五、目标代码生成
(17. 寄存器分配)

魏恒峰

hfwei@nju.edu.cn

2024 年 06 月 14 日

Register Allocation

The *Register Allocation problem* consists in mapping a program P_v, that can use an unbounded number of virtual registers, to a program P_p that contains a finite (possibly small) number of physical registers. Each target architecture has a different number of physical registers. If the number of physical registers is not enough to accommodate all the virtual registers, some of them will have to be mapped into memory. These virtuals are called *spilled virtuals*.

Register Allocation

The *Register Allocation problem* consists in mapping a program P_v, that can use an unbounded number of virtual registers, to a program P_p that contains a finite (possibly small) number of physical registers. Each target architecture has a different number of physical registers. If the number of physical registers is not enough to accommodate all the virtual registers, some of them will have to be mapped into memory. These virtuals are called *spilled virtuals*.

这也是一个组合优化问题

```
1  int sum(int n){
2    int s = 0, i = 0;
3    while(i <= n){
4      s += i;
5      i += 1;
6    }
7    return s;
8  }
```

```
1  int sum(int n):
2    L0:
3      s = 0;
4      i = 0;
5      goto L1;
6    L1:
7      if(i<=n, L2, L3);
8    L2:
9      s = s + i;
10     i = i + 1;
11     goto L1;
12   L3:
13     return s;
14 }
```

```
1  int sum(int n){
2    int s = 0, i = 0;
3    while(i <= n){
4      s += i;
5      i += 1;
6    }
7    return s;
8  }
```

```
1  int sum(int n):
2    L0:
3      s = 0;
4      i = 0;
5      goto L1;
6    L1:
7      if(i<=n, L2, L3);
8    L2:
9      s = s + i;
10     i = i + 1;
11     goto L1;
12   L3:
13     return s;
14 }
```

以非 SSA 形式的中间代码为例

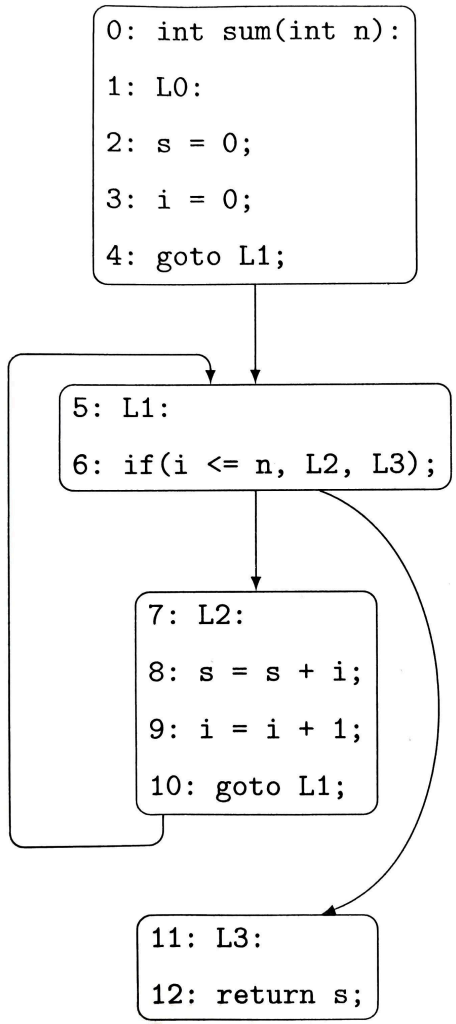
```
1  int sum(int n):
2    L0:
3      s = 0;
4      i = 0;
5      goto L1;
6    L1:
7      if(i<=n, L2, L3);
8    L2:
9      s = s + i;
10     i = i + 1;
11     goto L1;
12   L3:
13     return s;
14 }
```

