

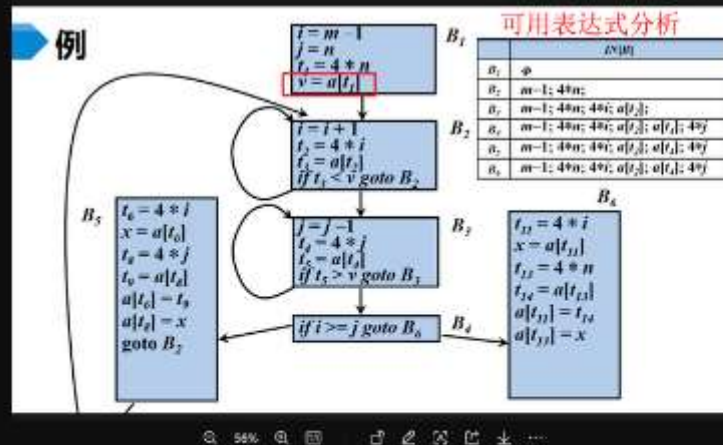
2024/05/31 20:17

TSUNDOKU



为啥可用表达式没a[t1]

TSUNDOKU



因为b5会重新赋值



就像DAG画图里面，有赋值，所有和a相关的都kill了



但是at2是在b5后面的



所以可以用

2024/05/31 20:25

TSUNDOKU



哦，对，看a

TSUNDOKU



不该看t1

2024/05/31 20:35

TSUNDOKU



所以算kill的时候, 发现对数组定值, 是不是要kill掉对这个数组所有引用



是的



cy 的慕课强调了这一个

TSUNDOKU



okk

2024/05/31 14:11

那如果要求画图的话S1.next要指向哪里啊

语句 “while  $a < b$  do if  $c < d$  then  $x = y + z$  else  $x = y - z$ ” 的三地址代码

$B_{first} \rightarrow$ 1: if $a < b$ goto 3	1: if <b>False</b> $a < b$ goto 11	避免生成冗余的 goto 指令
2: goto 11	2:	
$S_{first} \rightarrow$ 3: if $c < d$ goto 5	3: if $c < d$ goto 5	
4: goto 8	4: goto 8	
5: $t_1 = y + z$	5: $t_1 = y + z$	
6: $x = t_1$	6: $x = t_1$	
7: goto 1	7: goto 1	
8: $t_2 = y - z$	8: $t_2 = y - z$	
9: $x = t_2$	9: $x = t_2$	
10: goto 1	10: goto 1	
11:	11:	

```
graph TD
    B_begin[B.begin] --> B_true[B.true]
    B_true --> S1_next[S1.next]
    S1_next --> S1_code[S1.code]
    S1_code --> S1_next
    S1_next --> B_false[B.false]
    B_false --> B_begin
```

如果不优化的话

TSUNDOKU

这不写着吗

TSUNDOKU

B.begin

TSUNDOKU

只是改了SDT

那这感觉是优化了呀

不优化的话S1.next不应该指向goto吗

TSUNDOKU

那个goto语句是SDT生成的

TSUNDOKU



S1.next指向B.begin



嘶

TSUNDOKU



不是顺序执行下面的代码

TSUNDOKU



往上走

TSUNDOKU



故生成goto语句

TSUNDOKU



跳到上面代码



嘶我把这个例子想错了



我的本意是



如果S1中有一个if else的话



这个if else 内部的goto不是指向goto B.begin,而是指向B.begin



这个不属于goto优化对吗

TSUNDOKU



```

1: if a < c goto 3
2: goto 16
3: if b < d goto 5
4: goto 16
5: if a = 1 goto 7
6: goto 10
7: tj = c + 1
8: c := tj
9: goto 1
10: if a <= d goto 12
11: goto 1
12: tj = a + 2
13: a := tj
14: goto 10
15: goto 1
16:

```

```

1: (j := a, c, 3)
2: (j, ~, 16)
3: (j := b, d, 5)
4: (j, ~, 16)
5: (j := a, 1, 7)
6: (j, ~, 10)
7: (+, c, 1, tj)
8: (=, tj, c)
9: (j, ~, 1)
10: (j := a, d, 12)
11: (j, ~, 1)
12: (+, a, 2, tj)
13: (=, tj, a)
14: (j, ~, 10)
15: (j := 1)
16:

```

```

while a < c & b < d do
  if a = 1
    then c := c + 1
  else
    while a <= d do
      a := a + 2

```

TSUNDOKU

这个?

啊差不多

9 行就是goto 1

而不是goto 15

2024/05/31 14:21

TSUNDOKU

我想想

2024/05/31 14:31

TSUNDOKU

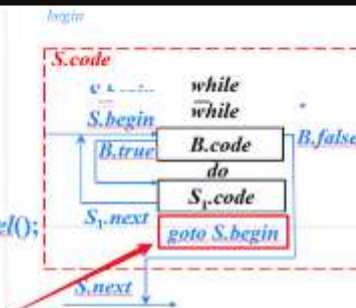
### while-do语句的SDT

$S \rightarrow \text{while } B \text{ do } S_1$

```

S → while { S.begin = newlabel();
label(S.begin);
B.true = newlabel();
B.false = S.next; } B
do { label(B.true); S1.next = S.begin; } S1
{ gen('goto' S.begin); }

