CS30017 编译

第八讲：静态单赋值

徐 辉

xuh@fudan.edu.cn

大纲

一、消除冗余load/store

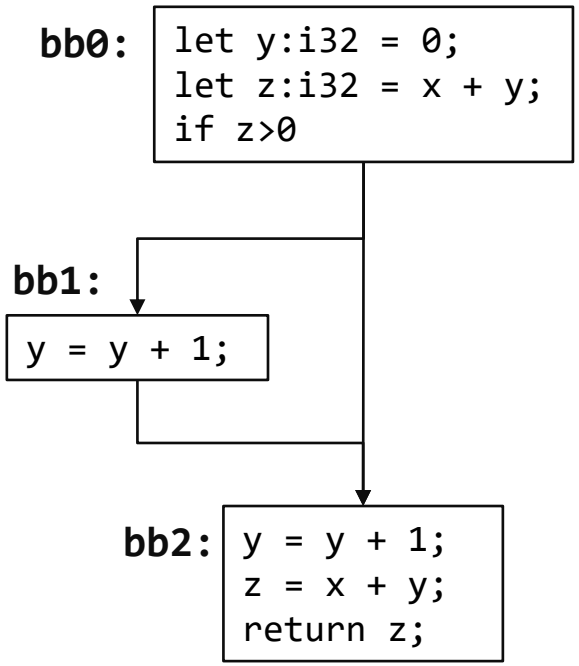
二、纯寄存器表示