问题 1: 变量 n, s, i 的活跃区间 (live interval) 分别是什么?

```
1   int sum(int n):
2     L0:
3       s = 0;
4       i = 0;
5       goto L1;
6     L1:
7       if(i<=n, L2, L3);
8     L2:
9       s = s + i;
10      i = i + 1;
11      goto L1;
12    L3:
13      return s;
14  }
```

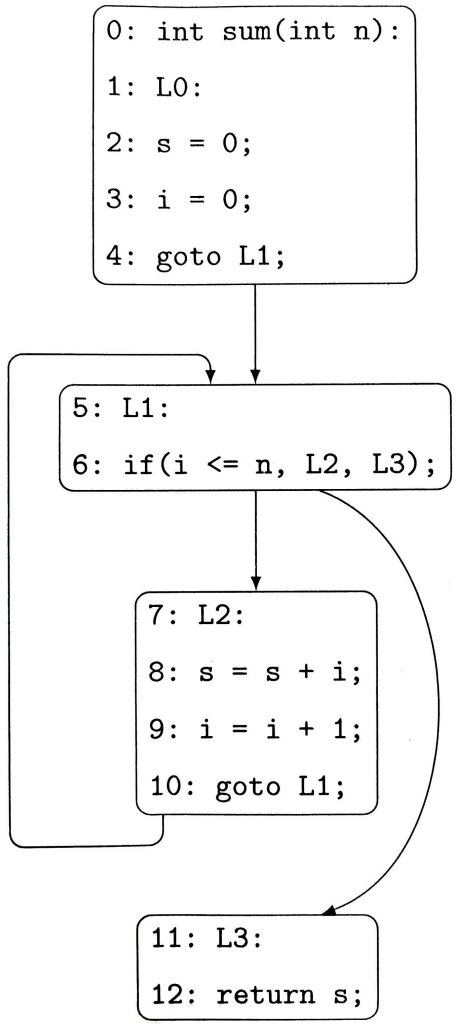
问题 1: 变量 n, s, i 的活跃区间 (live interval) 分别是什么?

```
1   int sum(int n):
2     L0:
3       s = 0;
4       i = 0;
5       goto L1;
6     L1:
7       if(i<=n, L2, L3);
8     L2:
9       s = s + i;
10      i = i + 1;
11      goto L1;
12    L3:
13      return s;
14  }
```



问题 1: 变量 n, s, i 的活跃区间 (live interval) 分别是什么?

```
1  int sum(int n):
2    L0:
3      s = 0;
4      i = 0;
5      goto L1;
6    L1:
7      if(i<=n, L2, L3);
8    L2:
9      s = s + i;
10     i = i + 1;
11     goto L1;
12   L3:
13     return s;
14 }
```



问题 2: 在第 3 行后, 有哪些变量是活跃的?

Definition (活跃 (Live))

对于给定的变量 x, 考虑从其一个定义点 p 到使用点 q 的路径 l。

对于该路径 l 上的任意点 r, 如果 r 和 q 之间没有对变量 x 的其它定义, 则称 x 在程序点 r 上是活跃的。

Definition (活跃 (Live))

对于给定的变量 x, 考虑从其一个定义点 p 到使用点 q 的路径 l。

对于该路径 l 上的任意点 r, 如果 r 和 q 之间没有对变量 x 的其它定义, 则称 x 在程序点 r 上是活跃的。

在同一个程序点上活跃的变量是有冲突的, 不能分配到同一个寄存器。

Definition (活跃 (Live))