```

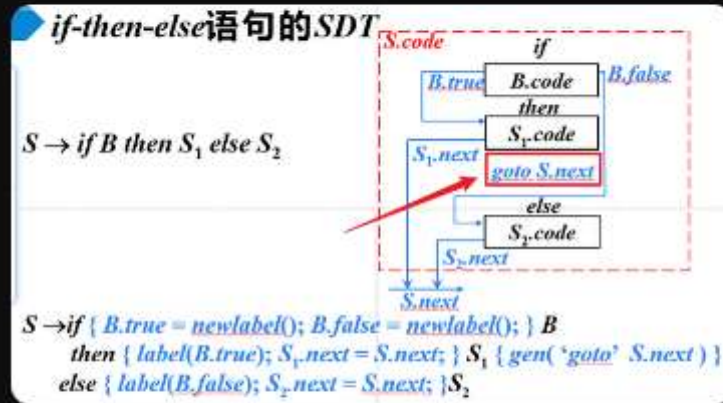


TSUNDOKU

这个对应15行代码

对

TSUNDOKU



TSUNDOKU

这个对应第9行

是的

这里S.next被设置成了while中的S.begin

TSUNDOKU

TSUNDOKU



这个图里的S.next相当于第一个图里的S1.next

所以里面第九行要直接指向1

TSUNDOKU



对

TSUNDOKU

他两相等

TSUNDOKU

都是一个值

所以写goto15是错的罢

我今天才发现问题

这个避免冗余goto是一定要做的对吧

2024/05/31 14:36

TSUNDOKU

正常来说应该按照SDT一步步搞

TSUNDOKU

有问题的其实都是人肉编译

TSUNDOKU

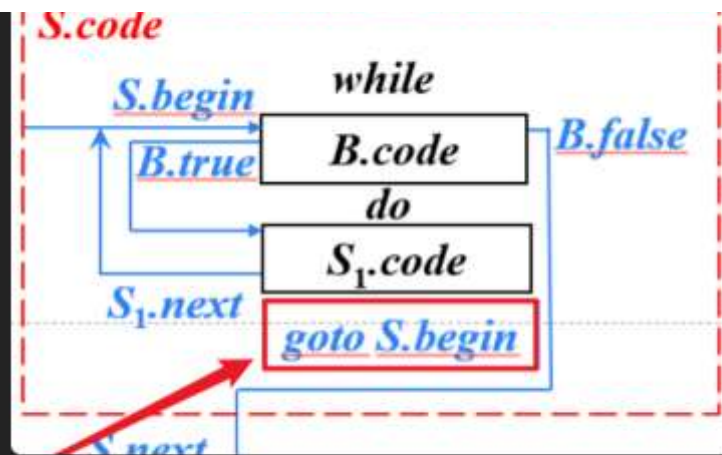
人肉编译尽量画个图

TSUNDOKU

知道goto到哪

如果单纯if while的话感觉可以直接人肉

TSUNDOKU



冗余的goto其实就相当于S1.next指向goto语句

是的

现在想想，这个似乎也不叫冗余goto吧

TSUNDOKU

对

比较goto S.begin还是会用到

TSUNDOKU

就是跳转的问题

TSUNDOKU

跳转的指令序号搞错了



2024/03/29 22:25

TSUNDOKU



老师，实验的选做可以不做吗

2024/03/29 22:56



Shelly Shan

可以不做。

TSUNDOKU



不做不会影响分数吧



Shelly Shan

不影响

2024/05/20 21:27

TSUNDOKU



➤ 试将下面的语句翻译成四元式序列  
(2) for  $i:=m$  step 2 until  $n$  do  
    if  $a<b$  then  $x:=x+1$

TSUNDOKU



➤ 在SPOC讲义6.3节（控制语句的翻译）所示的SDT中添加处理下列控制流构造的翻译方案  
➤ (1) repeat语句:  $S \rightarrow \text{repeat } S_1 \text{ while } B$   
➤ (2) for循环语句:  $S \rightarrow \text{for } (S_1; B; S_2) S_3$

TSUNDOKU



老师，这种题用掌握吗

TSUNDOKU



这种语句上课好像没讲过



Shelly Shan

讲的是方法，要求会运用解决问题

2024/05/20 21:35

TSUNDOKU



那这种也得掌握是吗

2024/05/21 09:02

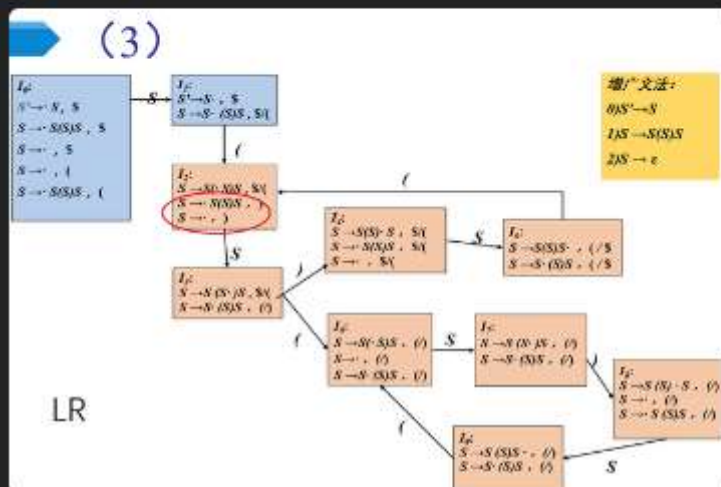


Shelly Shan

是的

2024/05/26 13:26

TSUNDOKU



老师，这展望符是不是少了左括号

2024/05/26 13:32



Shelly Shan

是的

TSUNDOKU



这是一个习题集，学长写的答案，那个I7到I8，是不是也是少分析了左括号的情况

2024/05/26 13:39



Shelly Shan

17是, 现在的18没有吧?

