.num = 1 + T.num
D —> proc id;D’;S	D.num = 1+ D’.num + S.num

(b)

P —> {D.depth = 1} D

D —> {D1.depth = D.depth} D1; {D2.depth = D.depth} D2

D —> id:T {print id,D.depth}

D —> proc id;D'{D'.depth = D.depth + 1}; S

Q&A

编译原理与技术

计算机科学与技术学院
李诚

Assignment 11

- 4.15

Assignment 11 – 4.15

- 题目：下面是构造语法树的一个S属性定义。将这里的语义规则翻译成LR翻译器的栈操作代码段。

- $E \rightarrow E_1 + T \quad E.nptr = mkNode('+', E_1.nptr, T.nptr)$
- $E \rightarrow E_1 - T \quad E.nptr = mkNode('-', E_1.nptr, T.nptr)$
- $E \rightarrow T \quad E.nptr = T.nptr$
- $T \rightarrow (E) \quad T.nptr = E.nptr$
- $T \rightarrow \mathbf{id} \quad T.nptr = mkLeaf(\mathbf{id}, \mathbf{id}.entry)$
- $T \rightarrow \mathbf{num} \quad T.nptr = mkLeaf(\mathbf{num}, \mathbf{num}.entry)$

Assignment 11 – 4.15

产生式	代码段
$E \rightarrow E$	$ck[top].val$
$E \rightarrow E$	$ck[top].val$
$E \rightarrow$	
$T \rightarrow$	val
$T \rightarrow \mathbf{id}$	
$T \rightarrow \mathbf{num}$	

注意：栈操作代码段不是SDT（语法制导翻译方案）！

Assignment 12

- 附加题

Assignment 12 – 附加题

- 题目：考虑以下消除左递归后的算数表达式文法

$$E \to TR \quad R \to +TR_1 \quad R \to \varepsilon$$
$$T \to FW \quad W \to *FW1 \quad W \to \varepsilon$$
$$F \to (E) \quad F \to \mathbf{id} \quad F \to \mathbf{num}$$

1. 写一个语法制导定义，计算表达式的值（提示： 需要使用继承属性）
2. 为上述语法制导定义写一个语法制导翻译方案
3. 将语法制导定义中的语义规则翻译成LR分析器 的栈操作代码

Assignment 12 – 附加题

产生式	语义规则
$E \to TR$	$R.i = T.nptr \;\; E.nptr = R.s$
$R \to +TR_1$	$R_1.i = mkNode(\text{`}+\text{'}, R.i, T.nptr)$ $R.s = R_1.s$
$R \to \varepsilon$	$R.s = R.i$
$T \to FW$	（同上）
$W \to *FW1$	
$W \to \varepsilon$	
$F \to (E)$	$F.nptr = E.nptr$
$F \to \mathbf{id}$	$F.nptr = mkLeaf(\mathbf{id}, \mathbf{id}.entry)$
$F \to \mathbf{num}$	$F.nptr = mkLeaf(\mathbf{num}, \mathbf{num}.val)$

Assignment 13

- 5.4 5.5 5.6

Assignment 13 – 5.4

- 为下列类型写类型表达式:

1. 指向实数的指针数组，数组的下标从0到99
2. 二维数组（即元素类型为一维数组的数组），它的行下标从0到9，列下标从0到19
3. 函数，它的定义是从整数到整数指针的函数，它的值域是由一个整数和一个字符组成的记录。

Assignment 13 – 5.4

1. Array(0..99, real)
2. Array(0..9, pointer(array(0..19, elemtype)))
3. Function(Function(integer→pointer(integer))
   →record(integer*character))

Assignment 13 – 5.5

- 5.5 假如有下列C的声明:

```
typedef struct{
        int a, b;
} CELL, *PCELL;
CELL foo[100];
PCELL bar(x, y) int x; CELL y; {}
```

为变量foo和函数bar的类型写出类型表达式。

Assignment 13 – 5.5

- array(0..99, record((a × integer) ×(b × integer)))

- (integer ×record((a × integer) ×(b × integer))) → pointer(record((a × integer) ×(b × integer)))

Assignment 13 – 5.6

- 下列方法定义字面常量表的表。

$$P \rightarrow D; E$$
$$D \rightarrow D; D | \mathbf{id}: T$$
$$T \rightarrow \mathbf{list\ of}\ T | \mathbf{char} | \mathbf{integer}$$
$$E \rightarrow (L) | \mathbf{literal} | \mathbf{num} | \mathbf{id}$$
$$L \rightarrow E, L | E$$