对于给定的变量 x, 考虑从其一个定义点 p 到使用点 q 的路径 l。

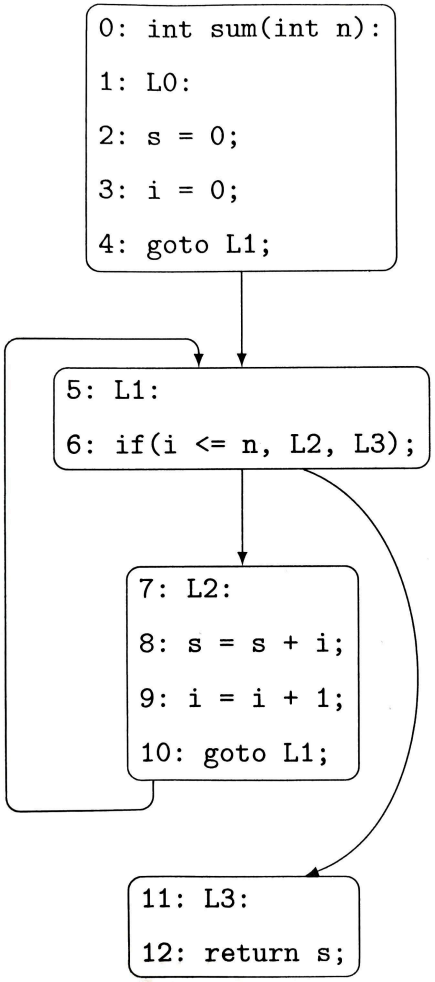
对于该路径 l 上的任意点 r, 如果 r 和 q 之间没有对变量 x 的其它定义, 则称 x 在程序点 r 上是*活跃*的。

在同一个程序点上活跃的变量是有冲突的, 不能分配到同一个寄存器。

Definition (活跃分析 (Liveness Analysis))