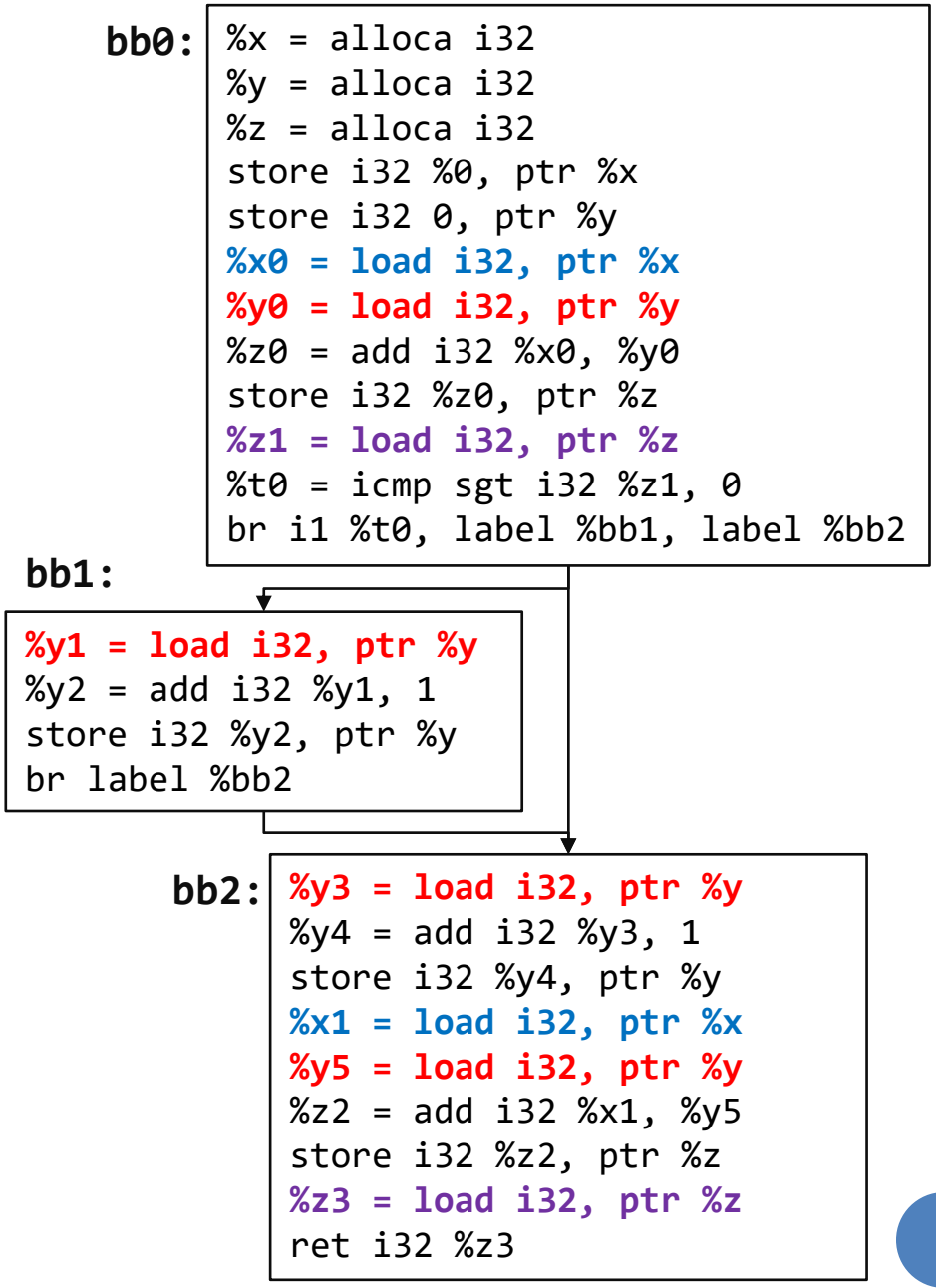
三、Phi指令优化

一、消除冗余load/store

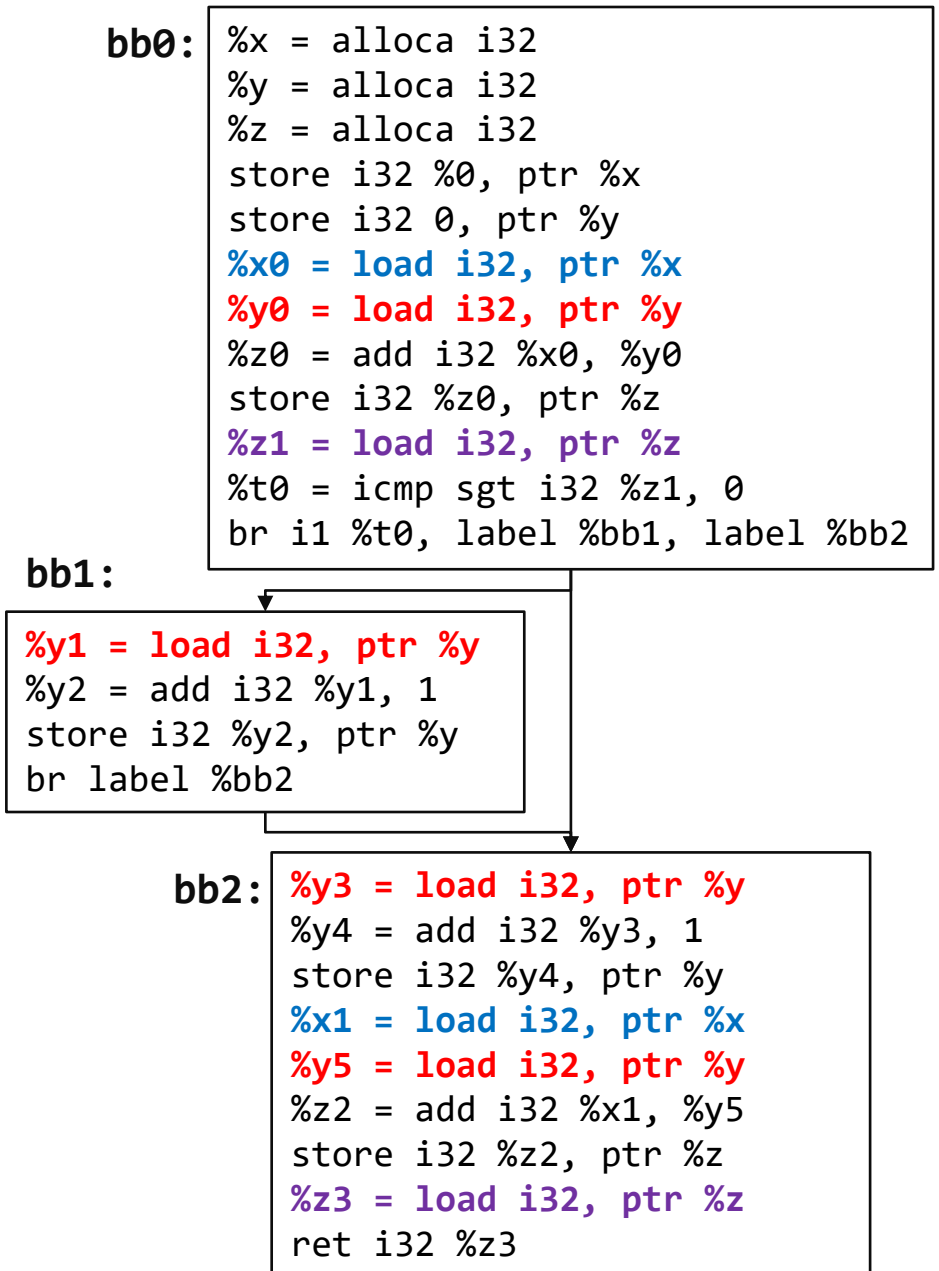
线性IR中的load冗余

```
fn foo(x:i32) -> i32
```





