Compiler

2019 Fall Final Examination Solution

Name_____ Student No._____ Score_____

Problem 1: Type Checking (16 points)

注：A 卷和 B 卷的 Type Checking 题目只是调换了题目顺序，题目内容是一样的。

1.

E0 = {n->int}

E1 = E0 + {output->string, t1->number, t2->number, nextTerm->number}

E2 = E1 + {tmp->int}

E3 = E1 + {i->int}

E4 = E3 + {output->int}

注：

①只需要写新创建的环境，恢复的环境不需要写。

②上面使用 **number** 的类型可以用 **int** 来表示，但是反过来不可以，使用 ***int*** 的类型不能用 **number** 来表示。

③**environment** 既可以只写每次变化的部分，也可以将其中所有的绑定写出来。

2.

Line 1: E0

Line 3-8: E1

Line 14-17: E2

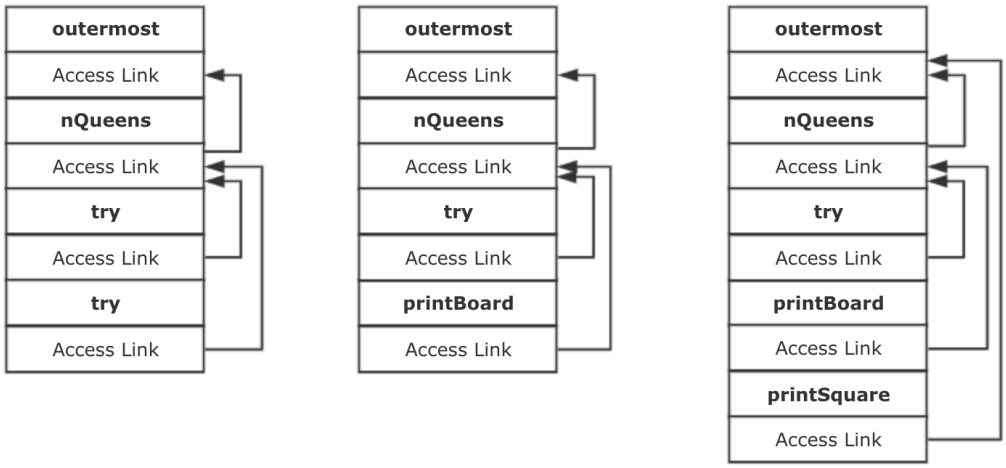
Line 31: E3

Line 33: E1

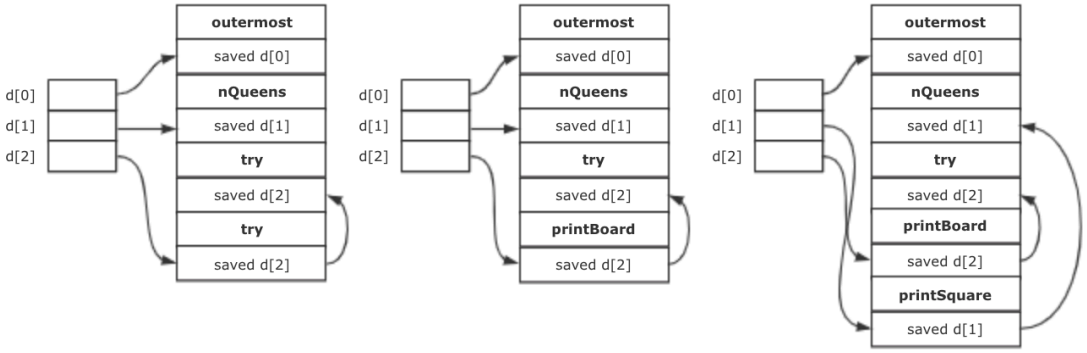
Problem 2: Static Link and Escape Calculation (30 points)

注：A 卷和 B 卷的 Static Link and Escape Calculation 题目只是调换了题目顺序，题目内容是一样的。

1.



2.



3.

Variable	Escape(Y/N)	Your Reason
maxN (Line 4)	N	没有被嵌套的函数访问，而且是简单的数据类型，能放入寄存器中
emptySymbol(Line 5)	Y	字符串变量是指向字符串的指针，虽然可以存放寄存器中，但被 printSquare 函数访问
queenSymbol(Line 9)	N	字符串变量是指向字符串的指针，只被当前函数所访问
col(Line 17)	Y	col 被嵌套的 try 函数访问，虽然 col 是一个数组类型，但是还是其数据是放在堆上的，col 也只是一个指针

Problem 3: Garbage Collection (20 points)

注：A 卷和 B 卷的 Garbage Collection 题目内容是一样的。

1.

