2007

一、简答题（共　15 分。）

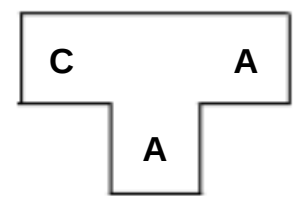
1. 通过合并 LR(1) 文法中的同心状态得到的 LALR(1) 文法可能会产生哪些冲突？一定不会产生哪些冲突？为什么？（ 5 分）

答：可能会产生归约 -归约冲突，一定不会产生移进 -归约冲突。

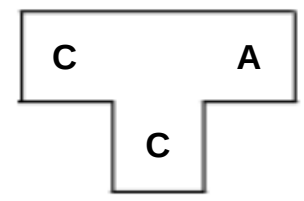
因为在对 LR(1) 合并同心集合时， 有可能将原本没有冲突的同心集的项目集合并后造成一些归约项目向前搜索符集合的交集不是空，产生归约 -归约冲突。但是由于文法本身已经是 LR(1) 文法， 因此可知， 在项目集中一定不存在移进 -归约冲突， 也就是移进项目要求输入的终结符和任意归约项目的向前搜索符集合的交集都是空集。这样，在将同心集合并之后，移进项目要求输入的终结符和归约项目的向前搜索符集合的交集也还是空集。

2. 如果在 A 机器上我们有 C 语言编译器 CCA，也有它的源码 SA（用 C 语言写成）。如何利用它通过尽量少的工作来得到 B 机器的 C 语言编译器 CCB。（5 分）

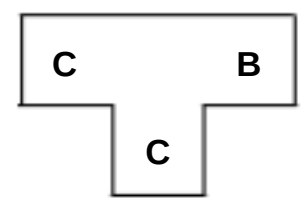
答： A 机器上 C 语言编译器 CCA 的结构如下：



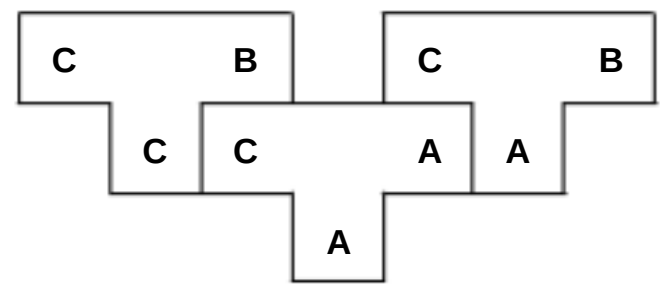
其源码 SA 结构如下：



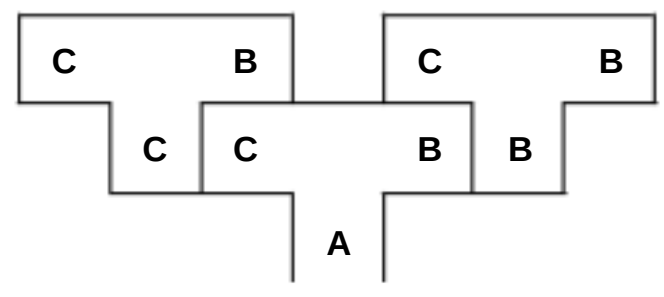
首先，用 C 语言编写一个从 C 语言到 B 机器语言的编译器，成为 SB，其结构如下：



第二步， 将这个编译器放到 CCA 中进行编译， 得到用 A 机器语言编写的， 将 C 语言编译成 B 机器代码的编译器，其过程和结构如下：



第三步， 再将 SB 放入新得到的这个编译器中去编译， 就得到了要求的编译器 CCB，其过程和结构如下：



3. Pascal 语言允许过程嵌套声明， C 语言的过程声明不能嵌套。在 Pascal 程序中，数据分为局部数据、非局部数据，而 C 程序中，数据分为局部数据和全局数据。因此， C 程序执行时只用到了控制链 (动态链)，不需要使用访问链 (静态链)。试根据前面的已知说明，为什么 Pascal 程序执行时需要使用访问链，而 c 程序不需要。 (5 分)