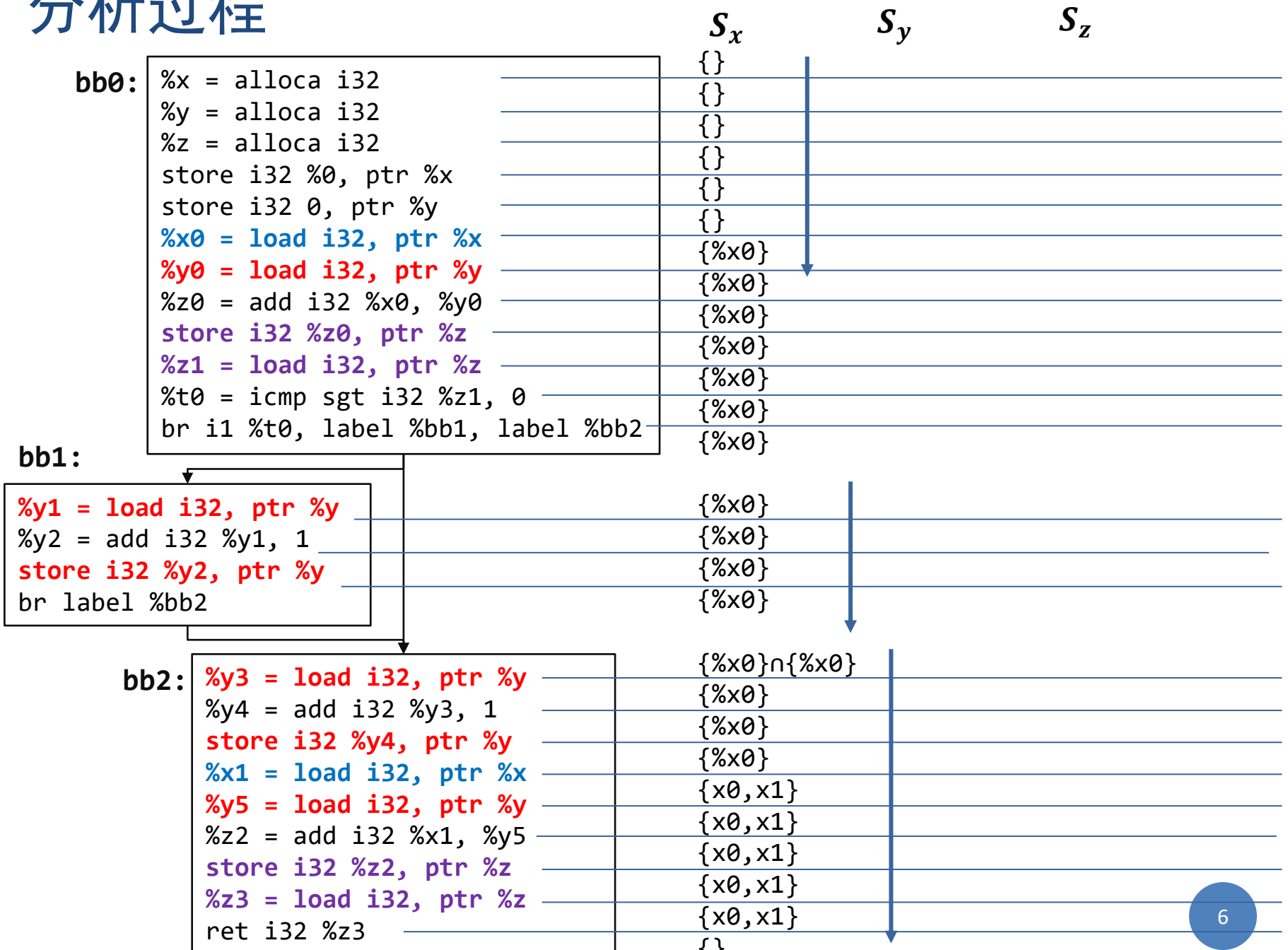
优化思路：可用寄存器分析



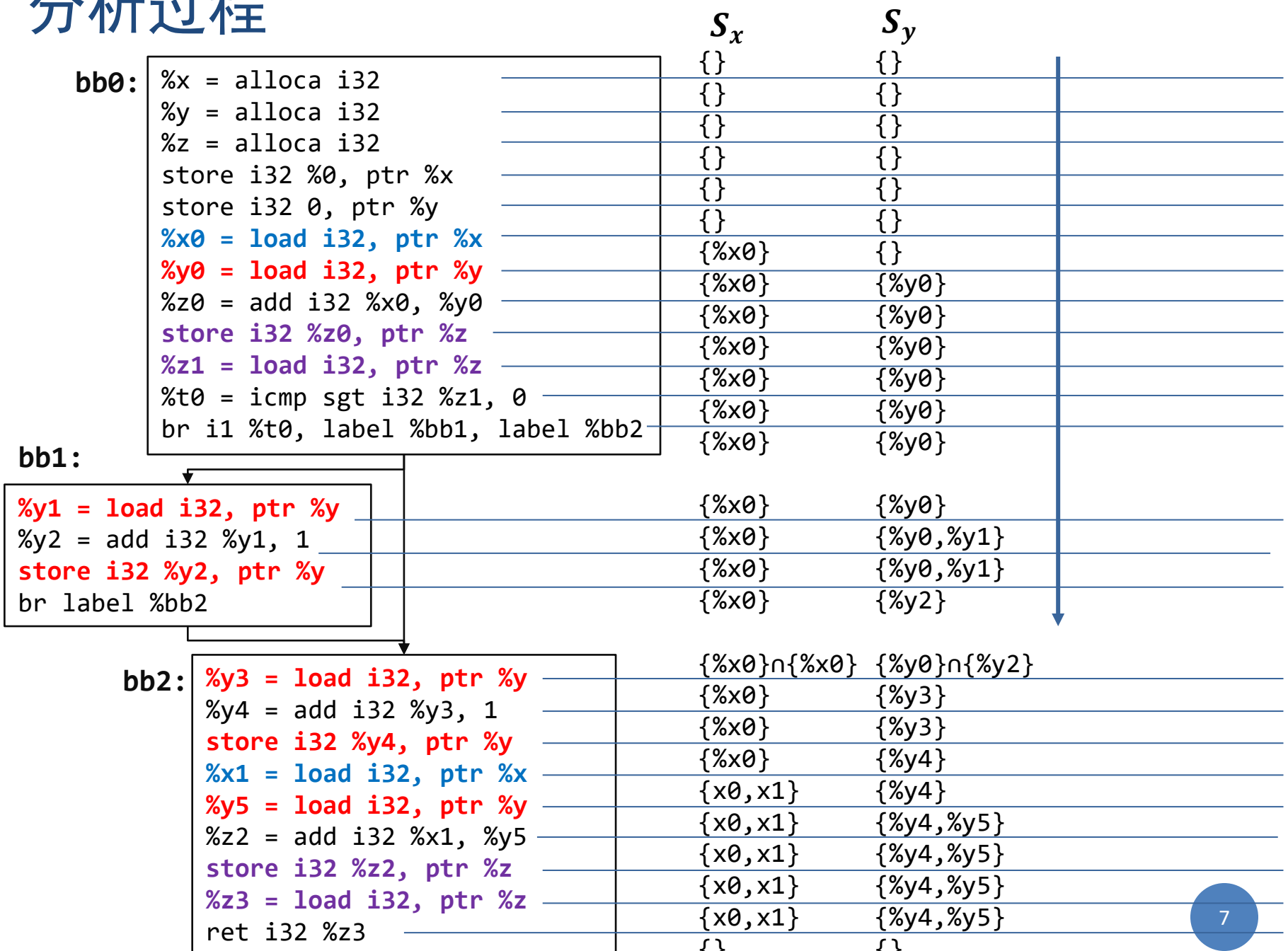
- 正向遍历控制流图
- Transfer函数定义：
  - `%t = load i32, ptr %x`
    - $S_x = S_x \cup \{t\}$
  - `store i32 %t, ptr %x`
    - $S_x = \{t\}$
- 遇到合并节点

