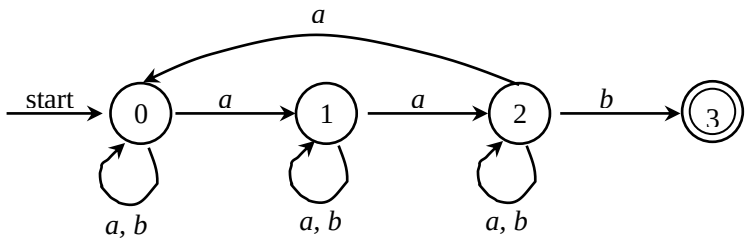
中 国 科 学 技 术 大 学

2006－2007 学年第二学期考试试卷（A）

考试科目：编译原理和技术　　得分：_____

学生所在系：_____ 姓名：_____ 学号：_____

1、（10 分）用子集构造法给出由下面的 NFA 得到的 DFA 的转换表



2、（20 分）（1）通过构造识别活前缀的 DFA 和构造分析表，来证明文法 E → E + **id** | **id** 是 SLR(1)文法。

（2）下面左右两个文法都和（1）的文法等价

E → E + M **id** | **id**　　E → M E + **id** | **id**

M → ε　　M → ε

请指出其中有几个文法不是 LR(1)文法，并给出它们不是 LR(1)文法的理由。

3、（10 分）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。

4、（10 分）C 语言函数 f 的定义如下：

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

变量 a 是指向 b 的指针，变量 b 是指向 c 的指针，c 是整型变量并且当前值是 4。那么执行 f (c, b, a)的返回值是多少？

5、（10 分）Java 语言的实现通常把对象和数组都分配在堆上，把指向它们的指针分配在栈上，依靠运行时的垃圾收集器来回收堆上那些从栈不可达的空间（垃圾）。这种方式提高了语言的安全性，但是增加了运行开销。编译时能否采用一些技术，以降低垃圾收集所占运行开销？概述你的方案。

6、（5 分）在面向对象语言中，编译器给每个对象分配空间的第 1 个域存放虚方法表的指针。是否可以把虚方法表指针作为最后 1 个域而不是第 1 个域？请简要说明理由。

7、（15 分）考虑一个简单语言，其中所有的变量都是整型（不需要显式声明），并且仅包含赋值语句、读语句和写语句。下面的产生式定义了该语言的语法（其中 **lit** 表示整型常量；OP 的产生式没有给出，因为它和下面讨论的问题无关）。

Program → StmtList
StmtList → Stmt StmtList | Stmt
Stmt → **id** := Exp; | read (**id**); | write (Exp);
Exp → **id | lit** | Exp OP Exp

我们把不影响 write 语句输出值的赋值（包括通过 read 语句来赋值）称为无用赋值，写一个语法指导定义，它确定一个程序中出现过赋予无用值的变量集合（不需要知道无用赋值的位置）和没有置初值的变量集合（不影响 write 语句输出值的未置初值变量不在考虑之中）。

非终结符 StmtList 和 Stmt 用下面 3 个属性（你根据需要来定义其它文法符号的属性）：

（1）uses_in：在本语句表或语句入口点的引用变量集合，它们的值影响在该程序点后的输出。

（2）uses_out：在本语句表或语句出口点的引用变量集合，它们的值影响在该程序点后的输出。

（3）useless：本语句表或语句中出现的无用赋值变量集合。

8、（10 分）一个 C 文件 array.c 仅有下面两行代码：

```
char a[ ][4] = { “123”, “456”};
char *p[ ]   = { “123”, “456”};
```

从编译生成的下列汇编代码可以看出对数组 a 和指针 p 的存储分配是不同的。试依据这里的存储分配，为置了初值后的数组 a 和指针 p 写出类型表达式。

```
        .file   “array.c”
.globl a
        .data
        .type a, @object
        .size a, 8
a:
        .string “123”
        .string “456”
        .section    .rodata
.LC0:
        .string “123”
.LC1:
        .string “456”
.globl p
        .data
        .align 4
        .type p, @object
        .size p, 8
p:
```

```
.long .LC0
.long .LC1
.section        .note.GNU-stack, “”, @progbits
.ident      “GCC: (GNU) 3.3.5 (Debian 1:3.3.5-13)”
```

9、（10 分）两个 C 文件 long.c 和 short.c 的内容分别是

```
long i = 32768*2;
```

和

```
extern short i;
main() { printf(“%d\n”, i); }
```

在 X86/Linux 机器上，用 cc long.c short.c 命令编译这两个文件，能否得到可执行目标程序？若能得到目标程序，运行时是否报错？若不报错，则运行结果输出的值是否为 65536？若不等于 65536，原因是什么？

中 国 科 学 技 术 大 学

2006－2007 学年第二学期（A）

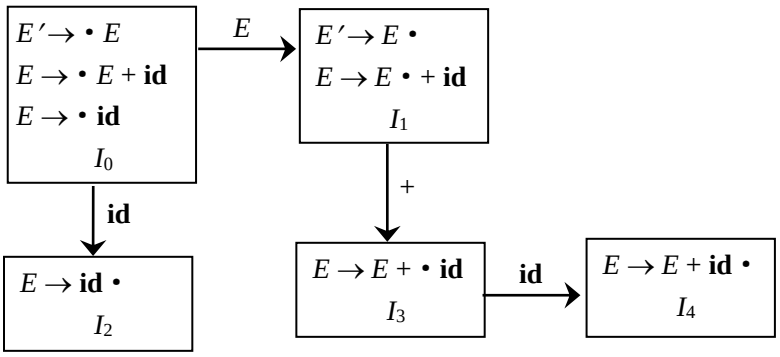
编译原理和技术参考答案

1、$A = \{0\}$, $B = \{0, 1\}$, $C = \{0, 1, 2\}$, $D = \{0, 1, 2, 3\}$

转换表

状　态	输　入　符　号	
	a	*b*
A	*B*	*A*
B	*C*	*B*
C	*C*	*D*
D	*C*	*D*

2、（1）先给出接受该文法活前缀的 DFA 如下：



再构造 SLR 分析表如下：

状态	动作			转移
	id	+	$	*E*
0	*s*2			1
1		*s*3	*acc*	
2		*r*2	*r*2	
3	*s*4			
4		*r*1	*r*1	