TSUNDOKU



是的

2024/05/27 13:58

TSUNDOKU



$S \rightarrow a \mid S + S \mid S S \mid S * \mid (S)$

$S \rightarrow a S'$

$S \rightarrow (S) S'$

$S' \rightarrow + S S'$

$S' \rightarrow S S'$

$S' \rightarrow * S'$

$S' \rightarrow \varepsilon$

TSUNDOKU



老师, 这样改写对吗

2024/05/27 14:37



Shelly Shan

对的

2024/05/27 16:33

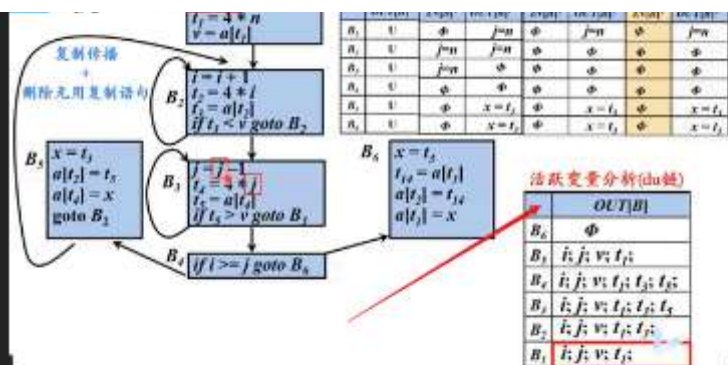
TSUNDOKU



例

$B, \begin{matrix} i = m - 1 \\ j = n \end{matrix}$

可用复制语句分析



老师，这块指的是出口的活跃变量吗

TSUNDOKU

如果是，为啥会有v啊

TSUNDOKU

v没被引用过啊

TSUNDOKU

还是说i判断用了v也算



Shelly Shan

引用就算

TSUNDOKU

所以说i判断用了v也算是吗



Shelly Shan

对的

2024/05/31 11:45

TSUNDOKU

## 习题7.4

说明下面的文法

$S \rightarrow SA \mid A$

$A \rightarrow a$

是SLR(1)的，但不是LL(1)的

TSUNDOKU

老师, 判断是不是LL (1) 文法

TSUNDOKU

是不是先消除左递归再判断



Shelly Shan

根据题意吧, 此题应该不改造。

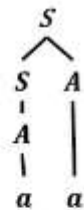
TSUNDOKU

直接找S的1, 2产生式的select集判断, 是吗

TSUNDOKU

$S \rightarrow SA|A$   
 $A \rightarrow a$

对于产生式 $S \rightarrow SA$ 和 $S \rightarrow A$ , 当选择终结符 $a$ 时, 二者都可以推出以 $a$ 开头的串



这是之前学长写的

2024/05/31 12:30

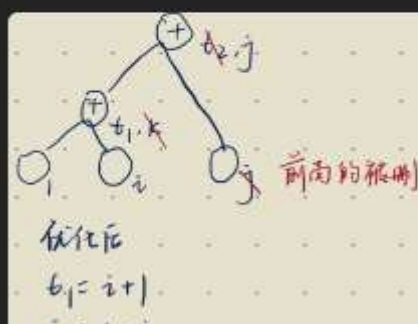


Shelly Shan

是的

2024/05/31 18:35

TSUNDOKU



$B_j$

```
t1=i+1;
k=t1;
t2=j+k;
j=t2;
```



老师, 这里*i*,*j*是活跃变量

TSUNDOKU

那个*t1*和*k*, 我保留*t1*还是*k*有区别吗

TSUNDOKU

$k = i + 1$

$j = j + k$

TSUNDOKU

像这个和图片里的

2024/05/31 21:25

TSUNDOKU

⑤  $T \rightarrow \uparrow T_1 \{ T.type = pointer(T_1.type); T.width = 4; \}$   
⑥  $B \rightarrow int \{ B.type = int; B.width = 4; \}$   
⑦  $B \rightarrow real \{ B.type = real; B.width = 8; \}$   
⑧  $C \rightarrow c \{ C.type = c; C.width = 4; \}$

TSUNDOKU

老师, *real*算8个字节吗

2024/05/31 21:42



Shelly Shan

没有区别



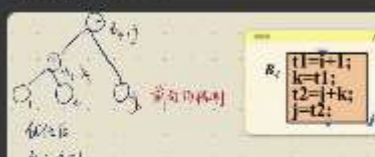
Shelly Shan

Shelly Shan

是的

TSUNDOKU

TSUNDOKU





那这个题呢

TSUNDOKU

t1和k, 我保留t1还是k有区别吗

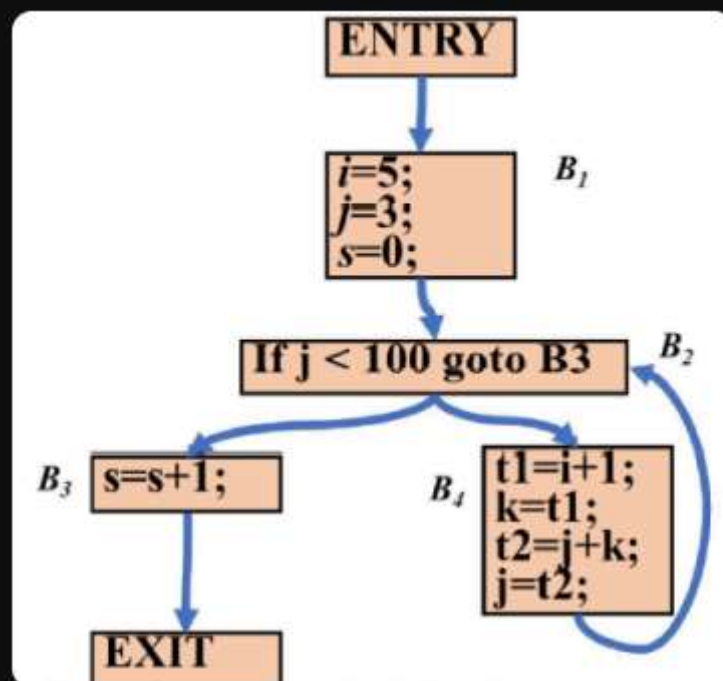
TSUNDOKU

Shelly Shan  
没有区别

哦, 看到了

2024/06/01 08:18

TSUNDOKU



TSUNDOKU

4. 找出所有的循环不变计算; 这些循环不变计算能不能提出循

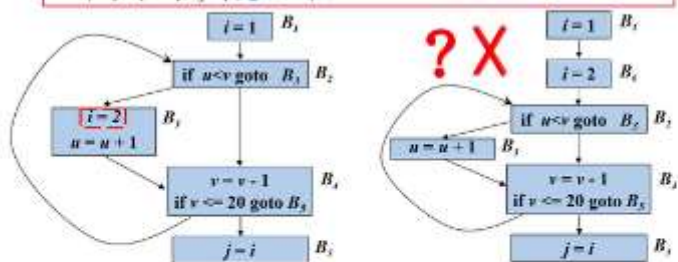
TSUNDOKU

老师, 2, 4构成的循环中, 循环不变计算能外提吗

TSUNDOKU

循环不变计算语句  $s : x = y + z$  移动的条件

(1)  $s$  所在的基本块是循环所有出口结点(有后继结点在循环外的结点)的支配结点



TSUNDOKU

感觉这个条件不满足

2024/06/01 10:17



Shelly Shan

不能外提

TSUNDOKU

3 以下说法不正确的是( )。

- ☐ A. 对于声明语句: 变量名和变量名列表后跟以冒号分隔的基型符, 为每一个变量分配一个存储地址
- ☐ B. 从变量状态可以知道该变量在运行时的函数存储位置。在编译时, 可以使用这些数据为每一个名字分配一个存储地址
- ☒ C. 名字和地址和相对地址都是存在于符号表中
- ☐ D. 对声明的变量进行声明, 但不产生中间代码

TSUNDOKU

那老师

TSUNDOKU

这个为啥选D啊

TSUNDOKU

感觉声明没有生成中间代码



Shelly Shan

错在构造二字

TSUNDOKU

那该是啥, 存入符号表?