$$\mathrm{IN}(n) = \bigcap_{n' \in \mathrm{predecessor}(n)} \mathrm{OUT}(n')$$

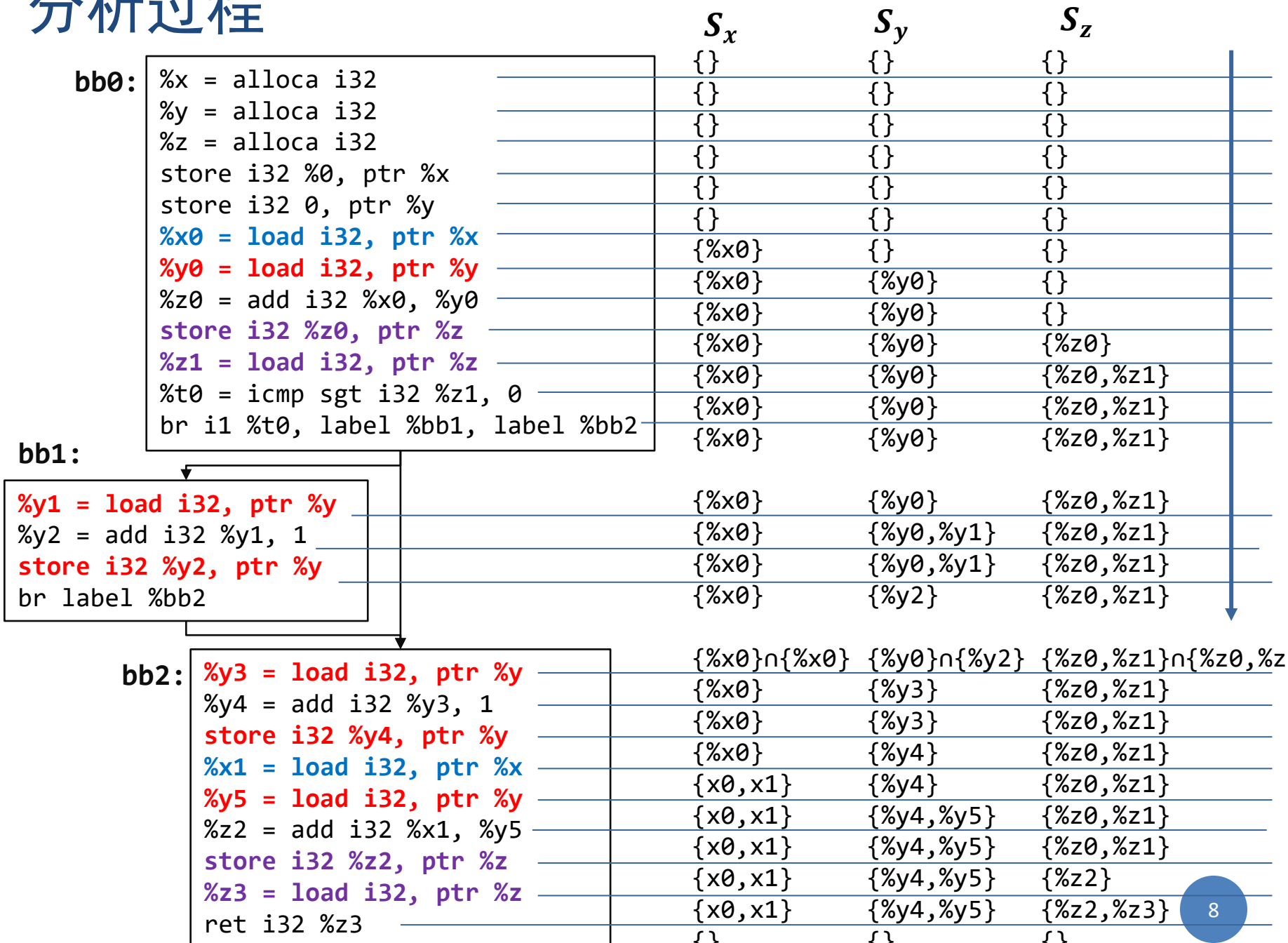
分析过程


S_x
S_y
S_z
bb0:
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 %0, ptr %x
store i32 0, ptr %y
%x0 = load i32, ptr %x
%y0 = load i32, ptr %y
%z0 = add i32 %x0, %y0
store i32 %z0, ptr %z
%z1 = load i32, ptr %z
%t0 = icmp sgt i32 %z1, 0
br i1 %t0, label %bb1, label %bb2
{}
{}
{}
{}
{}
{}
{%x0}
{%x0}
{%x0}
{%x0}
{%x0}
{%x0}
{%x0}
bb1:
%y1 = load i32, ptr %y
%y2 = add i32 %y1, 1
store i32 %y2, ptr %y
br label %bb2
{%x0}
{%x0}
{%x0}
{%x0}
bb2:
%y3 = load i32, ptr %y
%y4 = add i32 %y3, 1
store i32 %y4, ptr %y
%x1 = load i32, ptr %x
%y5 = load i32, ptr %y
%z2 = add i32 %x1, %y5
store i32 %z2, ptr %z
%z3 = load i32, ptr %z
ret i32 %z3
{%x0}∩{%x0}
{%x0}
{%x0}
{%x0}
{x0,x1}
{x0,x1}
{x0,x1}
{x0,x1}
{x0,x1}
{}