写一个类似5.3节中的翻译方案，以确定表达式(E)和表(L)的类型。

Assignment 13 – 5.6

- （仿照5.3.3节）

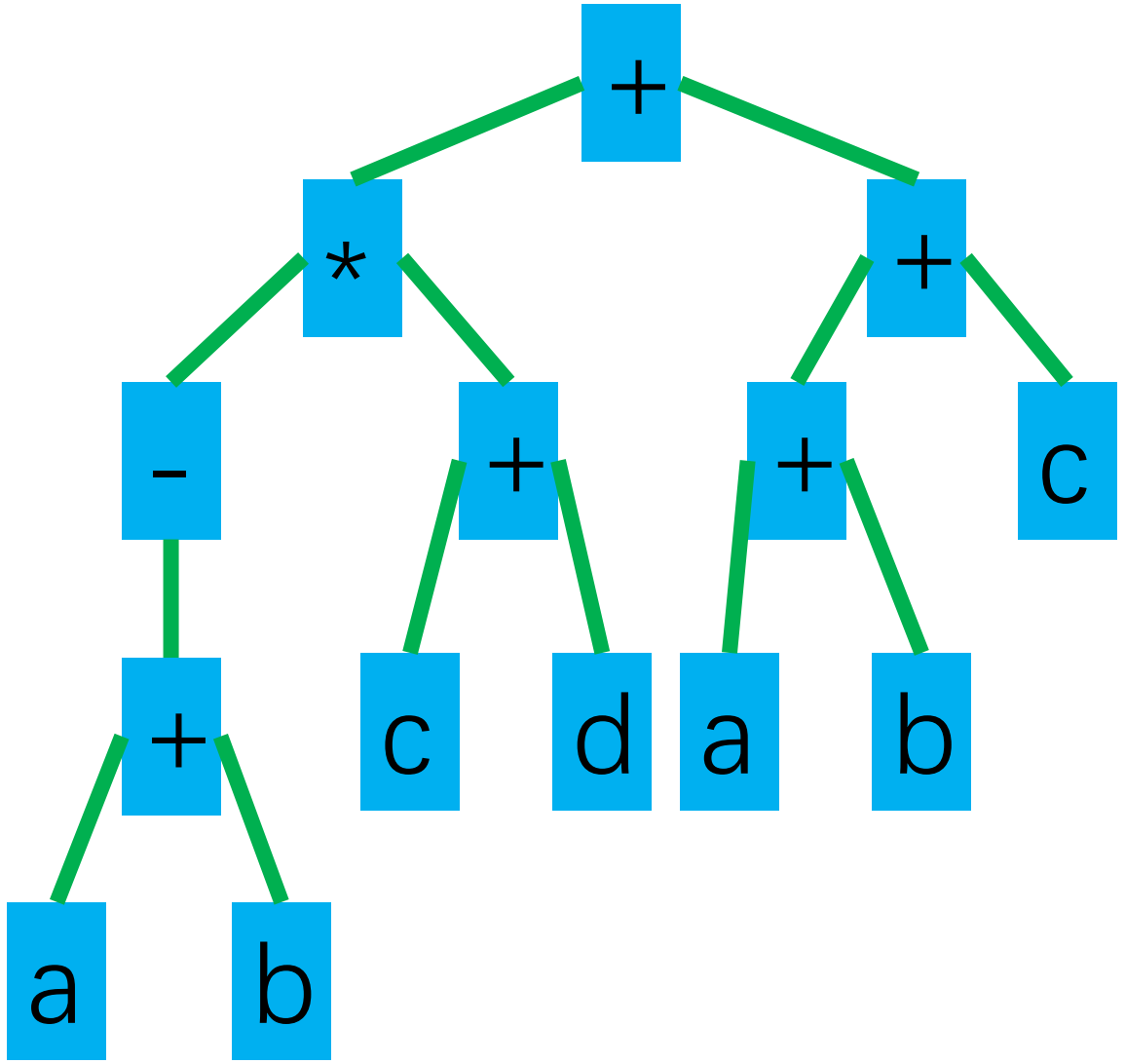
- $T \rightarrow list\ of\ T_1\ \{T.type = list(T_1.type)\}$

Assignment 14

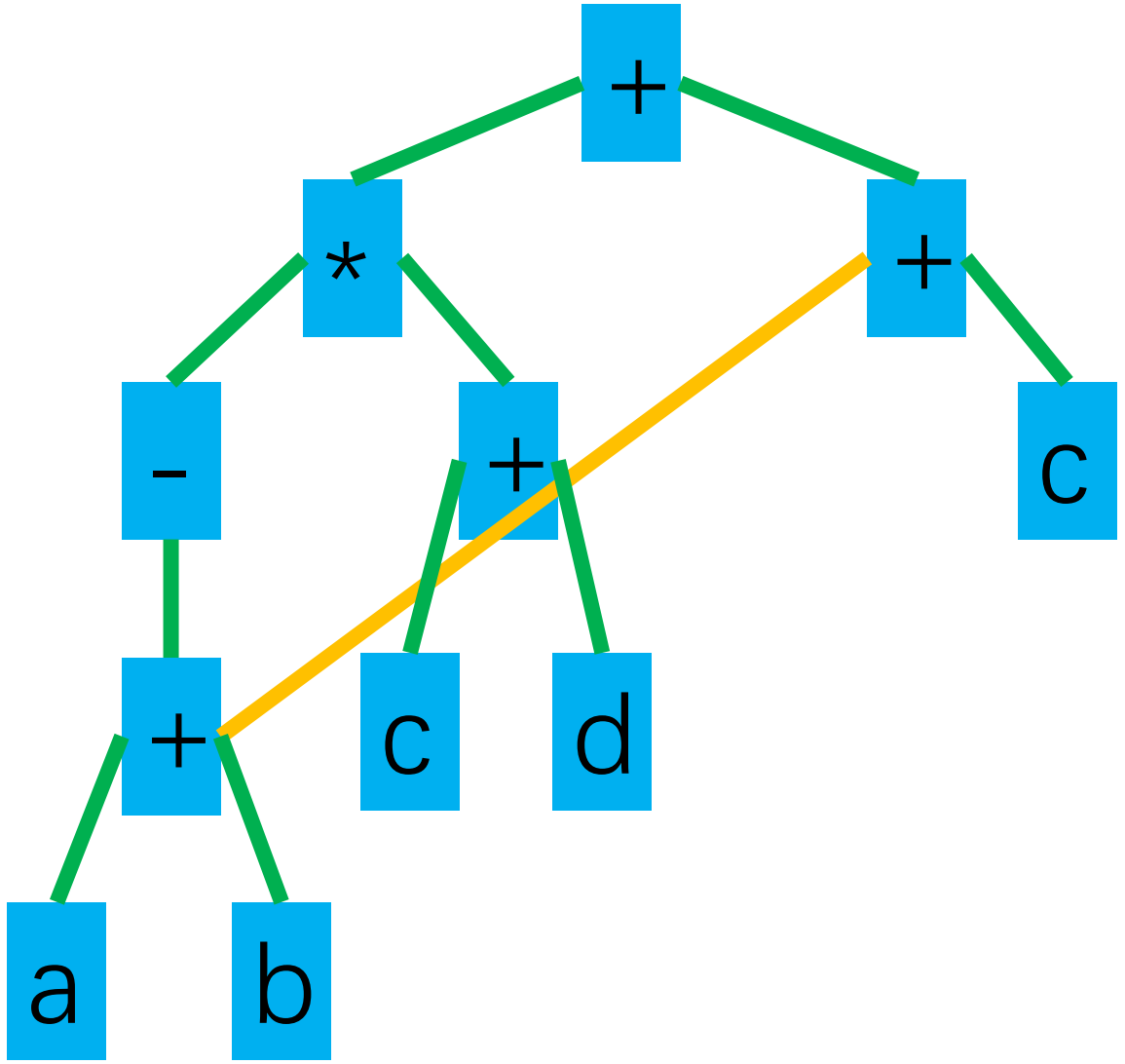
- 7.1 把算术表达式$-(a+b)*(c+d)+(a+b+c)$翻译成：

1. 语法树
2. 有向无环图
3. 后缀表示
4. 三地址代码

Assignment 14 – 7.1



Assignment 14 – 7.1



Assignment 14 – 7.1

- 后缀表示：ab+-cd+*ab+c++
- 三地址代码:

  t1=a+b

  t2=-t1

  t3=c+d

  t4=t2*t3

  t5=a+b

  t6=t5+c

  t7=t4+t6

Assignment 15

- 5.12 5.13 5.9b 5.21 7.2

Assignment 15 – 5.12

- 5.12 拓展5.3.3节的类型检查，使之能包含记录。有关记录部分的类型和记录域引用表达式的语法如下：

$$\mathrm{T} \rightarrow \boldsymbol{record}\ fields\ \mathbf{end}$$

$$fields \rightarrow fields; field \mid field$$

$$field \rightarrow \mathbf{id} : T$$

$$E \rightarrow E.\mathbf{id}$$

Assignment 15 – 5.12

- 5.12 拓展5.3.3节的类型检查，使之能包含记录。有关记录部分的类型和记录域引用表达式的语法如下：

$\mathrm{T} \rightarrow \boldsymbol{record}\ fields\ \mathbf{end}$ {T.type = record(fields.type)}

$fields \rightarrow fields; field$ {fields.type = fields.type × field.type}

$fields \rightarrow field$ {fields.type = field.type}

$field \rightarrow \mathbf{id} : T$ {field.type = id.name × T.type}

$E \rightarrow E_1.\mathbf{id}$ {E.type = if(E1.type == record(t))
lookup(E1.type,
id.name)
else
type_error;}

Assignment 15 – 5.13

- 5.13在文件stdlib.h中，关于qsort的外部声明如下：

extern void qsort(void *, size_t, size_t, int (*)(const void *, const void *));

用SPARC/Solaris C编译器编译下面的C程序时，错误信息如下：

type.c:24: warning: passing argument 4 of `qsort' from incompatible pointer type

请你对该程序略作修改，使得该警告错误能消失，并且不改变程序的结果。

```
#include <stdlib.h>

typedef struct{
            int     Ave;
            double  Prob;
}HYPO;

HYPO  *astHypo;
int  n;

int HypoCompare(HYPO *stHypo1, HYPO *stHypo2)
{
    if (stHypo1->Prob>stHypo2->Prob){
       return(-1);
    }else if (stHypo1->Prob<stHypo2->Prob) {
        return(1);
    }else{
        return(0);
    }
}/* end of function HypoCompare */

main()
{
   qsort ( astHypo,n,sizeof(HYPO),HypoCompare);
}
```

Assignment 15 – 5.13

- 5.13在文件stdlib.h中，关于qsort的外部声明如下：

extern void qsort(void *, size_t, size_t, int (*)(const void *, const void *));

问题：qsort的第四个形式参数类型与函数调用的传参类型不一致

```
#include <stdlib.h>

typedef struct{
                int     Ave;
                double  Prob;
}HYPO;

HYPO  *astHypo;
int  n;

int HypoCompare(HYPO *stHypo1, HYPO *stHypo2)
{
    if (stHypo1->Prob>stHypo2->Prob){
       return(-1);
    }else if (stHypo1->Prob<stHypo2->Prob) {
        return(1);
    }else{
        return(0);
    }
}/* end of function HypoCompare */

main()
{
   qsort ( astHypo, n, sizeof(HYPO), HypoCompare);
}
```

Assignment 15 – 5.13

- 5.13在文件stdlib.h中，关于qsort的外部声明如下：

extern void qsort(void *, size_t, size_t, int (*)(const void *, const void *));

问题：qsort的第四个形式参数类型与函数调用的传参类型不一致

方法一：修改HypoCompare函数形式参数的类型

```
#include <stdlib.h>

typedef struct{
                    int     Ave;
                    double  Prob;
}HYPO;

HYPO  *astHypo;
int  n;
```

int HypoCompare(const void *stHypo1, const void*stHypo2)

```
{
   if ((HYPO *)stHypo1->Prob>(HYPO *)stHypo2->Prob){
      return(-1);
   }else if ((HYPO *)stHypo1->Prob<(HYPO *)stHypo2->Prob) {
       return(1);
   }else{
       return(0);
   }
}/* end of function HypoCompare */

main()
{
   qsort ( astHypo, n, sizeof(HYPO), HypoCompare);
}
```

Assignment 15 – 5.13

- 5.13在文件stdlib.h中，关于qsort的外部声明如下：

extern void qsort(void *, size_t, size_t, int (*)(const void *, const void *));

问题：qsort的第四个形式参数类型与函数调用的传参类型不一致

方法二：强制修改qsort函数调用中第四个参数的类型

```
#include <stdlib.h>

typedef struct{
                    int     Ave;
                    double  Prob;
}HYPO;

HYPO  *astHypo;
int  n;

int HypoCompare(HYPO *stHypo1, HYPO *stHypo2)
{
   if (stHypo1->Prob>stHypo2->Prob){
      return(-1);
   }else if (stHypo1->Prob<stHypo2->Prob) {
      return(1);
   }else{
      return(0);
   }
}/* end of function HypoCompare */

main()
{
  qsort ( astHypo, n, sizeof(HYPO), int (*)(const void *, const void *) HypoCompare);
}
```

Assignment 15 – 5.9b

- 5.9 修改5.3.3节的翻译方案，使之能处理布尔表达式。加上逻辑算符and, or及not和关系算符的产生式。然后给出适当的翻译规则，它们检查这些表达式的类型。

Assignment 15 – 5.9b

- 产生式略。（仿照书5.3.3小节来写）

- 检查类型的翻译规则核心:
- 都是布尔值时才能做and,or,not操作，否则类型错误。

Assignment 15 – 5.21

- 5.21 使用例5.9的规则，确定下列哪些表达式有唯一类型（假定z是复数）：

  (a) 1*2*3

  (b) 1 * (z *2)