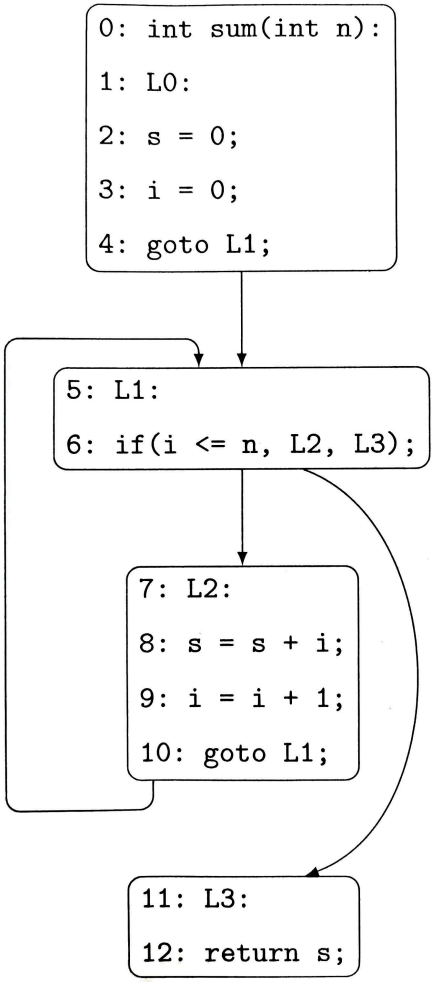
分析变量的活跃点的程序分析被称为 *活跃分析*。

LiveIn(s) : s 执行前的活跃变量集合



LiveOut(s) : s 执行后的活跃变量集合

$\textsc{LiveIn}(s)$: s 执行前的活跃变量集合

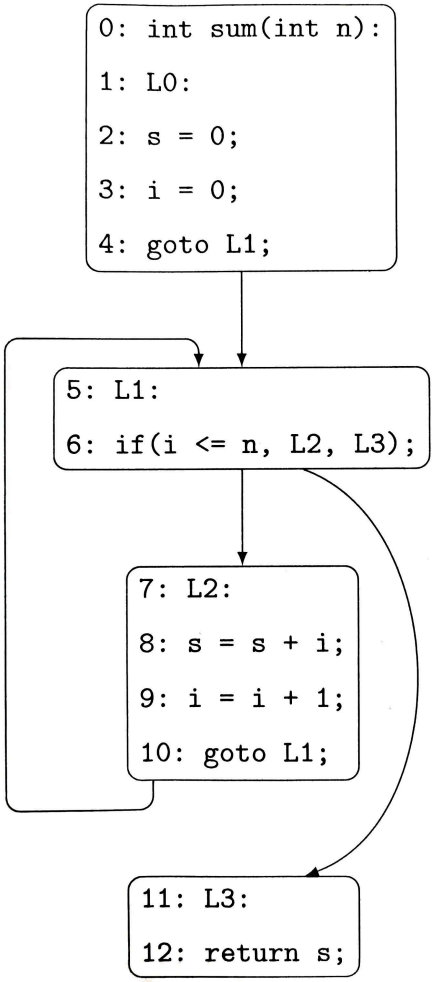


$$\textsc{LiveOut}(s) = \bigcup_{p \in succ(s)} \textsc{LiveIn}(p)$$

$$\textsc{LiveIn}(s) = (\textsc{LiveOut}(s) \setminus \mathsf{def}(s)) \cup \mathsf{use}(s)$$