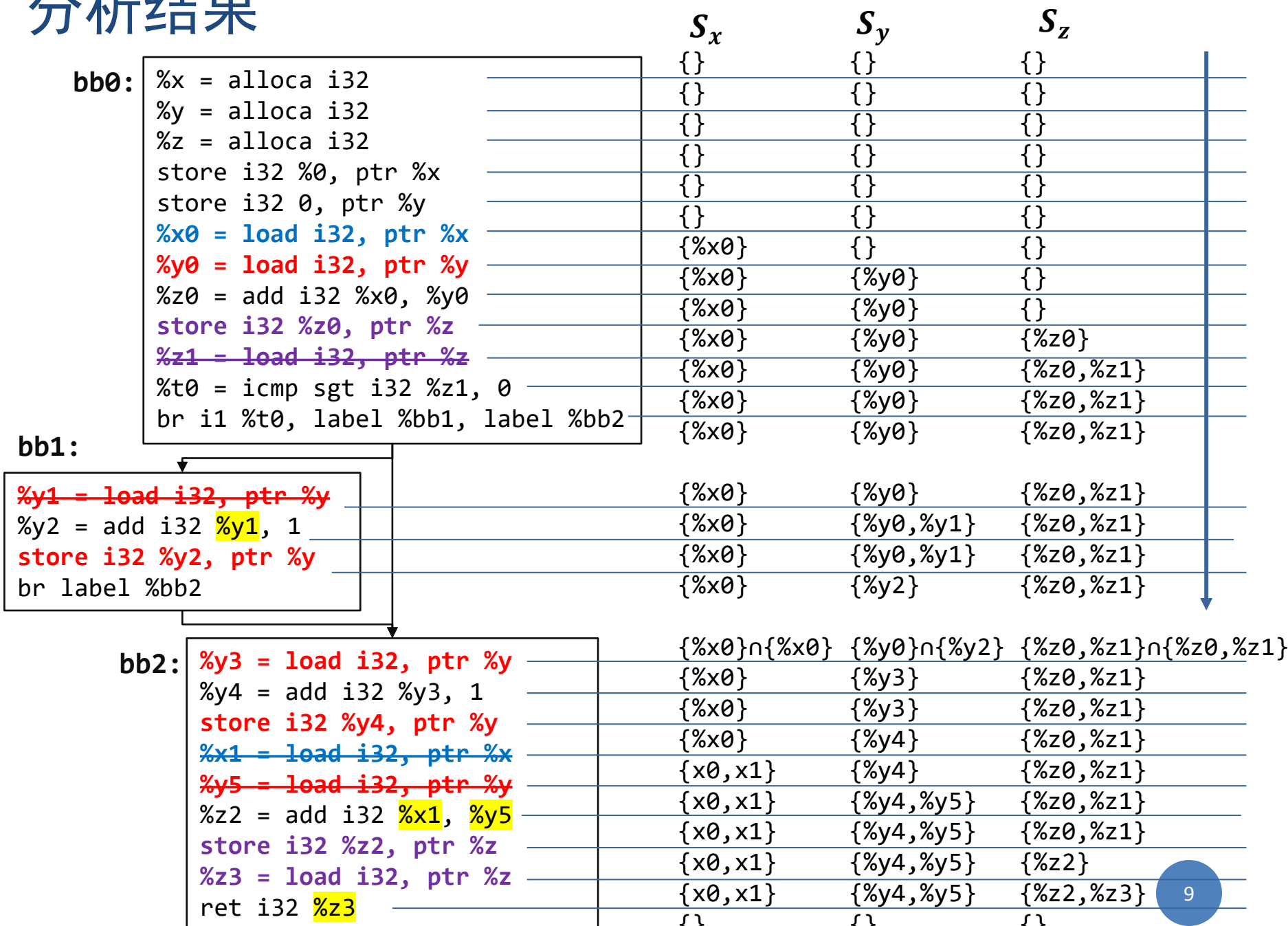
分析过程


S_x
S_y
bb0:
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 %0, ptr %x
store i32 0, ptr %y
%x0 = load i32, ptr %x
%y0 = load i32, ptr %y
%z0 = add i32 %x0, %y0
store i32 %z0, ptr %z
%z1 = load i32, ptr %z
%t0 = icmp sgt i32 %z1, 0
br i1 %t0, label %bb1, label %bb2
bb1:
%y1 = load i32, ptr %y
%y2 = add i32 %y1, 1
store i32 %y2, ptr %y
br label %bb2
bb2:
%y3 = load i32, ptr %y
%y4 = add i32 %y3, 1
store i32 %y4, ptr %y
%x1 = load i32, ptr %x
%y5 = load i32, ptr %y
%z2 = add i32 %x1, %y5
store i32 %z2, ptr %z
%z3 = load i32, ptr %z
ret i32 %z3
{} {}
{} {}
{} {}
{} {}
{} {}
{} {}
{%x0} {}
{%x0} {%y0}
{%x0} {%y0}
{%x0} {%y0}
{%x0} {%y0}
{%x0} {%y0}
{%x0} {%y0}
{%x0} {%y0}
{%x0} {%y0,%y1}
{%x0} {%y0,%y1}
{%x0} {%y2}
{%x0}∩{%x0} {%y0}∩{%y2}
{%x0} {%y3}
{%x0} {%y3}
{%x0} {%y4}
{x0,x1} {%y4}
{x0,x1} {%y4,%y5}
{x0,x1} {%y4,%y5}
{x0,x1} {%y4,%y5}
{x0,x1} {%y4,%y5}