  (c) (1 * z) * z

Assignment 15 – 5.21

- 5.21 使用例5.9的规则，确定下列哪些表达式有唯一类型（假定z是复数）：

  (a) 1*2*3

  (b) 1 * (z *2)

  (c) (1 * z) * z

- 运算规则:
  - int × int -> int
  - int × int -> complex
  - complex × complex -> complex

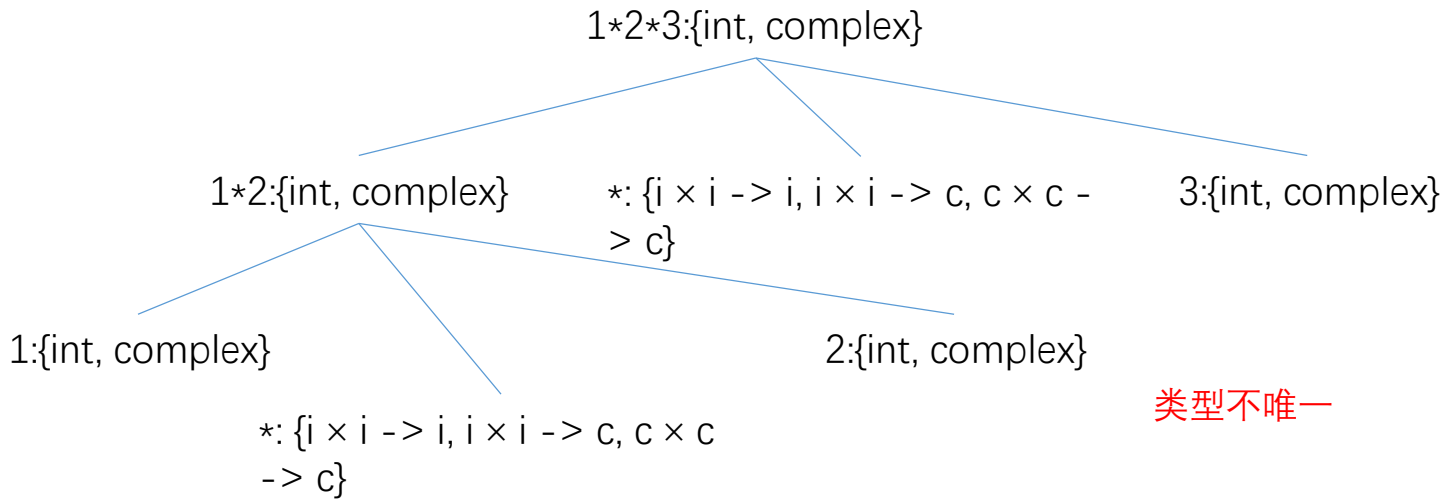
Assignment 15 – 5.21

- 5.21 使用例5.9的规则，确定下列哪些表达式有唯一类型（假定z是复数）：

  (a) 1*2*3

  (b) 1 * (z *2)

  (c) (1 * z) * z
- 运算规则：
  - int × int -> int
  - int × int -> complex
  - complex × complex -> complex



Assignment 15 – 5.21

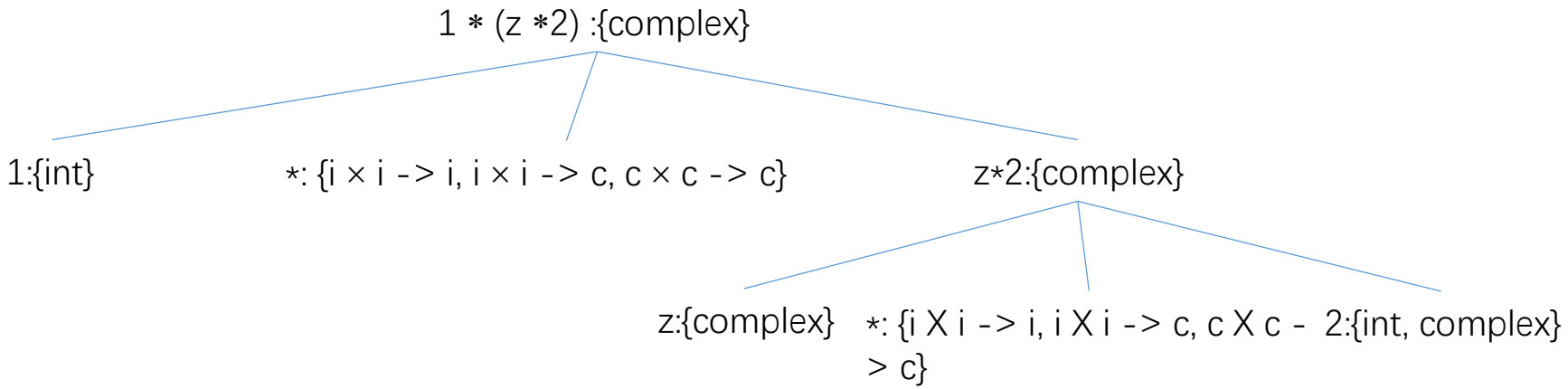
- 5.21 使用例5.9的规则，确定下列哪些表达式有唯一类型（假定z是复数）：

  (a) 1*2*3

  (b) 1 * (z *2)

  (c) (1 * z) * z

- 运算规则：
  - int × int -> int
  - int × int -> complex
  - complex × complex -> complex