2024/06/01 10:26



Shelly Shan

不总是构造



Shelly Shan

大多数时候只是添加条目

2024/06/01 12:42

TSUNDOKU



老师, a=1会不会跟e\_kill有关, 导致e\_kill添加所有和a相关的表达式

2024/06/01 15:38



Shelly Shan

当然

TSUNDOKU



```

S → { a1: L.side = left, }
LS' { a2: S.val = L.val + S'.val; }
S' → { a3: L.side = right, }
L { a4: S'.val = L.val; }
S' → ε { a5: S'.val = 0; }
L → B { a6: L'.side = L.side, }
L' { a7: if L.side = left then L.val = B.val * 2 ** length; L'.val;
      if L.side = right then L.val = (B.val + L'.val) / 2;
      L.length = L'.length + 1; }
L' → B { a8: L'.side = L'.side; }
L' { a9: if L'.side = left then L'.val = B.val * 2 ** length; L'.val;
      if L'.side = right then L'.val = (B.val + L'.val) / 2;
      L'.length = L'.length + 1; }
L' → ε { a10: L'.val = 0; L'.length = 0; }
B → 0 { a11: B.val = 0; }
B → 1 { a12: B.val = 1; }

```

(2) 为了让(1)中SDT能在自顶向下分析中实现, 请写出产生式 $L \rightarrow BL'$ 入栈时, 与右部各记录相关联的语义动作的栈操作。(参考第5章PPT73-76页)。

答:

```

S → { a1: L.side = left, }
LS' { a2: S.val = L.val + S'.val; }

```

符号	属性	执行代码	动作
a1		stack[top-1].side = left; top=top-1;	
L	side	右部: L->BL' 推导 stack[top+1].side = stack[top].side; stack[top].L.side = stack[top].side; top=top+2;	复制 到 (a6)和(a7)
Lsyn	val, length	stack[top-3].L.val = stack[top].val; top=top-1;	复制 到 (a2)
S'			
S'syn	val	stack[top-1].S'.val = stack[top].val; top=top-1;	复制 到 (a2)
a2	L.val, S'.val	stack[top-1].val = stack[top].L.val + stack[top].S'.val; top=top-1;	

TSUNDOKU



s'那栏用不用, 考虑top的变化

2024/06/01 15:44



Shelly Shan

不用

TSUNDOKU



多选题 1分

在预分配标号的SDT中，关于B.true 和B.false的叙述正确的是：

- ☐ A 都是继承属性，记录跳转指令目标标号
- ☐ B 在生成跳转指令时，目标标号已经确定
- ☐ C 在生成跳转指令时，其值不能确定
- ☐ D 都是综合属性，记录跳转指令目标标号，在翻译B后才能确定。

提交答案: AC

得分: 0/1 用时: 57秒

正确答案: AB

TSUNDOKU



那老师

TSUNDOKU



这个C问题在哪呢

2024/06/01 15:50



Shelly Shan

B和C只能有一个对吧?

TSUNDOKU



嗯



Shelly Shan

预分配跳转指令的标号，在生成跳转指令时，当然就有标号了



Shelly Shan

否则，预分配标号就没有意义了

TSUNDOKU



预分配跳转指令的标号，指的是L1这种标号吗



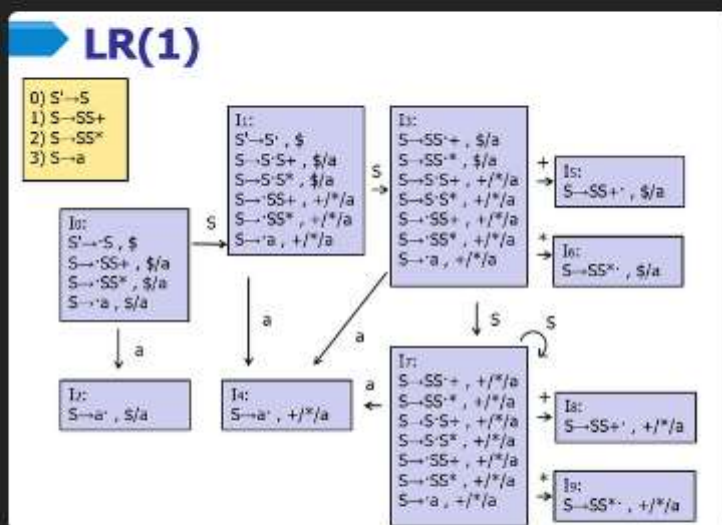
Shelly Shan

相反回填的方法中，在生成跳转指令时，跳转指令标号是未知的，不填充



Shelly Shan

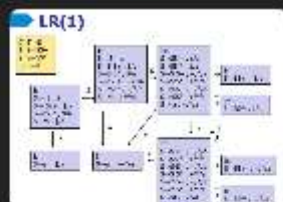
是的



2024/05/23 13:34



## 2. 闭包运算中新加入项的展望符 $FIRST(\beta a)$

 $[A \rightarrow \alpha \cdot B \beta, a]$ 
$$B \rightarrow \gamma \in P$$
$$[B \rightarrow \cdot \gamma, b]$$


图用的hei ta应该且空吧



图里的beta应该是主吧

不应该是直接继承吗

求 $S \rightarrow SS+$  的等价项目的时候

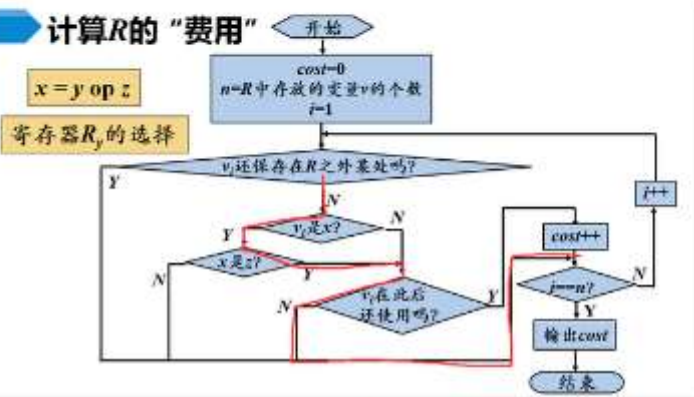
beita是 $S+$

不应该是直接继承吗

哦我懂了，完事还得再求一次等价项

2024/05/24 22:40

### 计算 $R$ 的“费用”



这条路想不明白，为啥这条路的条件下 $v_i$ 不需要被保存

$R$ 唯一保存了 $v_i$ ， $v_i$ 的值是 $x$ 以及 $z$ ，那计算 $op$ 的时候要  
用到 $z$ 啊，不保存 $v_i$ ， $R$ 的值把 $v_i$ 改了，用到 $z$ 的时候找不  
到了啊