$\textsc{LiveOut}(s)$: s 执行后的活跃变量集合

$\textsc{LiveIn}(s)$: s 执行前的活跃变量集合

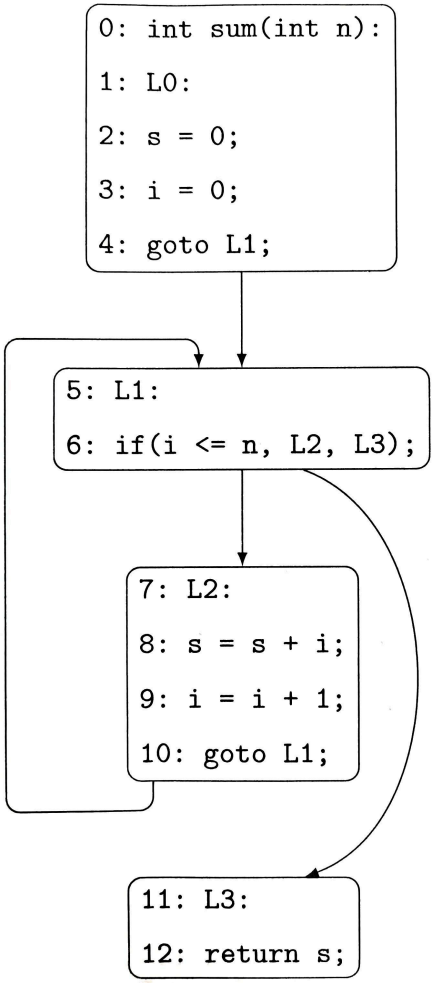


$$\textsc{LiveOut}(s) = \bigcup_{p \in succ(s)} \textsc{LiveIn}(p)$$

$$\textsc{LiveIn}(s) = (\textsc{LiveOut}(s) \setminus \mathsf{def}(s)) \cup \mathsf{use}(s)$$

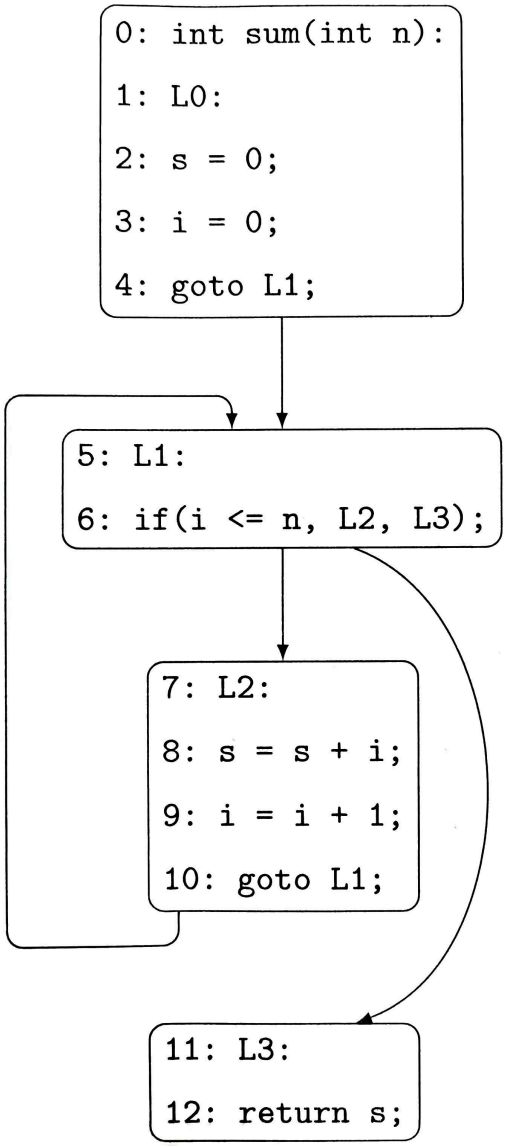
如何求解这个“数据流”方程组?