类型唯一

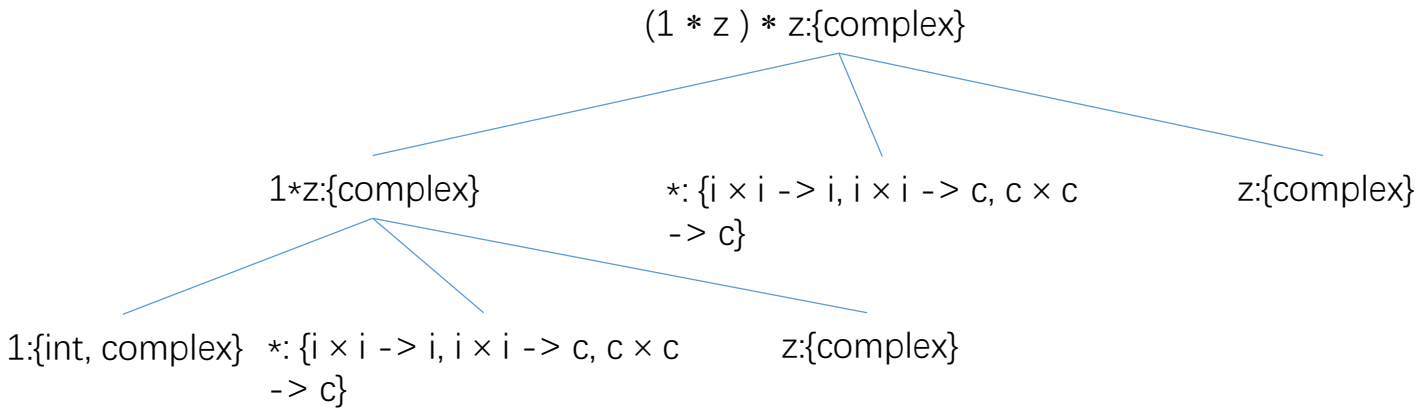
Assignment 15 – 5.21

- 5.21 使用例5.9的规则，确定下列哪些表达式有唯一类型（假定z是复数）：

  (a) 1*2*3

  (b) 1 * (z *2)

  (c) (1 * z) * z
- 运算规则：
  - int × int -> int
  - int × int -> complex
  - complex × complex -> complex



类型唯一

Assignment 15 – 7.2

- 7.2 把C程序

```
main() {
        int i;
        int a[10];
        while ( i <= 10)
                a[i] = 0;
}
```

的可执行语句翻译成:

语法树、后缀表示、三地址代码

Assignment 15 – 7.2

- 语法树图略。

- 后缀表达式：i 10 ≤ a i array 0 = while

1. if i ≤ 10 goto 3
2. goto 5
3. a[i] = 0
4. goto 1
5. return 0

Assignment 16—7.5

修改7.5中的翻译方案，使得offset不是全局变量而是继承属性

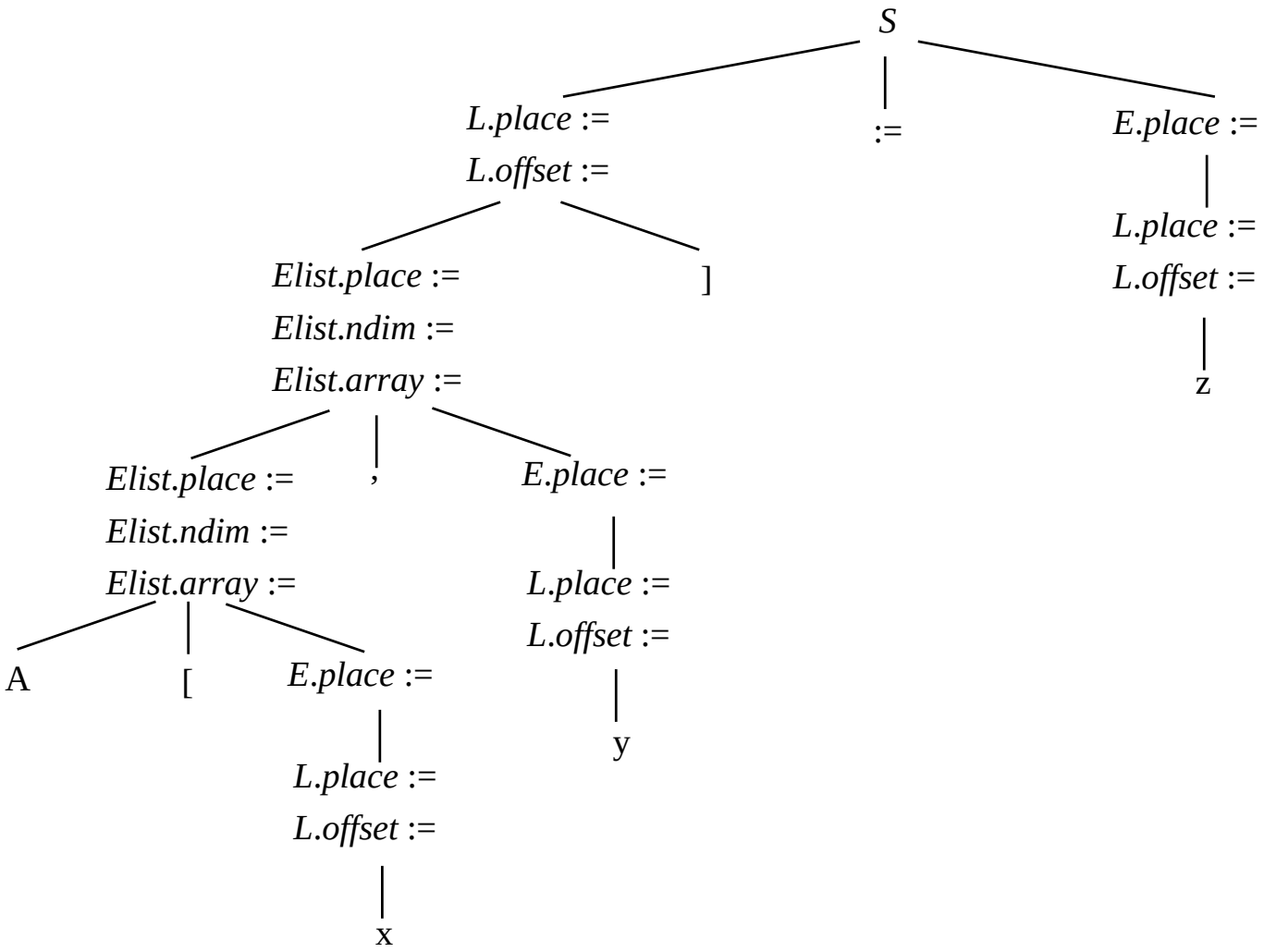
P -> {$D.offset_1$ =0;} D {$P.offset = D.offset_2$;}
D-> {$D_1.offset_1 = D.offset_1$;} $D_{1;}$
{$D_2.offset_1 = D_1.offset_1$;} $D_{2;}$ {$D.offset_2 = D_2.offset_2$;}