答：由于 C 语言不允许嵌套的过程声明，因此所有的非局部名字都可以静态地绑定到所分配的存储单元，因此，可以不使用访问链。而 Pascal 语言允许过程的嵌套，并使用静态作用域，确定用于名字的声明需要根据过程的嵌套层次来决定。和 C 语言不同的是， Pascal 语言的非局部名字不一定就是全局的。 运行时访问非局部名字的时候，我们首先要确定该非局部名字被绑定到的活动记录，因此就必须要用到访问链。

二、简单计算题（共 25 分）

1. 令文法 G[E] 为

E→T | E+T | E-T

T→F | T*F | T/F

F→(E) | i

(1) 证明 E+T*F 是它的一个句型，指出这个句型的所有短语、直接短语和句柄。

(2) 给出 i+i*i 、 i*(i+i) 的最左推导和最右推导； (10 分)

答： (1)E=>E+T=>E+T*F ，其短语有： T*F， E+T*F ；直接短语是 T*F，句柄是 T*F

(2) i+i*i 的最左推导：

E =>E+T
=>T+T
=>F+T
=>i+T
=>i+T*F
=>i+F*F
=>i+i*F
=>i+i*i

最右推导

E =>E+T
=>E+T*F
=>E+T*i
=>E+F*i
=>E+i*i
=>T+i*i
=>F+i*i
=>i+i*i

i*(i+i) 的最左推导：

E =>T
=>T*F
=>F*F
=>i*F
=>i*(E)
=>i*(E+T)
=>i*(T+T)
=>i*(F+T)
=>i*(i+T)
=>i*(i+F)
=>i*(i+i)

最右推导

E =>T
=>T*F
=>T*(E)
=>T*(E+T)
=>T*(E+F)
=>T*(E+i)
=>T*(T+i)
=>T*(F+i)
=>T*(i+i)
=>F*(i+i)
=>i*(i+i)

2. 已知语言写出相应的文法： （6 分）

(1) 已知语言 L={WaW ^{r} | W 属于 (0 | a)*，W^{r} 表示 W 的逆 }，试构造相应的上下文无关文法。

(2) 已知语言 L={1 ^{n}0^{m}1^{m}0^{n} | m>0, n>=0} ， 试构造相应的上下文无关文法。

(3) 已知语言 L={ a ^{n}b^{n}a^{m}b^{m} | m>=0, n>0} ， 试构造相应的上下文无关文法

答：

(1) 文法为： ({S},{0,a},P,S) ， 其中 P 为 S→0S0 | aSa | a

(2)文法为： ({A,S},{0,1},P,S) ， 其中 P 为

S→1S0 | A

A →0A1 | 01

(2)文法为： ({A,B,S},{a,b},P,S) ， 其中 P 为

S→AB

A →aAb | ab

B→aBb | ε

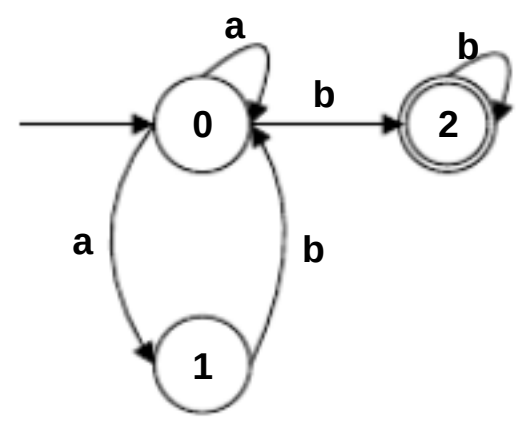
3. 构造一个 NFA ,

(1) 接受字母表 {a,b} 上的正规式 (ab|a)*b+ 描述的集合。

(2) 将(1)中的 NFA 转换为等价的 DFA

(3) 将(2)中的 DFA 转换为最小状态 DFA（写出步骤）（9 分）

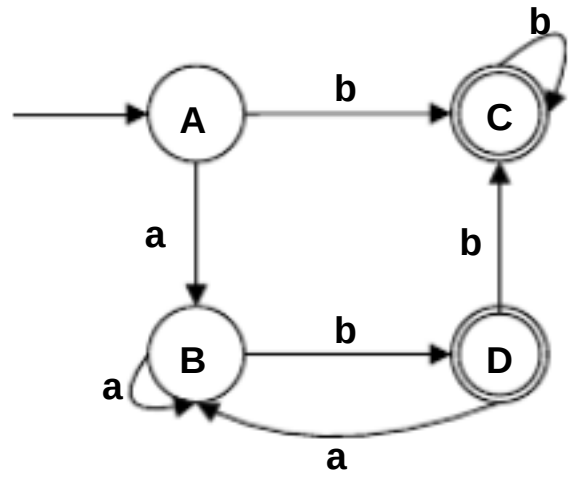
答：构造的 NFA 如下：



确定化过程：

状态集合	a	b
{0} A	{0,1} B	{2} C
{0,1} B	{0,1} B	{0,2} D
{2} C		{2} C
{0,2} D	{0,1} B	{2} C