优点：减少内存碎片化，操作简单复杂性低。

缺点：如果大部分变量存活时间很长，会导致内存拷贝过多，overhead 很大，而且复制回收只能用一半内存空间。

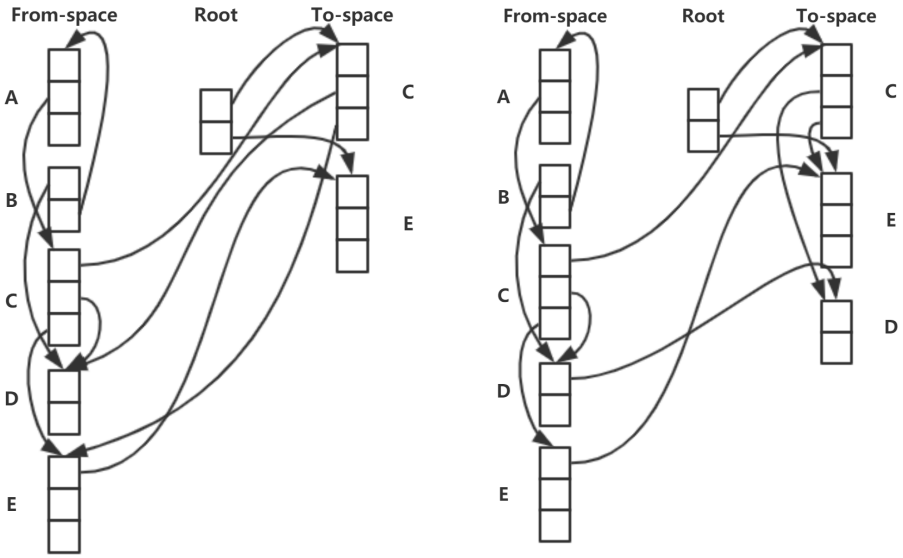
2.

对溢出变量染色，进行内存分配，这样可以把活跃范围不重叠的溢出变量分配到同一个内存地址处。

3.

会，活性分析是编译时的工作，而垃圾回收是运行时做的，它无法获取到变量的活跃信息。

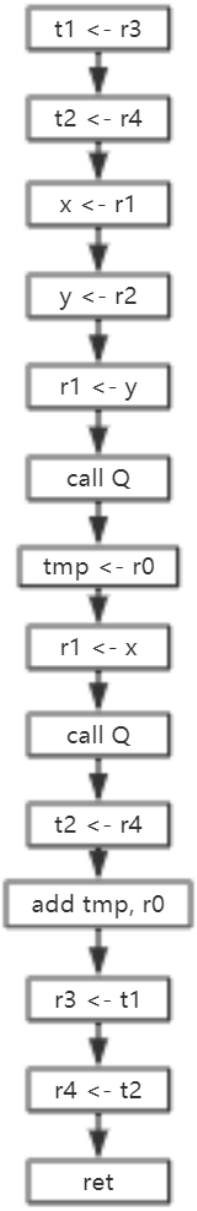
4.



Problem 4: Register Allocation (34 points)

注：B 卷相比 A 卷就是将 r0->r5， r1,r2 与 r3,r4 互换。

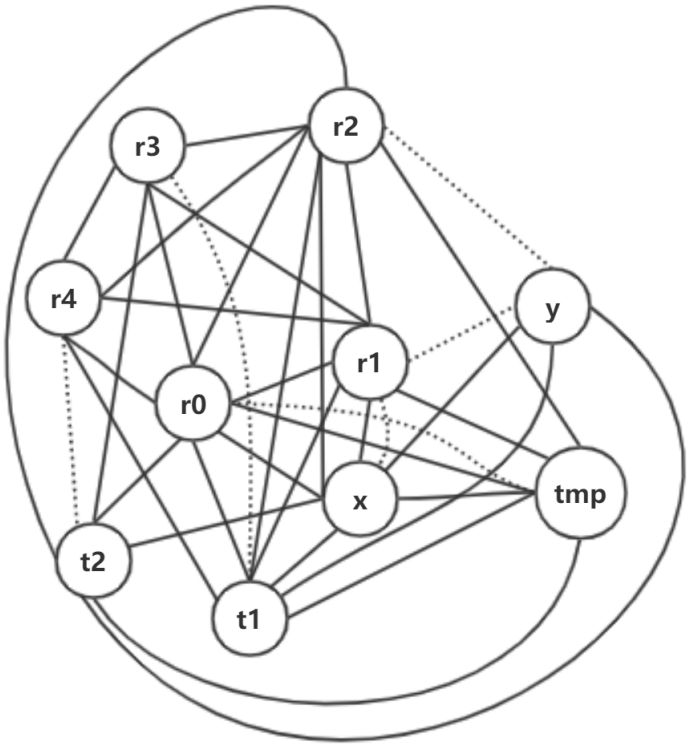
1.