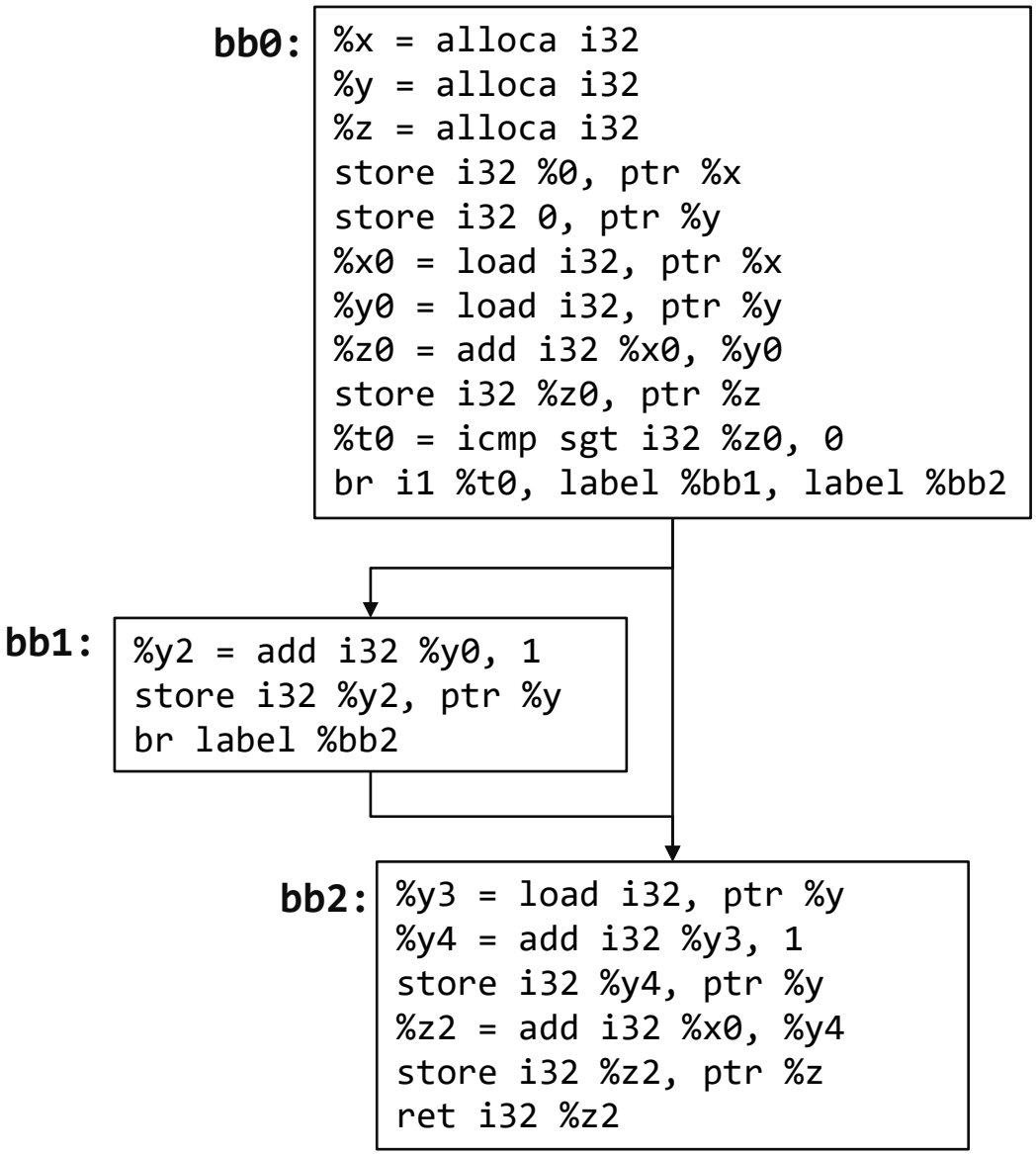
分析过程


S_x
S_y
S_z
bb0:
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 %0, ptr %x
store i32 0, ptr %y
%x0 = load i32, ptr %x
%y0 = load i32, ptr %y
%z0 = add i32 %x0, %y0
store i32 %z0, ptr %z
%z1 = load i32, ptr %z
%t0 = icmp sgt i32 %z1, 0
br i1 %t0, label %bb1, label %bb2
bb1:
%y1 = load i32, ptr %y
%y2 = add i32 %y1, 1
store i32 %y2, ptr %y
br label %bb2
bb2:
%y3 = load i32, ptr %y
%y4 = add i32 %y3, 1
store i32 %y4, ptr %y
%x1 = load i32, ptr %x
%y5 = load i32, ptr %y
%z2 = add i32 %x1, %y5
store i32 %z2, ptr %z
%z3 = load i32, ptr %z
ret i32 %z3
{} {} {}
{} {} {}
{} {} {}
{} {} {}
{} {} {}
{} {} {}
{%x0} {} {}
{%x0} {%y0} {}
{%x0} {%y0} {}
{%x0} {%y0} {%z0}
{%x0} {%y0} {%z0,%z1}
{%x0} {%y0} {%z0,%z1}
{%x0} {%y0} {%z0,%z1}
{%x0} {%y0} {%z0,%z1}
{%x0} {%y0,%y1} {%z0,%z1}
{%x0} {%y0,%y1} {%z0,%z1}
{%x0} {%y2} {%z0,%z1}
{%x0}∩{%x0} {%y0}∩{%y2} {%z0,%z1}∩{%z0,%z1}
{%x0} {%y3} {%z0,%z1}
{%x0} {%y3} {%z0,%z1}
{%x0} {%y4} {%z0,%z1}
{x0,x1} {%y4} {%z0,%z1}
{x0,x1} {%y4,%y5} {%z0,%z1}
{x0,x1} {%y4,%y5} {%z0,%z1}
{x0,x1} {%y4,%y5} {%z2}
{x0,x1} {%y4,%y5} {%z2,%z3}

分析结果


S_x S_y S_z
bb0:
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 %0, ptr %x
store i32 0, ptr %y
%x0 = load i32, ptr %x
%y0 = load i32, ptr %y
%z0 = add i32 %x0, %y0
store i32 %z0, ptr %z
%z1 = load i32, ptr %z
%t0 = icmp sgt i32 %z1, 0
br i1 %t0, label %bb1, label %bb2
bb1:
%y1 = load i32, ptr %y
%y2 = add i32 %y1, 1
store i32 %y2, ptr %y
br label %bb2
bb2:
%y3 = load i32, ptr %y
%y4 = add i32 %y3, 1
store i32 %y4, ptr %y
%x1 = load i32, ptr %x
%y5 = load i32, ptr %y
%z2 = add i32 %x1, %y5
store i32 %z2, ptr %z
%z3 = load i32, ptr %z
ret i32 %z3
{} {} {}
{} {} {}
{} {} {}
{} {} {}
{} {} {}
{} {} {}
{%x0} {} {}
{%x0} {%y0} {}
{%x0} {%y0} {}
{%x0} {%y0} {%z0}
{%x0} {%y0} {%z0,%z1}
{%x0} {%y0} {%z0,%z1}
{%x0} {%y0} {%z0,%z1}
{%x0} {%y0} {%z0,%z1}
{%x0} {%y0,%y1} {%z0,%z1}
{%x0} {%y0,%y1} {%z0,%z1}
{%x0} {%y2} {%z0,%z1}
{%x0}∩{%x0} {%y0}∩{%y2} {%z0,%z1}∩{%z0,%z1}
{%x0} {%y3} {%z0,%z1}
{%x0} {%y3} {%z0,%z1}
{%x0} {%y4} {%z0,%z1}
{x0,x1} {%y4} {%z0,%z1}
{x0,x1} {%y4,%y5} {%z0,%z1}
{x0,x1} {%y4,%y5} {%z0,%z1}
{x0,x1} {%y4,%y5} {%z2}
{x0,x1} {%y4,%y5} {%z2,%z3}