D-> **id** : T {enter(**id**.name, T.type, $D.offset_1$); $D.offset_2 = D.offset_1$ + T.width ;}
T-> **intergr** {T.type = intergr; T.width = 4}
T-> **real** {T.type = real; T.width = 8}

D 的 $offset_1$ 为继承属性，表示分析前D所使用变量的offset大小
D 的 $offset_2$ 为综合属性，表示分析后D所使用变量的offset大小
P 的 offset 为综合属性，记录该过程所分配的空间

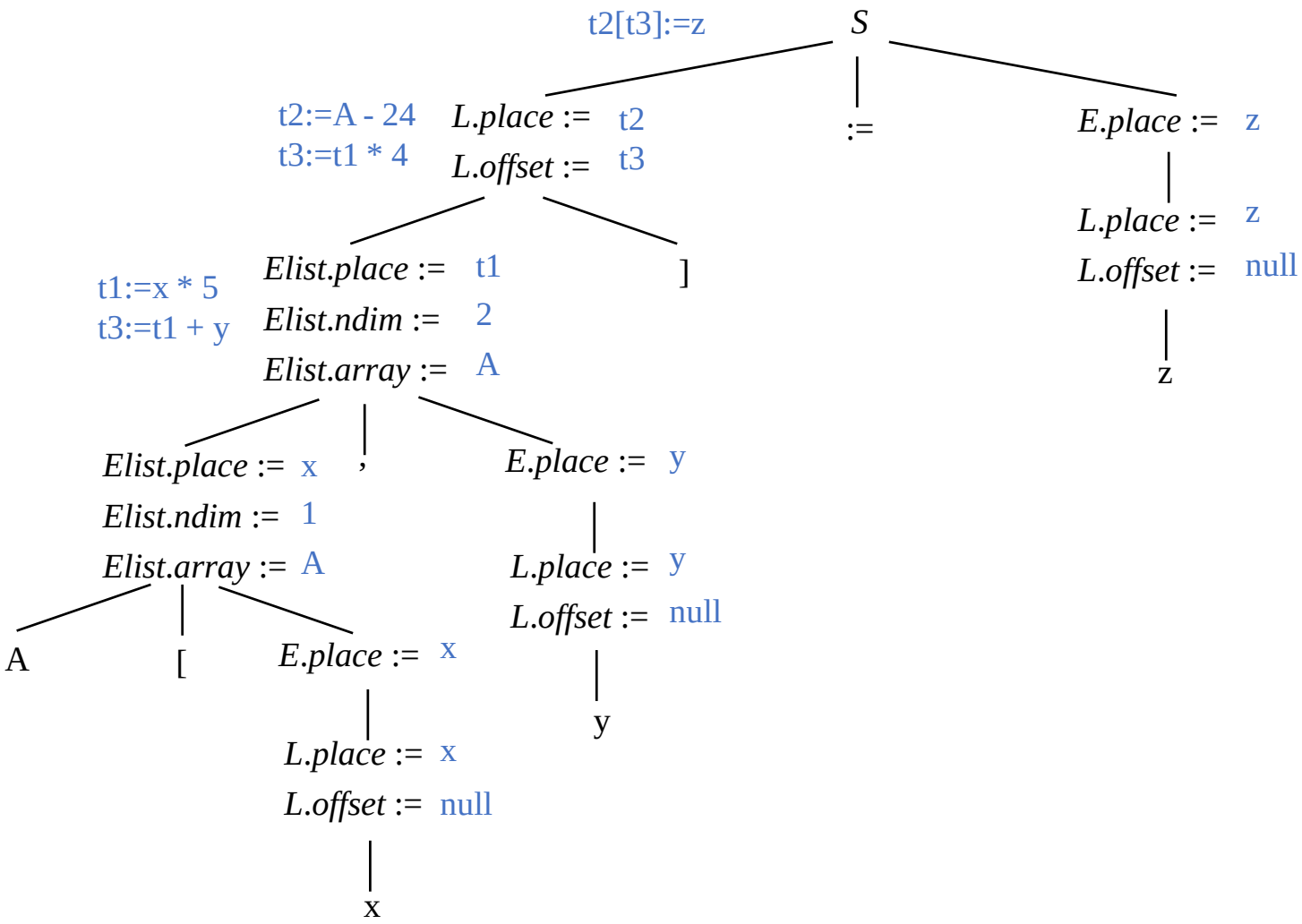
Assignment 17—7.16

补全图7.3中赋值语句A[x,y]:=z注释分析书的属性值与每步规约的中间代码



Assignment 17—7.16

补全图7.3中赋值语句A[x,y]:=z注释分析书的属性值与每步规约的中间代码



Assignment 17—7.17

C语言和Java语言的数组声明和数组元素引用的语法形式同7.3.节讨论的不一样，例如float A[10][20]和A[i+1][j-1]，并且每一维的下界都是0。若适应这种情况的赋值语句的文法如下：

S --> L := E

E --> E + E | (E) | L

L --> L [E] | id

a）重新设计7.3.2节数组元素的地址计算公式，以方便编译器产生数组元素地址计算的中间代码。不要忘记每一维的下界都是0

b）重新设计数组元素地址计算的翻译方案。只需写出产生式L --> L [E] | id的翻译方案，但要能和7.3.3节中产生式S --> L := E和E --> E + E | (E) | L的翻译方案衔接。若翻译方案中引入新的函数调用，要解释这些函数的含义

Assignment 17—7.17

a）重新设计7.3.2节数组元素的地址计算公式，以方便编译器产生数组元素地址计算的中间代码。不要忘记每一维的下界都是0

对于一个二维数组，$A[i_1][i_2]$的地址计算公式：

$base + i_1 * w_1 + i_2 * w_2$

w_1 是存放一行元素的字节数（可以认为其是一个二维元素）

w_2 是存放行中一个元素的字节数（可以认为其是一个一维元素）

推广到多维： $A[i_1][i_2]\ldots[i_k]$的地址计算公式：

$base + i_1 * w_1 + i_2 * w_2 + \ldots + i_k * w_k$

w_k 是存放一个k 维元素的字节数

Assignment 17—7.17

b）重新设计数组元素地址计算的翻译方案。只需写出产生式L --> L [E] | id的翻译方案，但要能和7.3.3节中产生式S --> L := E和E --> E + E | (E) | L的翻译方案衔接。若翻译方案中引入新的函数调用，要解释这些函数的含义