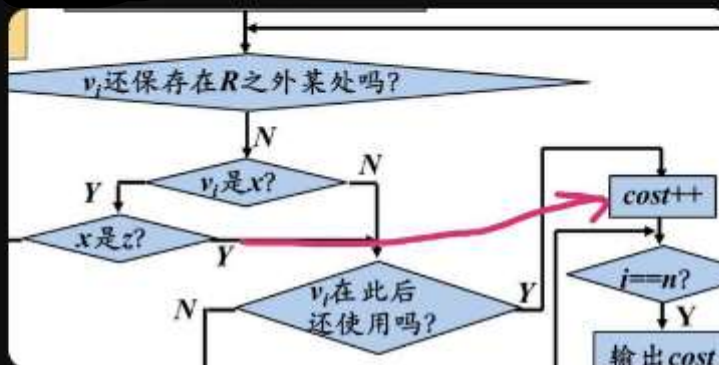
2024/05/25 06:40

R唯一保存了 $v_i$ ,  $v_i$ 的值是 $x$ 以及 $z$ , 那计算 $op$ 的时候要用到...

确实, 你说的对, 这个图是有些小问题, 我理解的是当 $v_i$ 的值是 $x$ 以及 $x$ 就是 $z$ 的时候, 直接 $cost++$ 了, 因为 $v_i$ 的值紧接着就会被用到

所以这条路径下这个判断是冗余的

流程图应该修改为:



(我课后就这个小点向陈老师问了一下, 她也表示认同🐼)

2024/05/25 18:09

想问一下,  $t1 = y + 1$

param t1

t2 = call f, 1

t3 = t2 + 2

x = t3

三地址码关于  $x = f(y + 1) + 2$  的翻译中

t2=call f,1中的1是啥意思

参数个数

2024/05/28 17:24

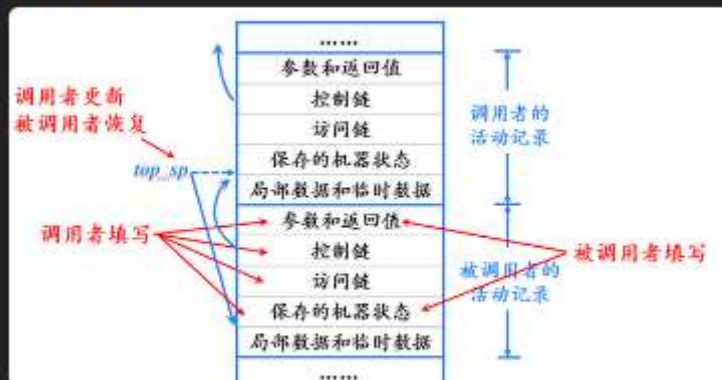
问一下，访问链是由被调用者填写还是由调用者填写

2024/05/28 18:40

TSUNDOKU

问一下，访问链是由被调用者填写还是由调用者填写

@



cyMOOC的课件

TSUNDOKU

### 调用者和被调用者之间的任务划分



但sll的没写

TSUNDOKU

## 访问链的建立

- 建立访问链的代码属于调用序列的一部分
- 假设嵌套深度为 $n_x$ 的过程 $x$ 调用嵌套深度为 $n_y$ 的过程 $y$  ( $x \rightarrow y$ )
  - $n_x < n_y$  的情况 (外层调用内层)
    - $y$  一定是在 $x$ 中定义的 (例如:  $s \rightarrow q, q \rightarrow p$ ) , 因此,  $n_y = n_x + 1$
    - 在调用代码序列中增加一个步骤: 在 $y$ 的访问链中放置一个指向 $x$ 的活动记录的指针



过程	嵌套深度
sort	1
readarray	2
exchange	2
quicksort	2
partition	3

不过sll课件后面提到了这句

2024/05/28 21:53

## 习题19.1

- 对于下图中的流图:
  - (1) 找出流图中的循环。
  - (2) B1中的语句 (1) 和 (2) 都是复制语句。其中 $a$ 和 $b$ 都被赋予了常量值。我们可以对 $a$ 和 $b$ 的哪些使用进行复制传播, 并把对它们的使用替换为对一个常量的使用? 在所有可能的地方进行这种替换。
  - (3) 对每个循环, 找出所有的全局公共子表达式。
  - (4) 寻找每个循环中的归纳变量。同时要考虑在 (2) 中引入的所有常量。
  - (5) 寻找每个循环的全部循环不变计算。

对于循环的公共子表达式应该怎么定义? 感觉按照公共子表达式的定义似乎是具体到出现的位置?

2024/05/28 22:04

局部的公共子表达式在基本块内就可以解决, 全局的公共子表达式可以用可用表达式分析, 以基本块为单位, 这时候其实跟循环没关系了, 但循环相关的一些“约束”应该是在算可用表达式的时候就体现出来了

2024/05/28 22:10

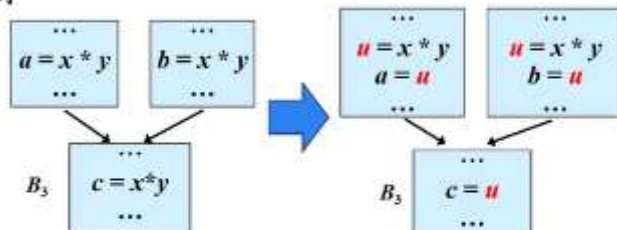


➤ 如果表达式  $x \text{ op } y$  先前已被计算过，并且从先前的计算到现在， $x \text{ op } y$  中变量的值没有改变，那么  $x \text{ op } y$  的这次出现就称为 **公共子表达式** (common subexpression)