优化结果


bb0:
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 %0, ptr %x
store i32 0, ptr %y
%x0 = load i32, ptr %x
%y0 = load i32, ptr %y
%z0 = add i32 %x0, %y0
store i32 %z0, ptr %z
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
bb1:
%y2 = add i32 %y0, 1
store i32 %y2, ptr %y
br label %bb2
bb2:
%y3 = load i32, ptr %y
%y4 = add i32 %y3, 1
store i32 %y4, ptr %y
%z2 = add i32 %x0, %y4
store i32 %z2, ptr %z
ret i32 %z2

遇到循环：基于循环迭代的数据流分析

```
For (each instruction n):
    IN[n] = {<v: ∅>: v is a program variable}
    OUT[n] = {<v: ∅>}
Repeat:
    For(each instruction n):
        For(each n's predecessor p)
            IN[n] = IN[n] ∩ OUT[p]
        OUT[n] = TRANSFER(n)
Until IN[n] and OUT[n] stops changing for all n
```

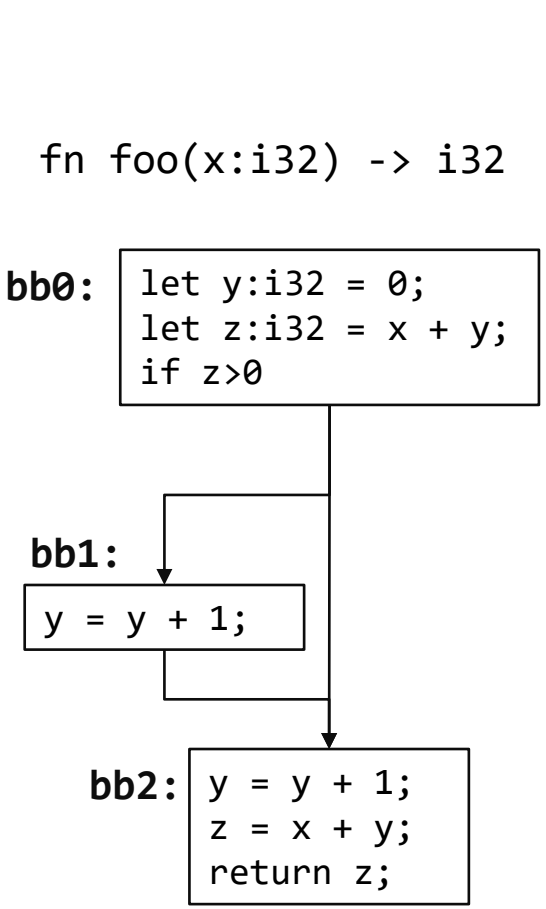
问题：算法是否一定会终止？

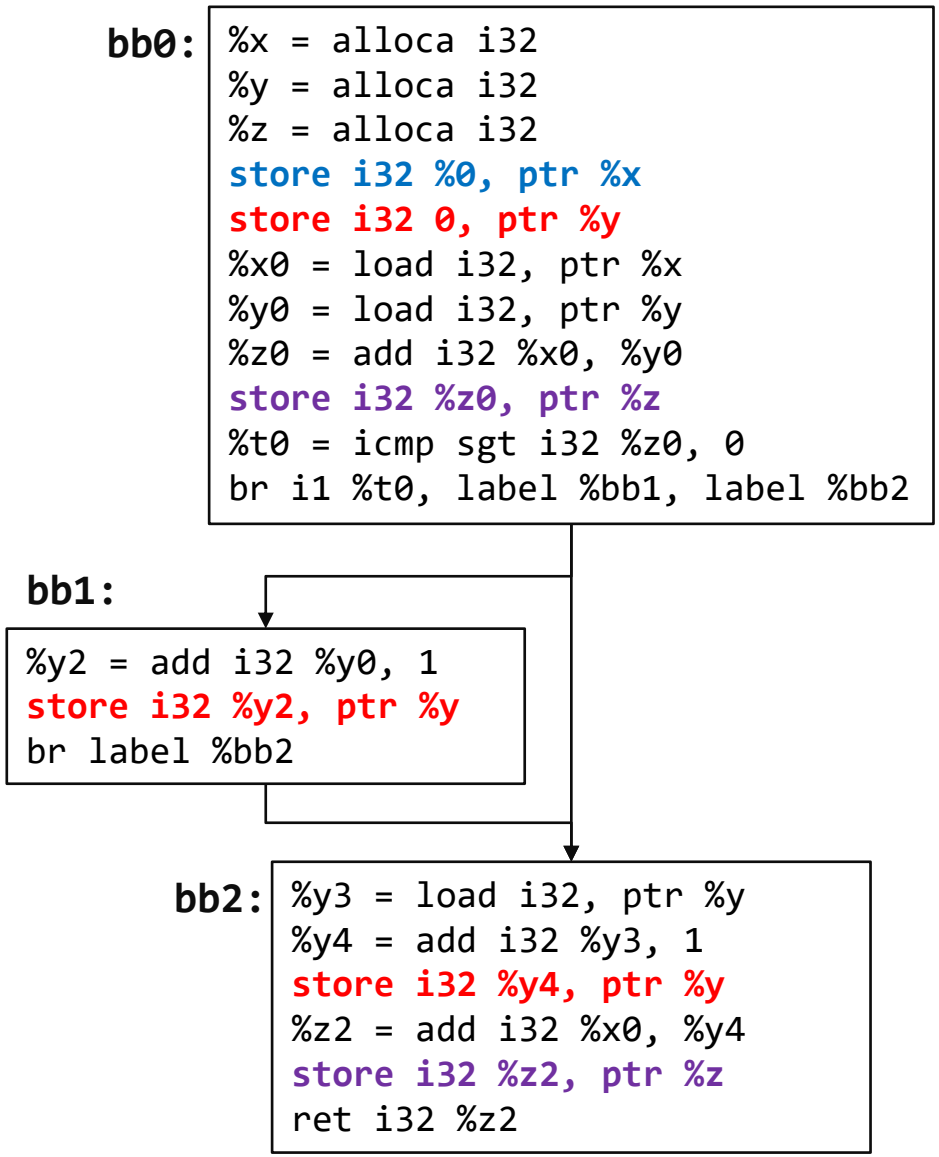
- 每个程序节点的可用寄存器数目单调递减

```
fn fac(n: i32) -> i32 {
    let r = 1;
    while n>0 {
        r = r * n;
        n = n-1;
    }
    return r;
}
```