L -> L_1 [E]
{L.array = L_1.array; L.ndim = L_1.ndim + 1;
  w = getwidth(L.array, L.ndim);
  if (L.ndim == 1) **begin**
    L.offset = newtemp;
    emit (L.offset, '=', E.place, '*', w);
  end else begin
    t = newtemp; L.offset = newtemp;
    emit (t, '=', E.place, '*', w);
    emit (L.offset, '=', L_1.offset, '+', t);
  end}

L -> **id** {L.place = **id**.place; L.offset = **null**; L.ndim = 0; L.array = **id**.place;}

Assignment 18—7.8

表7.3的语法制导定义把B-> E1< E2翻译成一对语句
if E1.place< E2.place goto …
goto ...
可以用一个语句
if E1.place>= E2.place goto...
来代替，当B是真时执行后继代码。修改表7.3的语法制导定义，使之产生这种性质的代码。

Assignment 18—7.8

需要修改 E→ E1 or E2 和 E→ id1 relop id2 两个产生式的语法制导定义。

E→ E1 or E2

E1.true = E.true ;

E1.false = newlable ;

E2.true = E.true ;

E2.false = E.false ;

E.code = E1.code || gen (‘goto’ E1.true)|| gen (E1.false ‘:’) || E2.code

E→ id1 relop id2

E.code = gen (‘if’ id1.place conreop id2.place ‘goto’ E.false)

conrelop 为原 relop 关系运算符的 ‘非’ 关系运算符。

Assignment 18—7.10

1)为什么会出现1条指令前有多个标号的情况

答: 编译器每次遇到可能会跳转到此处的地方会预留一个标记，当后面的代码需要跳转时又会在相应的地方添加标记，所以会有多条。

2)每个函数都有这样的标号.L1，它可能有什么作用？为什么本函数未引用它

答: L1 是预留以用来返回的，本函数没有return，所以没有用到L1

Assignment 19—7.9

为C语言程序

```
main(){
    int i, j;
    while ((i || j) && ( j>5 )){
    i = j;
}
}
```

的汇编代码中有关指令后加注释，判断是否确实短路计算

Assignment 19—7.9

主要关心跳转指令附近的代码，为以下代码进行注释

```
.L2:
        cmpl $0, -4(%ebp)  //计算i的bool值
        jne .L6            //真转L6
        cmpl $0, -8(%ebp)  //计算j的bool值
        jne .L6            //真转L6
        jmp .L6            //(i || j)假则转L5
        .p2align 4 , , 7
.L6:
        cmpl $5, -8(%ebp)  //计算j>5的bool值
        jg .L4             //真转L4
        jmp .L5            //假转L5
       .p2align 4 , , 7
.L5:
        jmp .L3            //(i || j)&&(j>5)假时转L3, 与while翻译有关
       .p2align 4 , , 7
```

可看出是短路计算

Assignment 20—6.2

一个含有3文件的C程序如下
head.h:
short int a = 10;

file1.c:
#include "head.h"
main(){}

file2.c:
#include "head.h“

gcc file1.c file2.c
报错:multiple definition of ‘a’

Assignment 20—6.2

可以采取以下解决方案防止犯错

```
#ifndef __XX_H__
#define __XX_H__

#endif
```

Assignment 20—6.5

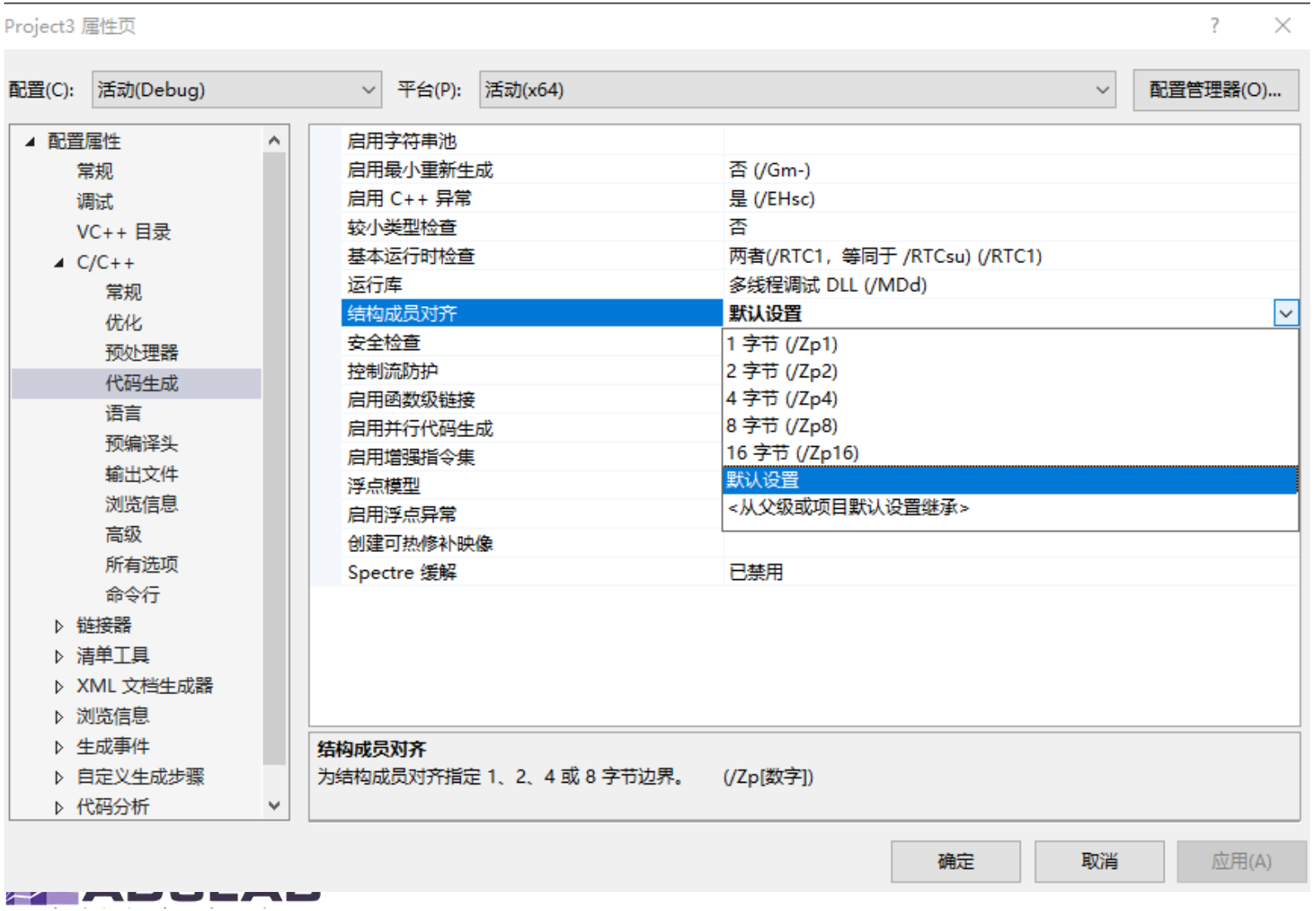
```
typedef struct a{
short i;
short j;
short k;
}a;

typedef struct b{
long i;
short k;
}b;

main(){
printf("size of short, long, a and b = %d,%d,%d,%d\n", sizeof(short),sizeof(long),sizeof(a),sizeof(b));
}
```