$\textsc{LiveOut}(s)$: s 执行后的活跃变量集合

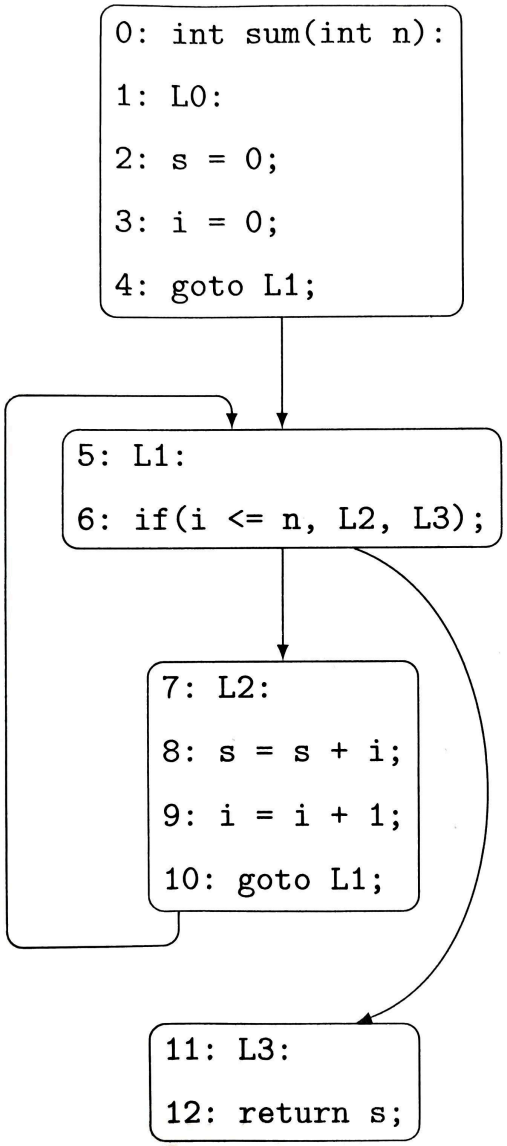


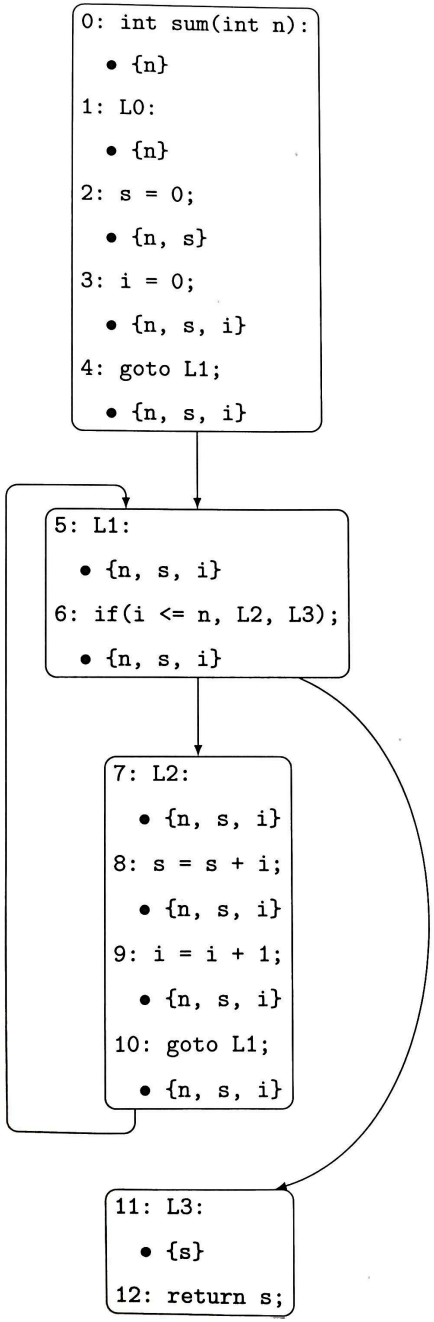

```
1  void liveness(program p){
2    for(each statement s in p){
3      liveIn[s] = ϕ;
4      liveOut[s] = ϕ;
5    }
6    while(liveIn or liveOut set still changing){
7      for(each statement s in p){
8        liveOut[s] = ⋃_i liveIn(p_i); // p_i are successor of s
9        liveIn[s] = (liveOut[s] - def(s)) ⋃ use(s);
10     }
11   }
12 }
```

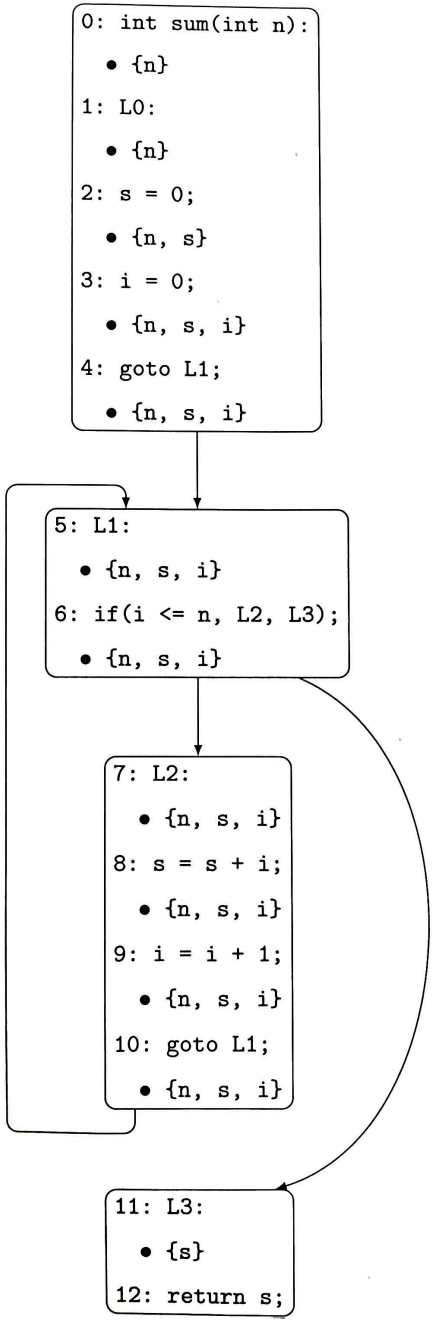
不动点算法



语句	use	def	out/in 初始值	out	in	out	in
2	∅	s	∅	n,s	n	…	…
3	∅	i	∅	i,n,s	n,s	…	…
6	i,n	∅	∅	i,s	i,n,s	…	…
8	i,s	s	∅	i	i,s	…	…
9	i	i	∅	∅	i	…	…
12	s	∅	∅	∅	s	…	…