表中没有多重定义的条目，因此该文法是 SLR(1)的。

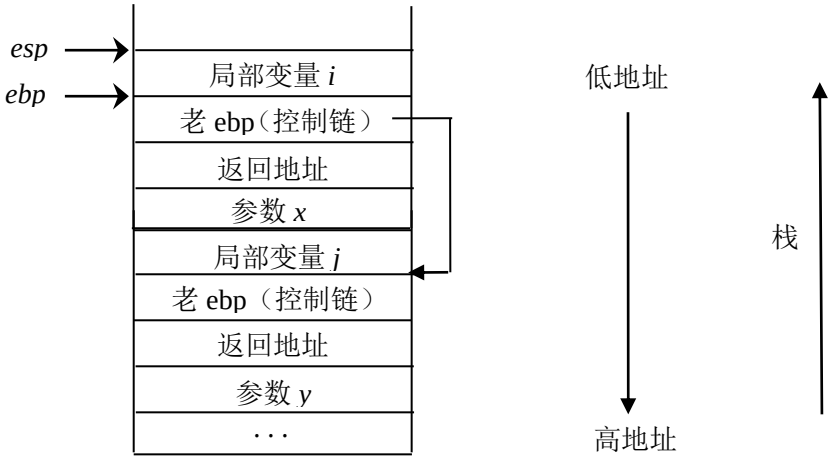
（2）只有文法

E → M E + **id** | **id**

M → ε

不是 LR(1)文法。因为对于句子 **id**+**id**+…+**id** 来说，分析器在面临第一个 **id** 时需要做的空归约次数和句子中+**id** 的个数一样多，而此时句子中+**id** 的个数是未知的。

3、



4、参数传递使得 x 的值和 c 的值一样，都是 4；py 的值和 b 的值一样，指向 c；ppz 的值和 a 的值一样，指向 b。

ppz +=1 的执行使得 c 的值为 5。*py +=2 的执行使得 c 的值为 7。x +=3 的执行使得 x 的值为 7。这时 x、*py 和ppz 的值都是 7，因此返回值是 21。

5、从两方面考虑。

（1）减少垃圾：用程序分析技术确定数据的生存期是否超过它的静态作用域所对应的生存期，若没有超过则将这样的数据分配在栈上，而不必分配在堆上。

（2）尽量静态确定垃圾：用程序分析技术把能静态确定的垃圾都标注出来，可使得运行时对它们回收开销很小。

6、不行。按书上目前的分配方式，若作为最后 1 个域，则难以为该对象产生它超类的视图。

7、 Exp 的属性 uses 表示它引用的变量集合。Program 的 useless 和 no_initial 分别表示程序的无用赋值变量集合和未置初值变量集合。

Program → StmtList	StmtList.uses_out:=∅;
	Program.useless := StmtList.useless;
	Program.no_initial := StmtList.uses_in;
StmtList → Stmt $StmtList_1$	$StmtList_1$.uses_out := StmtList.uses_out;
	Stmt.uses_out := $StmtList_1$.uses_in;
	StmtList.uses_in := Stmt.uses_in
	StmtList.useless := $StmtList_1$.useless ∪ Stmt.useless;
StmtList → Stmt	Stmt.uses_out := StmlList.uses_out;
	StmlList.uses_in := Stmt.uses_in;
	StmtList.useless := Stmt.useless
Stmt → **id** := Exp;	Stmt.useless :=**if id**.lexeme∈Stmt.uses_out **then** ∅ **else**{**id**.lexeme};
	Stmt.uses_in := **if id**.lexeme∈Stmt.uses_out
	then (Stmt.uses_out–{**id**.lexeme})∪Exp.uses **else** Stmt.uses_out
Stmt → read (**id**);	Stmt.useless:=**if id**.lexeme∈Stmt.uses_out **then** ∅ **else**{**id**.lexeme};

	Stmt.uses_in := Stmt.uses_out – {**id**.lexeme}
Stmt → write (Exp);	Stmt.useless := ∅, Stmt.uses_in := Stmt.uses_out ∪ Exp.uses
Exp → **id**	Exp.uses:= {**id**.lexeme}
Exp → **lit**	Exp.uses:= ∅
Exp → Exp_1 OP Exp_2	Exp.uses:= Exp_1.uses ∪ Exp_2.uses

8、置了初值后的数组 a 和指针 p 的类型表达式分别是：

array(0..1, string)和 array(0..1, pointer(string))

或者写成：

array(0..1, array(0..3, char))和 array(0..1, pointer(array(0..3, char)))

9、编译以文件为单位，不会发现文件之间的类型错误。由于数据的类型信息没有附加在目标文件中，因此连接时也不会发现两个文件中变量 i 的类型不一致。

运行时不会报错，但输出结果不是 65536，而是 0。这是由于 X86 机器对于整型数据来说，是低地址放整数的低位，高地址放整数的高位。在 short.c 文件中，short i 取的是 long.c 文件中 long i 的那两个低位字节，它们都是 0，因此输出结果是 0。