### ① 删除全局公共子表达式

➤ 可用表达式的数据流问题可以帮助确定位于流图中  $p$  点的表达式是否为 **全局公共子表达式**

➤ 例



### 全局公共子表达式删除算法

➤ 输入：带有可用表达式信息的流图

➤ 输出：修正后的流图

➤ 方法：

➤ 对于语句  $s: z = x \text{ op } y$ ，如果  $x \text{ op } y$  在  $s$  之前可用，那么执行如下步骤：

① 从  $s$  开始逆向搜索，但不穿过任何计算了  $x \text{ op } y$  的块，找到所有离  $s$  最近的计算了  $x \text{ op } y$  的语句

② 建立新的临时变量  $u$

③ 把步骤①中找到的语句  $w = x \text{ op } y$  用下列语句代替：

$u = x \text{ op } y$   
 $w = u$

④ 用  $z = u$  替代  $s$

涉及到定义的也就这三页ppt了，第一张是广义的，后两个是全局公共子表达式相关的

#### 习题 19.1

下列语句的流图：

1. 计算并输出  $x + y$ 。

2. 计算并输出  $x + y$ 。如果  $x + y$  的值是偶数，那么将  $x$  和  $y$  的值都乘以 2。

3. 计算并输出  $x + y$ 。如果  $x + y$  的值是奇数，那么将  $x$  和  $y$  的值都乘以 2。

4. 计算并输出  $x + y$ 。如果  $x + y$  的值是偶数，那么将  $x$  和  $y$  的值都乘以 2。

5. 计算并输出  $x + y$ 。如果  $x + y$  的值是奇数，那么将  $x$  和  $y$  的值都乘以 2。

结合这个题来看，我感觉就是把这个循环里的公共子表达式找出来就行，不用在意有没有相关部分在循环外

结合这个题来看，我感觉就是把这个循环里的公共...

嗯，我的意思是讲公共子表达式的时候要不要明确他在哪个位置出现

2024/05/28 22:17

在算可用表达式分析的时候，ppt里只用了表达式，没有标出来位置，像这样

将相关的表达式加入到  $e\_kill_B$  中

块B语句	$e\_kill_B$	$U = \{b+c, a-d\}$
..... $a := b+c$	$\emptyset$	
..... $b := a-d$	$\{a-b\}$	—— 删除 $b+c$ , 加入 $a-b$
..... $c := b+c$	$\{b+c\}$	—— 删除 $a-d$ , 加入 $b+c$
..... $d := a-d$	$\{b+c\}$	—— 删除 $b+c$ , 加入 $b+c$
.....	$\{b+c, a-d\}$	—— 删除 $a-d$ , 加入 $a-d$

。但我觉得删除公共子表达式的时候，或许需要列出来删除的表达式出自哪一句

TSUNDOKU

sll练习题答案不是明确位置了吗

比如 $a+b$ 的话并不是在循环中的所有位置都是可用的，那它可以称为循环中的公共子表达式吗😂

TSUNDOKU

sll练习题答案不是明确位置了吗

对，我正要找这个@TSUNDOKU

很不幸，我找了sll的那道差不多的题，是这么问的

练习3. 对于下图中的流图：

- (1) 找出流图中的自然循环。
- (2)  $B_i$  中的语句 (1) 和 (2) 都是复制语句且  $a$  和  $b$  都被赋予了常量值。我们可以对  $a$  和  $b$  的哪些引用进行复制传播，并把对它们的引用替换为对一个常量的引用？在所有可能的地方进行这种替换。
- (3) 找出所有的全局公共子表达式。



2024/05/29 10:25



➤ 三地址语句

➤ *call callee*

➤ 目标代码: 调用序列

➤ *ADD SP, SP, #caller.recordsize*

➤ *ST 0(SP), #here + 16*

➤ *BR callee.codeArea*



想问一下这个的16是怎么计算出来的呀

TSUNDOKU



ST BR 这两个指令#here+16 callee.codeArea这两个常量, 每个算一个字, 占4字节

TSUNDOKU



$4 \times 4 = 16$

TSUNDOKU



0 (SP)是寄存器

TSUNDOKU



不占字节

2024/05/29 11:16



```
PROGRAM p;  
  VAR a, b, c, d, e: real;  
  PROCEDURE a;  
    VAR c, e, f, g: real;  
    BEGIN  
    ...
```

```

c;
...
END;
PROCEDURE b;

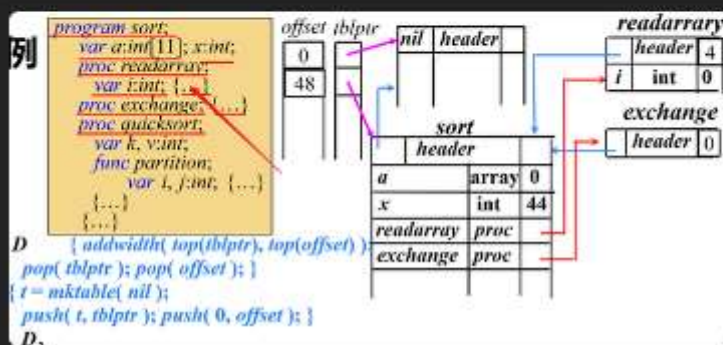
```

请问这个“c”要怎么往符号表里写呀，还是不需要管呢

TSUNDOKU

不用吧

TSUNDOKU



c相当于这里面的东西

这里的C是引用不是声明吧

之前声明过了

2024/05/29 12:15

## 基本块的 DAG 表示

例

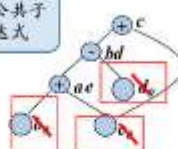
```

a = b + c
b = a - d
c = b + c
d = a - d
e = a

```

对于形如 $x = y + z$ 的三地址指令，如果已经有一个结点表示 $y + z$ ，就不向DAG中增加新的结点，而是给已经存在的结点附加定值变量 $x$

发现公共子表达式



基本块中的每个语句 $s$ 都对应一个内部结点 $N$

结点 $N$ 的标号是 $s$ 中的运算符；结点 $N$ 同时关联一个定值变量表，表示 $s$ 是在此基本块内最晚对表中变量进行定值的语句

$N$ 的子结点是基本块中在 $s$ 之前、最后一个对 $s$ 所使用的运算分量进行定值的语句对应的结点。如果 $s$ 的某个运算分量在基本块内没有在前被定值，则这个运算分量的子结点就是叶结点（其定值变量表中的变量加上下标0）