已知short和long分别对齐2和8的倍数，为什么b的size为16

答：为方便数组计算时，每个元素能对其到该元素长度倍数，所以取最大元素倍数

Assignment 20—6.5

(但你也可以打开VS里面结构体成员对齐的设置，会有不一样的情况哦)



Assignment 20—6.6

下面是C语言两个函数f和g的概略（它们不再有其它的局部变量）：

```
int f (int x) { int i; …return i +1; … }
int g (int y) {int j; … f (j +1); … }
```

请按照教材上例6.5中图6.13的形式，画出函数g调用f，f的函数体正在执行时，活动记录栈的内容及相关信息，并按图6.12左侧箭头方式画出控制链。假定函数返回值是通过寄存器传递的。

Assignment 21—6.9

c函数定义如下

```
int f(int x, int* py,int** ppz){
        **ppz+=1; *py+=2;x+=3;
        return x+*py+**ppz;
}
```

a为指向b指针，b为指向c指针，c为int,值为4
求f(c,b,a)返回值

答案：c+=1;c+=2;x+=3;(此时c=7,x=7)
return x+c+c=21

Assignment 21—6.10

一个 C 语言程序如下：

```
func ( i1, i2, i3) long i1, i2, i3;{
long j1, j2, j3;
printf( "Addresses of i1, i2, i3 = %o, %o, %o \n", &i1, &i2, &i3); printf( "Addresses of j1, j2, j3 = %o, %o, %o \n",
&j1, &j2, &j3);
}
main(){
long i1, i2, i3; func(i1, i2, i3);
}
```

该程序在 X86/Linux 机器上的运行结果如下：

```
Addresses of i1, i2, i3 = 27777775460, 27777775464, 27777775470
Addresses of j1, j2, j3 = 27777775444, 27777775440, 27777775434
```

func 函数的 3个形式参数的地址依次升高，而 3 个局部变量的地址依次降低。
试说明为什么会有这个区别。注意，输出的数据是八进制的

Assignment 21—6.10

一个 C 语言程序如下：

```
func ( i1, i2, i3) long i1, i2, i3;{
long j1, j2, j3;
printf( "Addresses of i1, i2, i3 = %o, %o, %o \n", &i1, &i2, &i3); printf( "Addresses of j1, j2, j3 = %o, %o, %o \n",
&j1, &j2, &j3);
}
main(){
long i1, i2, i3; func(i1, i2, i3);
}
```

答案：局部变量与参数虽然均保存在活动记录栈中，但保存位置不同，进栈次序也不同
实参进栈次序是i3, i2, i1，保存在栈底，栈向低地址增长，所以i3 i2 i1依次增大
局部变量j1, j2, j3在栈顶部，且依次进栈，所以j1, j2, j3地址逐渐减小

Assignment 21—6.13

```
int fact(i)                          | main()
int i;                               | {
{     if(i==0)                       |     printf("%d\n", fact(5));
            return 1;                |      printf("%d\n", fact(5,10,15));
      else                           |     printf("%d\n", fact(5.0));
            return i*fact(i-1);      |     printf("%d\n", fact());
}                                    | }
```

该程序在X86/Linux机器上的运行结果如下：

```
120
120
1
Segmentation fault (core dumped)
```

Assignment 21—6.13

1)第二次调用为什么没有受参数过多的影响?

参数表达式逆序计算并进栈，fact取到了第一个参数，后面两个参数无空间保存，所以运行结果与第一次相同

Assignment 21—6.13

1)第二次调用为什么没有受参数过多的影响?

参数表达式逆序计算并进栈，fact取到了第一个参数，后面两个参数无空间保存，所以运行结果与第一次相同

2）第三次调用为什么结果变成1?

将5.0以二进制表示
01000000000101000000000000000000000000000000000000000000000000
取后四个字节，即是0，输出1

Assignment 21—6.13

1)第二次调用为什么没有受参数过多的影响?

参数表达式逆序计算并进栈，fact取到了第一个参数，后面两个参数无空间保存，所以运行结果与第一次相同

2）第三次调用为什么结果变成1?

将5.0以二进制表示
01000000000101000000000000000000000000000000000000000000000000
取后四个字节，即是0，输出1

3)第四次调用为什么会出现Segmentation fault?

由于没有提供参数，而main函数又无局部变量，fact把老ebp（控制链）（main的活动记录中保存的ebp）当成参数，它一定是一个很大的整数，使得活动记录栈溢出

Q&A

祝大家期末加油！

Q&A