确定的 DFA 如下：



上面的 DFA 已经是最小化的。

三、证明推导题（共 30 分，每题 10 分）

2 下列文法由开始符号　S 产生一个二进制数，令综合属性　val 给出该数的值：

S→L.L | L

L→LB | B

B→0 | 1

试设计求　S.val 的属性文法，　其中，　已知 B 的综合属性　c，给出由　B 产生的二进位的结果值。

答：

产生式	语义规则
S　L1.L2	S.val:=L1.val+L2.val/2$^{L2.length}$
S　L	S.val:=L.val
L　L1B	L.val:=L1.val*2+B.val L.length:=L1.length+1
L　B	L.val:=B.val L.length:=1
B　0	B.val:=0
B　1	B.val:=1

四、程序计算 /设计题（共 30 分，每题 10 分）

1. 设 A 为一个 10×20 的数组，即 n1=10，n2=20，并设 w=4。
试将赋值语句 x:=A[y,z] 翻译成三地址语句序列。

答： T_1 := y*20

T_1 := T_1 + z

T_2 := A - 84

T_3 := 4 * T_1

T_4 := T_2[T_3]

x := T_4

2. 如果不存在形如 S=> ωX γ=> ωx γ的推导，则文法符号 X 是无用的，即从开始符号 S，不能够推导出含有 X 的句型，也就是说 X 不会出现在任何句子的推导中。试设计一个算法，从文法中删除包含无用符号的产生式。

答：设计一个队列，用来存放可以出现在句子推导中的非终结符号。

对队列进行初始化，将 S 放在其中。

(1) 从队列中取出一个符号，设其为 A，考察产生式，将形如 A→ α的产生式右部符号串中出现的非终结符都放入队列中。如果非终结符已经存在队列中，则不重复加入。

(2) 重复上述动作，直至队列中所有的非终结符都考察过，并且没有新的非终结符加入队列为止。

此时得到的队列就是那些有用的非终结符。那些含有不在此队列中出现的非终结符的产生式就是包含无用符号的产生式，将其删除即可。

3. 考虑下面程序

```
void Q(int x)
{
    int i=1;
    x = x+2;
    i = 2;
    x = x+2;
}
void main()
{
    int i;
    int B[3];
B[1]=1;
    B[2]=2;
    i = 1;

    Q(B[i]);
    }
```

试问：若参数传递方式分别采取 (1)传值调用，(2)引用调用，(3) 复制 -恢复调用，(4)传名调用时，程序执行后输出 B[1] 和 B[2] 的值分别是什么？请简要写出计算过程。

答：

(1) 采用传值调用时， 将实在参数的值传递给形式参数， 而后在函数调用过程中， 操作的是形式参数，形式参数的值发生改变，而且这些改变不能重新传递给实在参数，
所以得到的结果是 B[1]=1 ；B[2]=2

(2) 采用应用调用， 将实在参数的地址传递给形式参数， 此时对形式参数的操作就相当于对其指向的地址单元进行操作，其操作影响了实在参数，所以得到的结果是
B[1]=5 ；B[2]=2

(3) 采用复制 -恢复调用，首先将实在参数的值传递给形式参数，此时， x=1， y=2，进行函数调用后，得到， x=5， y=2，调用返回时，将形式参数的值传递到相应的实在参数的地址中， 即 x 的值传递到 B[1] 的地址中，所以得到的结果是 B[1]=5 ;B[2]=2

(4) 采用传名调用，将 B[i] 当成整个的一个整体，替换函数调用中的 x，得到：

```
i=1;
B[i] = B[i]+2;
i = 2;
B[i] = B[i]+2;
```

计算得到， B[1]=3 ， B[2]=4