在为语句 $x = y + z$ 构造结点 $N$ 的时候，如果 $x$ 已经在某结点 $M$ 的定值变量表中，则从 $M$ 的定值变量表中删除变量 $x$

问一下，这个删除操作，对后续DAG的应用有什么意义吗，这个删除操作对代码优化有帮助吗

2024/05/29 12:33

TSUNDOKU



感觉只是用来分析变量的最新值，对DAG删除无用代码和找局部公共子表达式好像没什么用

2024/05/29 16:00



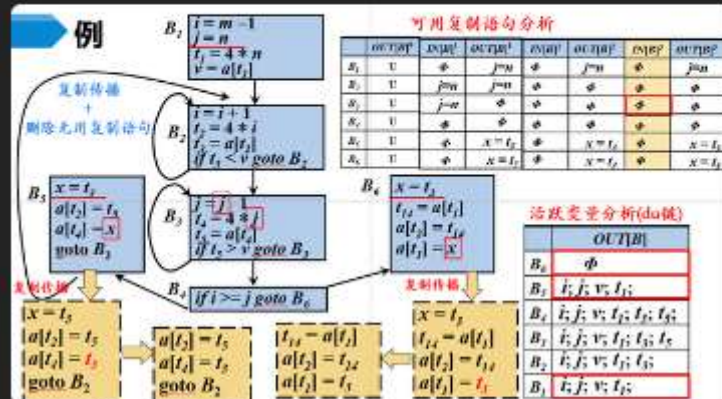
### 常用的优化方法

- 删除公共子表达式
- 删除无用代码



请问哪里用到了活跃变量分析

TSUNDOKU



B5中的x,出了B5就不活跃了

TSUNDOKU



就把x=t3删了

2024/05/29 22:48



$x=0;$

这种复制语句翻译成三地址代码还用不用添加临时变量了?

t1=0;x=t1这种?

还是直接翻译成x=0;

2024/05/29 22:58

TSUNDOKU

直接就行吧

TSUNDOKU

**赋值语句的SDT**

符号	综合属性
$S$	code
$E$	code    addr

$S \rightarrow id = E; \{ p = \text{lookup}(id, lexeme); \text{if } p = \text{nil then error};$   
     $S.code = E.code \parallel$   
     $\text{gen } p := E.addr; \}$

$E \rightarrow E_1 + E_2 \{ E.addr = \text{newtemp}();$   
     $E_1.code = E_1.code \parallel E_2.code \parallel$   
     $\text{gen}(E.addr := E_1.addr '+' E_2.addr);$   
     $\}$

$E \rightarrow E_1 * E_2 \{ E.addr = \text{newtemp}();$   
     $E_1.code = E_1.code \parallel E_2.code \parallel$   
     $\text{gen}(E.addr := E_1.addr '*' E_2.addr);$   
     $\}$

$E \rightarrow -E_1 \{ E.addr = \text{newtemp}();$   
     $E_1.code = E_1.code \parallel$   
     $\text{gen}(E.addr := \text{'aminus'} E_1.addr); \}$

$E \rightarrow (E_1) \{ E.addr = E_1.addr;$   
     $E.code = E_1.code; \}$

$E \rightarrow id \{ E.addr = \text{lookup}(id, lexeme); \text{if } E.addr = \text{nil then error};$   
     $E.code = \epsilon; \}$

**newtemp():** 生成一个新的临时变量t, 返回t的地址

**gen(code):** 生成三地址指令code

**lookup(name):** 查询符号表, 返回name对应的记录

TSUNDOKU

这几种生成临时变量了

2024/05/30 16:10

语法分析是先消除左递归还是先提取左公因子?

先消除左递归

按老师的说法, 先消除左递归可能消掉左公因子, 先消除左公因子可能产生左递归



2024/05/30 16:49



为了实现数组引用翻译，除了数组的类型和数组的基地址，还需要符号表中数组的哪个信息？

- ☐ A 数组各子类型的宽度
- ☐ B 数组的维度
- ☐ C 数组引用的下标
- ☐ D 数组类型本身的宽度

请问这个为什么不选c?  $a[i] = \text{base} + i * w$ , A选项就是w, C选项就是i吧



计科-琪铠

为了实现数组引用翻译，除了数组的类型和数组的基地址，还需要符号表中数组的哪个信息？

- ☐ A 数组各子类型的宽度
- ☐ B 数组的维度
- ☐ C 数组引用的下标
- ☐ D 数组类型本身的宽度

@计科-琪铠 符号表中信息?

2024/05/30 16:57



计科-琪铠

为了实现数组引用翻译，除了数组的类型和数组的基地址，还需要符号表中数组的哪个信息？

- ☐ A 数组各子类型的宽度
- ☐ B 数组的维度
- ☐ C 数组引用的下标
- ☐ D 数组类型本身的宽度

@计科-琪铠 i的信息不存在符号表中数组那块

2024/05/30 17:05



@计科-琪铠 i的信息不存在符号表中数组那块

好的

在自底向上的翻译中，假设栈顶已形成句柄：  
 $A \rightarrow XYZ$ ，且  $A$  有继承属性  $A.inh$ ，那么  $A.inh$  的位置在

- A top
- B top-1
- C top-2
- D top-3

这个题为什么选d啊

TSUNDOKU

答案解析

归约后，XYZ被弹出栈，A被压入栈，A将在现在X的位置top-2，A的继承属性一定在紧挨着A之下即top-3的位置

2024/05/30 17:11

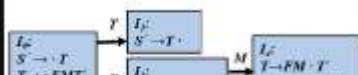
这里使用自底向上翻译，那对应S-SDD吧，S-SDD不是没有继承属性吗

TSUNDOKU

应该是改装

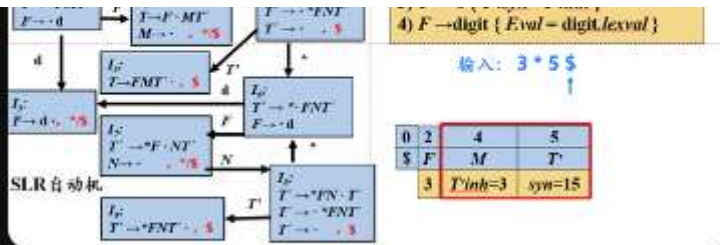
TSUNDOKU

例



- 1)  $T \rightarrow FMT' \{ T.val = T'.syn \}$   
 $M \rightarrow \varepsilon \{ M.i = Eval; M.s = M.i \}$
- 2)  $T' \rightarrow *FNT_1' \{ T'.syn = T_1'.syn \}$   
 $N \rightarrow \varepsilon \{ N.i1 = T'.inh; N.i2 = Eval; N.s = N.i1 \times N.i2 \}$
- 3)  $T' \rightarrow \varepsilon \{ T'.syn = T'.inh \}$

