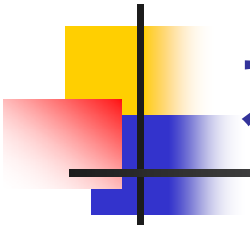
中间表示：活性分析

编译原理

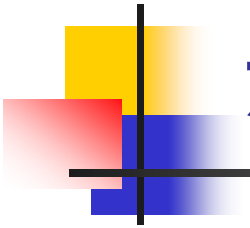
华保健

bjhua@ustc.edu.cn



进行活性分析的动机

- 在代码生成的讨论中，我们曾假设目标机器有无限多个（虚拟）寄存器可用
  - 简化了代码生成的算法
  - 对物理机器是个坏消息
    - 机器只有有限多个寄存器
      - 必须把无限多个虚拟寄存器分配到有限个寄存器中
- 这是寄存器分配优化的任务
  - 需要进行活性分析



示例

考虑这段三地址码：

```
a = 1

b = a + 2

c = b + 3

return c
```

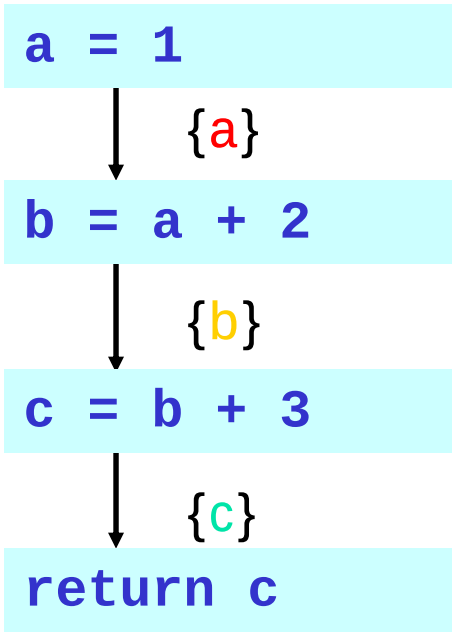
有三个变量a，b，c.

假设目标机器上只有一个物理寄存器：r.

是否可能把三个变量a，b，c同时放到寄存器r中?

示例

考虑这段三地址码：



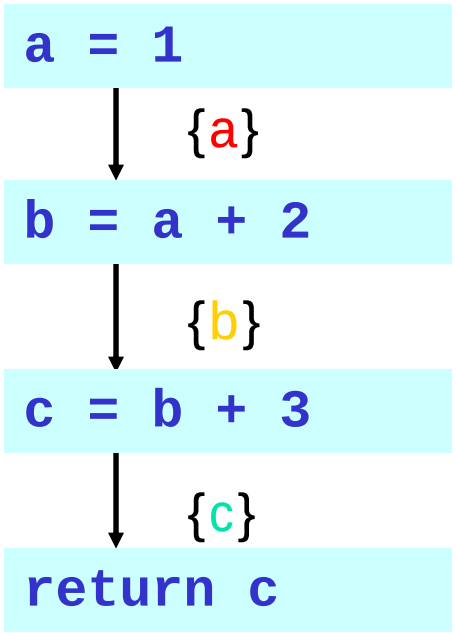
计算在给定的程序点，哪些变量是“活跃”的

活跃信息给出了活跃区间的概念.

活跃区间互不相交，所以三个变量可交替使用同一个寄存器。

示例

考虑这段三地址码：



寄存器分配：

a => r

b => r

c => r

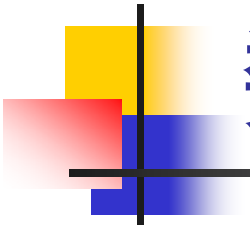
代码重写：

r = 1

r = r + 2

r = r + 3

return r



数据流方程

对任何一条语句：

[d: s]

给出两个集合：

gen[d: s] = {x | 变量x在语句s中被使用}

kill[d: s] = {x | 变量x在语句s中被定义}

```
1: x = y + z
2: z = z + x
```

数据流方程

- 基本块内的后向数据流方程：

$out[s_i] = in[s_{i+1}]$

$in[s] = gen[s] \cup (out[s] - kill[s])$

// 示例1:

a = 1

int

b = a + 2

out

c = b + 3

return c

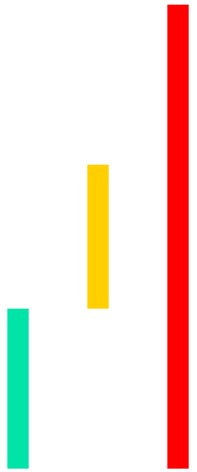
// 示例2:

a = 1

b = a + 2

c = b + 3

return a + c



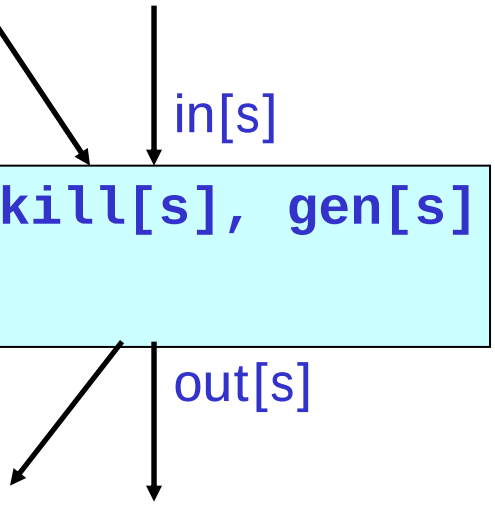
一般的数据流方程

- 方程:

$$out[s] = \bigcup_{p \in succ[s]} in[p]$$

$$in[s] = gen[s] \cup (out[s] - kill[s])$$

- 同样可给出不动点算法
  - 从初始的空集{}出发
  - 循环到没有集合变化为止



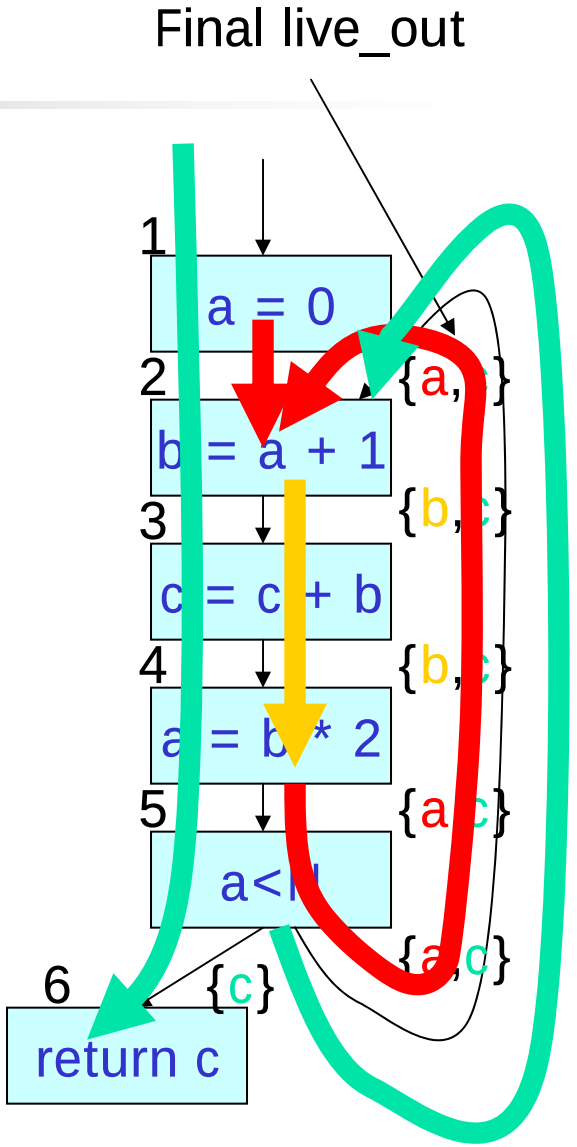
$$out[s] = \cup_{p \in succ[s]} in[p]$$
$$in[s] = gen[s] \cup (out[s]-kill[s])$$

示例

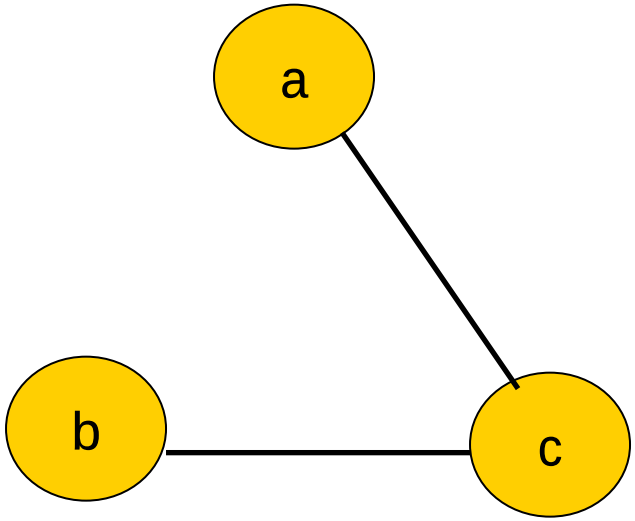
	in/out	in/out	in/out	in/out	in/out
1	{} {}	{} {}	{} {a}	…	
2	{} {}	{a} {}	{a} {b,c}	…	
3	{} {}	{b,c} {}	{b,c}{b}	…	
4	{} {}	{b} {}	{b}{a,c}	…	
5	{} {}	{a} {a}	{a}{a,c}	…	
6	{} {}	{c} {}	{c} {}	…	

语句的遍历顺序: 1, 2, 3, 4, 5, 6

node	1	2	3	4	5	6
def	{a}	{b}	{c}	{a}	{}	{}
use	{}	{a}	{b, c}	{b}	{a, N}	{c}



干扰图



干扰图是一个无向图G=(V，E)：

1. 对每个变量构造无向图G中一个节点；

2. 若变量x，y同时活跃，在x、y间连一条无向边。