2.

instr	**def**	**use**	**in**	**out**
t1 <- r3	t1	r3	r1,r2,r3,r4	r1,r2,r4,t1
t2 <- r4	t2	r4	r1,r2,r4,t1	r1,r2,t1,t2
x <- r1	x	r1	r1,r2,t1,t2	r2,x,t1,t2
y <- r2	y	r2	r2,x,t1,t2	x,y,t1,t2
r1 <- y	r1	y	x,y,t1,t2	r1,x,t1,t2
call Q	r0,r1,r2	r1	r1,x,t1,t2	r0,x,t1,t2
tmp <- r0	tmp	r0	r0,x,t1,t2	tmp,x,t1,t2
r1 <- x	r1	x	tmp,x,t1,t2	tmp,r1,t1,t2
call Q	r0,r1,r2	r1	tmp,r1,t1,t2	tmp,r0,t1,t2
add tmp, r0	r0	tmp,r0	tmp,r0,t1,t2	r0,t1,t2
r3 <- t1	r3	t1	r0,t1,t2	r0,r3,t2
r4 <- t2	r4	t2	r0,r3,t2	r0,r3,r4
ret			r0,r3,r4	**r0,r3,r4**

3.



4.

合并 **y，r2（George）**

冻结 **r0,r1,r3,r4**

溢出 **t1**

溢出 **t2**

简化 **x**

简化 **tmp**

弹出栈分配颜色，**t1,t2** 成为实际溢出

(下面步骤不生成 t3， t4 两个新的临时变量，直接将 t1,t2 替换成内存也算正确)

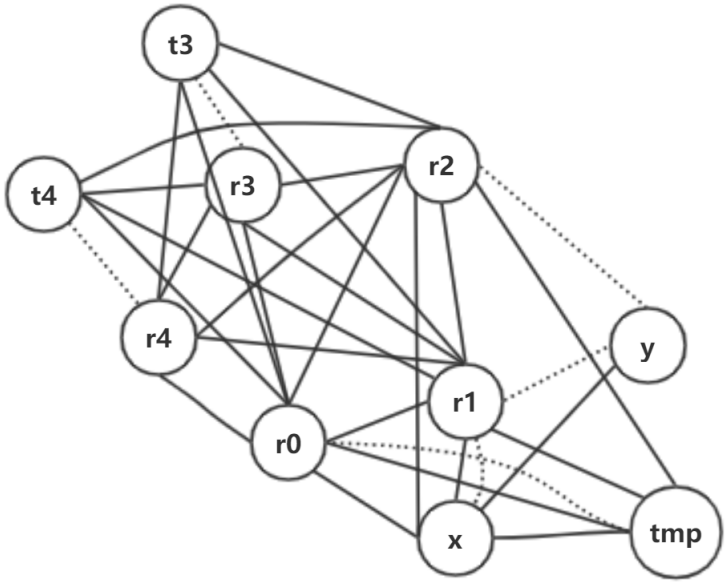
分配栈空间，重写指令：

t3 <- r3
M[t1] <- t3
t4 <- r4
M[t2] <- t4
x <- r1
y <- r2
r1 <- y
call Q
tmp <- r0
r1 <- x
call Q
add tmp, r0
t3 <- M[t1]
r3 <- t3
t4 <- M[t2]
r4 <- t4
ret

instr	def	use	in	out
t3 <- r3	t3	r3	r1,r2,r3,r4	r1,r2,r4,t3
M[t1] <- t3		t3	r1,r2,r4,t3	r1,r2,r4
t4 <- r4	t4	r4	r1,r2,r4	r1,r2,t4
M[t2] <- t4		t4	r1,r2,t4	r1,r2
x <- r1	x	r1	r1,r2	r2,x
y <- r2	y	r2	r2,x	x,y
r1 <- y	r1	y	x,y	r1,x
call Q	r0,r1,r2	r1	r1,x	r0,x
tmp <- r0	tmp	r0	r0,x	tmp,x
r1 <- x	r1	x	tmp,x	tmp,r1
call Q	r0,r1,r2	r1	tmp,r1	tmp,r0

add tmp, r0	r0	tmp,r0	tmp,r0	r0
t3 <- M[t1]	t3		r0	r0,t3
r3 <- t3	r3	t3	r0,t3	r0,r3
t4 <- M[t2]	t4		r0,r3	r0,r3,t4
r4 <- t4	r4	t4	r0,r3,t4	r0,r3,r4
ret			r0,r3,r4	**r0,r3,r4**

重新构造冲突图如下：



合并 t3，r3（George）

合并 t4，r4（George）

合并 y，r2（George）

所有传送指令变为受抑制的，可以消除

简化 tmp

简化 x

分配 r3 给 tmp，r4 给 x

染色成功，重写指令并消除多余传送指令，结果如下：

M[t1] <- r3
M[t2] <- r4
r4 <- r1
r1 <- r2
call Q
r3 <- r0
r1 <- r4

```
call Q
add r3, r0
r3 <- M[t1]
r4 <- M[t2]
ret
```