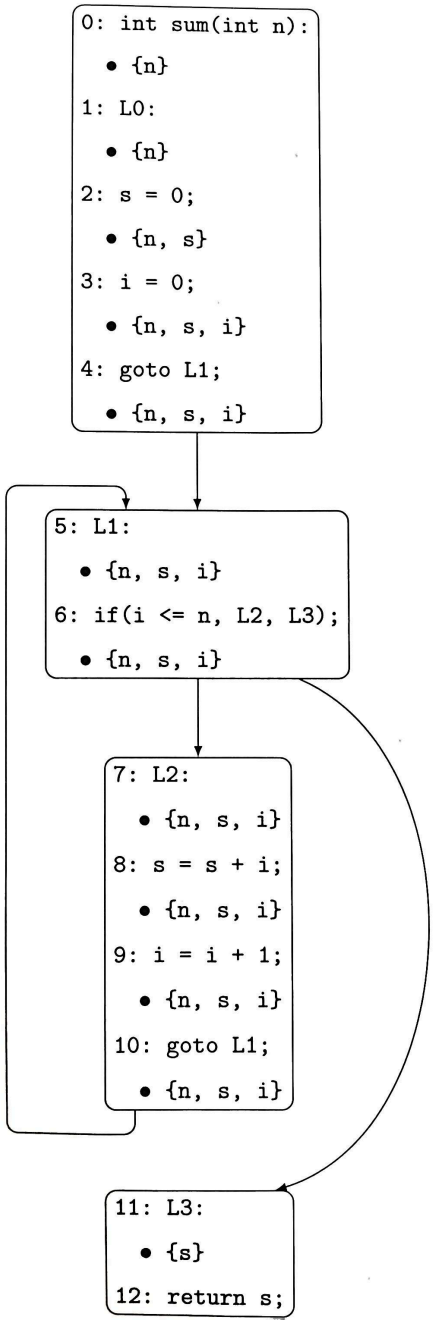

0: int sum(int n):
1: L0:
2: s = 0;
3: i = 0;
4: goto L1;
5: L1:
6: if(i <= n, L2, L3);
7: L2:
8: s = s + i;
9: i = i + 1;
10: goto L1;
11: L3:
12: return s;


0: int sum(int n):
• {n}
1: L0:
• {n}
2: s = 0;
• {n, s}
3: i = 0;
• {n, s, i}
4: goto L1;
• {n, s, i}
5: L1:
• {n, s, i}
6: if(i <= n, L2, L3);
• {n, s, i}
7: L2:
• {n, s, i}
8: s = s + i;
• {n, s, i}
9: i = i + 1;
• {n, s, i}
10: goto L1;
• {n, s, i}
11: L3:
• {s}
12: return s;



s.index：语句 s 的行号

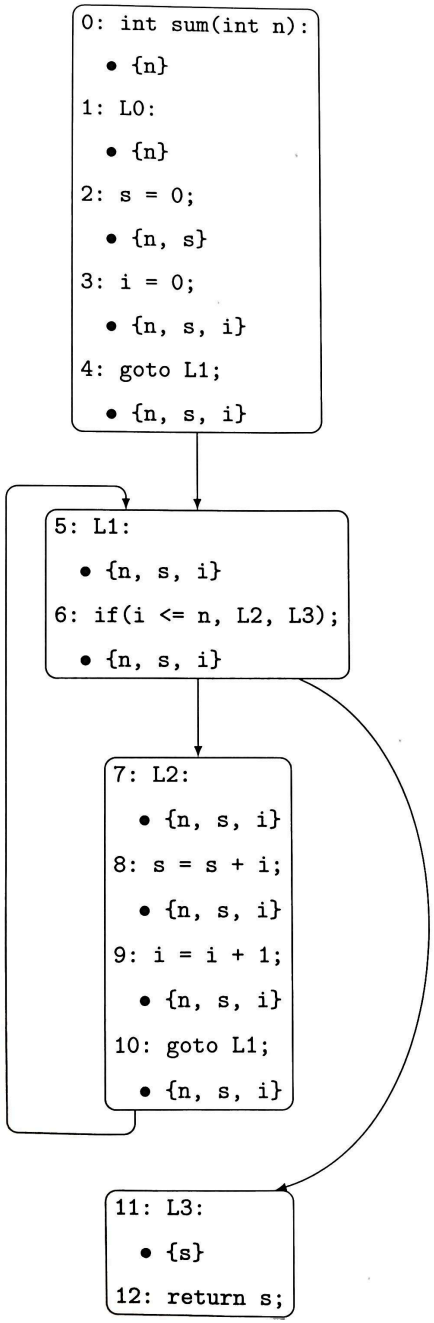
```
14  // Input: a list of basic blocks, which have been linearized
15  // for any variable x in the program, update the lower and upper bound
16  // of variable x in the map "interval"
17  void calculate_intervals(block blocks[]){
18    for(each variable x in this program)
19      for(each statement s in blocks)
20        if(x ∈ liveOut(s)){ // x is live at statement s
21          if(s.index < interval[x].l)
22            interval[x].l = s.index;
23          if(s.index > interval[x].h)
24            interval[x].h = s.index;
25        }
26  }
```