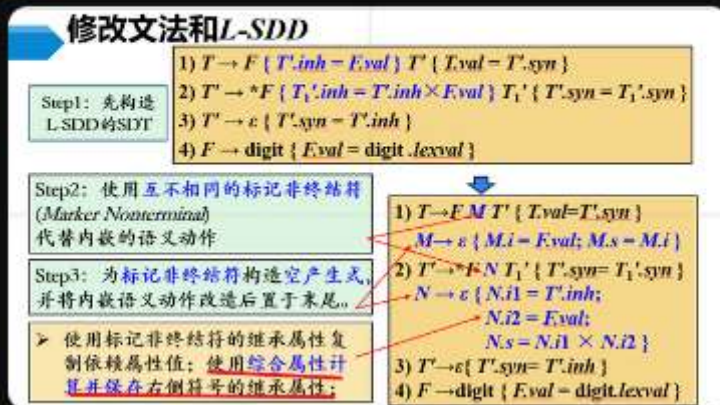




TSUNDOKU

类似这种

TSUNDOKU



2024/05/31 21:11

五、(8分)给定基本块:

$A_0=0$   
 $B_0=0$   
 $C_0=12$   
 $D_0=0$   
 $A_1=0$   
 $C_1=1$   
 $B_1=0$

假定块基本块后, 只有 A、B、C、D 能活跃, 给出用 DAG 图完成优化后的代码序列。

@新李 有老哥写了这个的答案吗

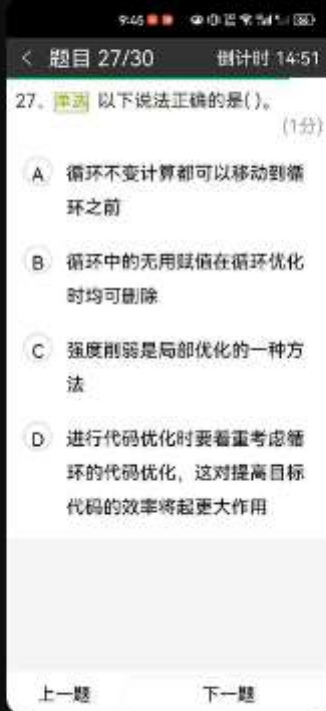
2024/05/31 21:16

我猜  $c=28$ ,  $e=e+f$ ,  $a=e+12$

我猜  $c=28$ ,  $e=e+f$ ,  $a=e+12$

@我也这么觉得

2024/05/31 21:45



xdm选啥



d



但ab哪错了

2024/05/31 21:50



a 并不都可以

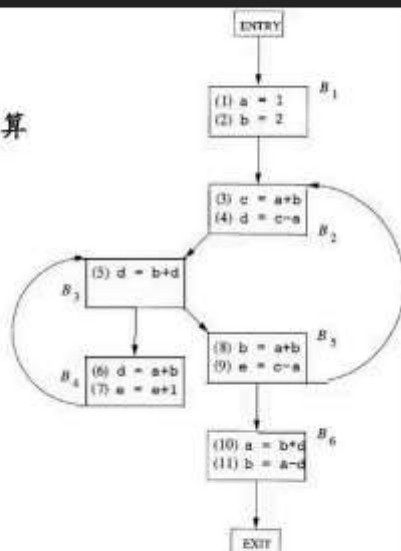


b 循环中无用 外面可能活跃

2024/06/01 11:37

## 习题18.1

对下图中的流图，计算可用表达式问题中的  $e\_gen$ 、 $e\_kill$ 、IN和OUT集合。



请问一下在计算B1的 $e\_gen$ 时不用加入 $a=1$ 、 $b=1$ ，看后面有个题涉及到复制传播

要吧

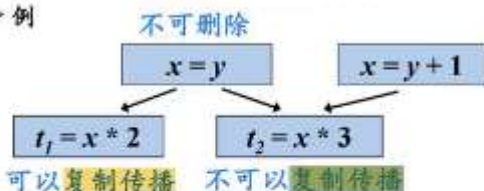
复制也要加吗？

我以为只加右边是表达式那种

## ② 删除复制语句

对于复制语句 $s: x=y$ ，如果在 $x$ 的所有引用点都可以用对 $y$ 的引用代替对 $x$ 的引用（复制传播），那么可以删除复制语句 $x=y$

例

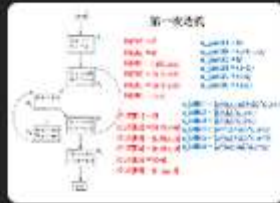


在 $x$ 的引用点 $u$ 用 $y$ 代替 $x$ （复制传播）的条件

复制语句 $s: x=y$ 在 $u$ 点“可用”

考虑复制传播得看有没有重定义吧😭

反正算kill的时候得算这俩



看这个答案里没有，感觉很奇怪

算gen应该算不算都行

2024/06/01 12:39



我又看了下概念，e\_gen和e\_kill的元素是表达式x op y。a=1, b=2是定值，和e\_gen、e\_kill没关系吧

应该和kill有关

答案给的e\_killB1也确实不是空集

2024/06/01 12:45

在复制传播时怎么说明可用呢？

kill不为空我可以理解，因为对a和b重新定值且没有再

$e\_kill$ 不空我可以理解，因为对 $a$ 和 $b$ 里新定值是肯定有再计算，杀死相应的表达式

😬在复制传播时怎么说明可用呢？ $e\_kill$ 不空我可以理解...

感觉ppt这里不太严谨。复制传播那里的意思是把 $x = y$ 这种也当作一个表达式去分析是不是可用的

2024/06/01 13:26

我的想法是这快他会把 $x = y$ 作为一个表达式。

把 $x = y$ 当做表达式？

嗯，他不是说 $x$ 等于 $y$ 是可以用的。

相当于前面的 $x \text{ op } y$ 是可用的。

2024/06/01 14:48

6.进行移入-规约分析时，

请问这个填什么

分析栈中内容 + 剩余输





$a[t2]=t5$ ,  $t5=a[t2]$ , 这里的 $a$ 是常量, 不涉及引用定值,  $t2$ 都算做引用