s.index：语句 s 的行号

```
14  // Input: a list of basic blocks, which have been linearized
15  // for any variable x in the program, update the lower and upper bound
16  // of variable x in the map "interval"
17  void calculate_intervals(block blocks[]){
18    for(each variable x in this program)
19      for(each statement s in blocks)
20        if(x ∈ liveOut(s)){ // x is live at statement s
21          if(s.index < interval[x].l)
22            interval[x].l = s.index;
23          if(s.index > interval[x].h)
24            interval[x].h = s.index;
25        }
26  }
```

$$n : [0, 10] \quad s : [2, 10] \quad i : [3, 11]$$



$n : [0, 10] \quad s : [2, 10] \quad i : [3, 11]$

线性扫描分配算法 @ TOPLAS1999

Linear Scan Register Allocation

MASSIMILIANO POLETTO
Laboratory for Computer Science, MIT
and
VIVEK SARKAR
IBM Thomas J. Watson Research Center

三大关键操作: **占用、释放、溢出**

$x_1 : [2, 16]$

$x_2 : [2, 20]$

$x_3 : [7, 8]$

$x_4 : [9, 10]$

$x_5 : [11, 12]$

$x_6 : [15, 19]$

$x_7 : [17, 19]$

$|R| = 3 \quad (R_1, R_2, R_3)$

$x_1 : [2, 16]$

$x_2 : [2, 20]$

$x_3 : [7, 8]$

$x_4 : [9, 10]$

$x_5 : [11, 12]$

$x_6 : [15, 19]$

$x_7 : [17, 19]$

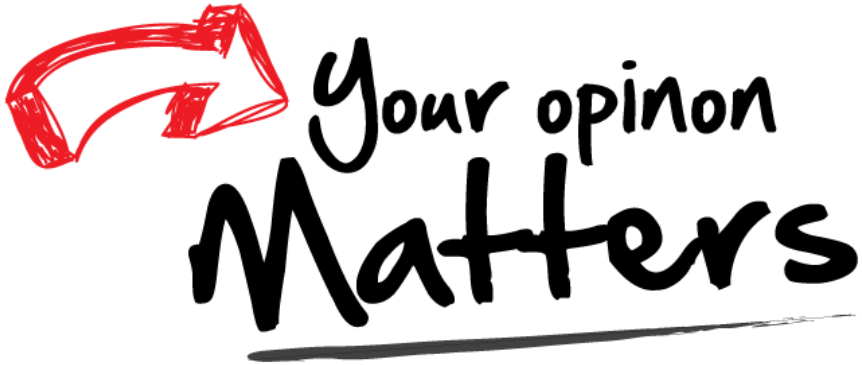
$|R| = 2 \quad (R_1, R_2)$

如何溢出 (两种方法, 避免迭代)

伪线性序问题

ϕ 消除问题


Thank
You!



Office 926

hfwei@nju.edu.cn