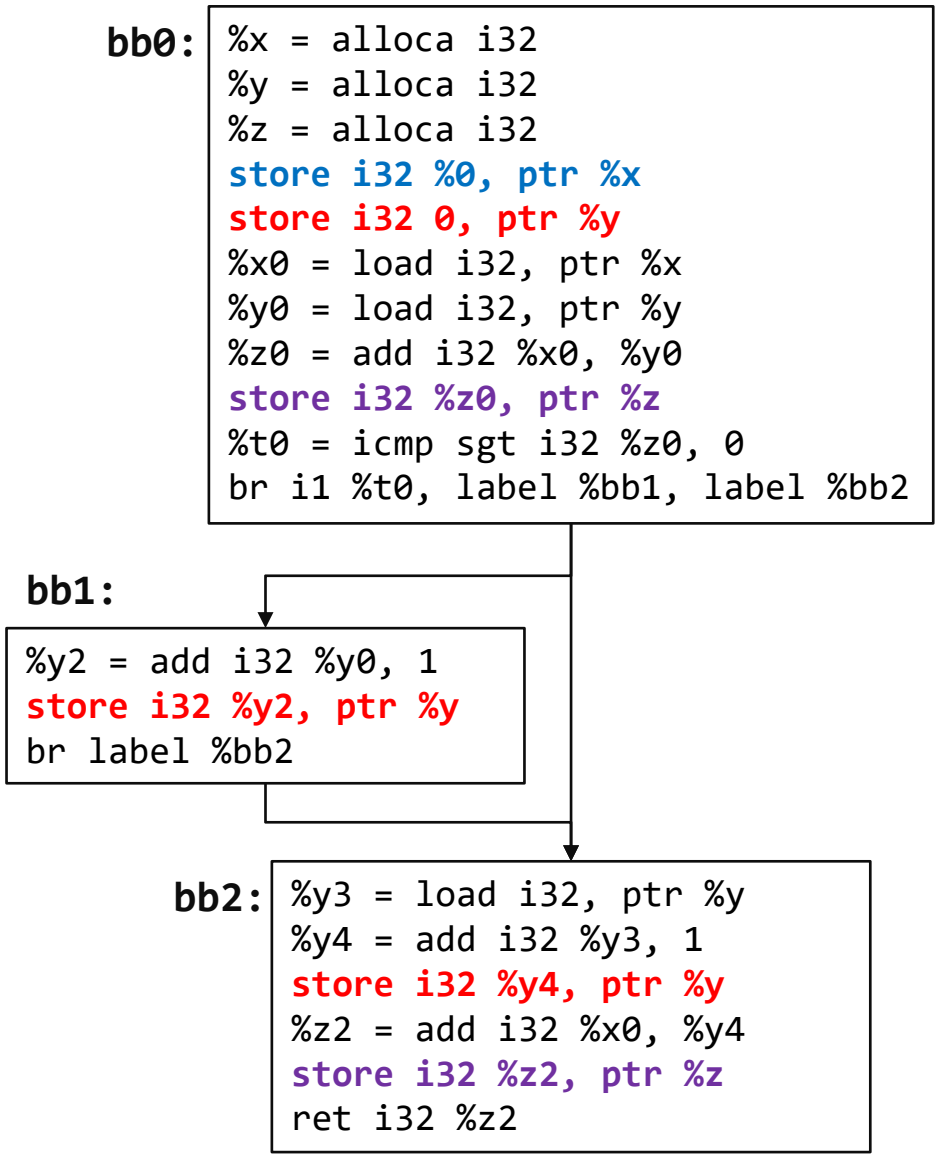
```
define i32 @fac(i32 %0) {
bb0:
    %n = alloca i32
    %r = alloca i32
    store i32 %0, ptr %n
    store i32 1, ptr %r
    br label %bb1
bb1:
    %t1 = load i32, ptr %n
    %t2 = icmp sgt i32 %t1, 0
    br i1 %t2, label %bb2, label %bb3
bb2:
    %t3 = load i32, ptr %r
    %t4 = load i32, ptr %n
    %t5 = mul i32 %t3, %t4
    store i32 %t5, ptr %r
    %t6 = load i32, ptr %n
    %t7 = sub i32 %t6, 1
    store i32 %t7, ptr %n
    br label %bb1
bb3:
    %t8 = load i32, ptr %r
    ret i32 %t8
}
```

线性IR中的store冗余





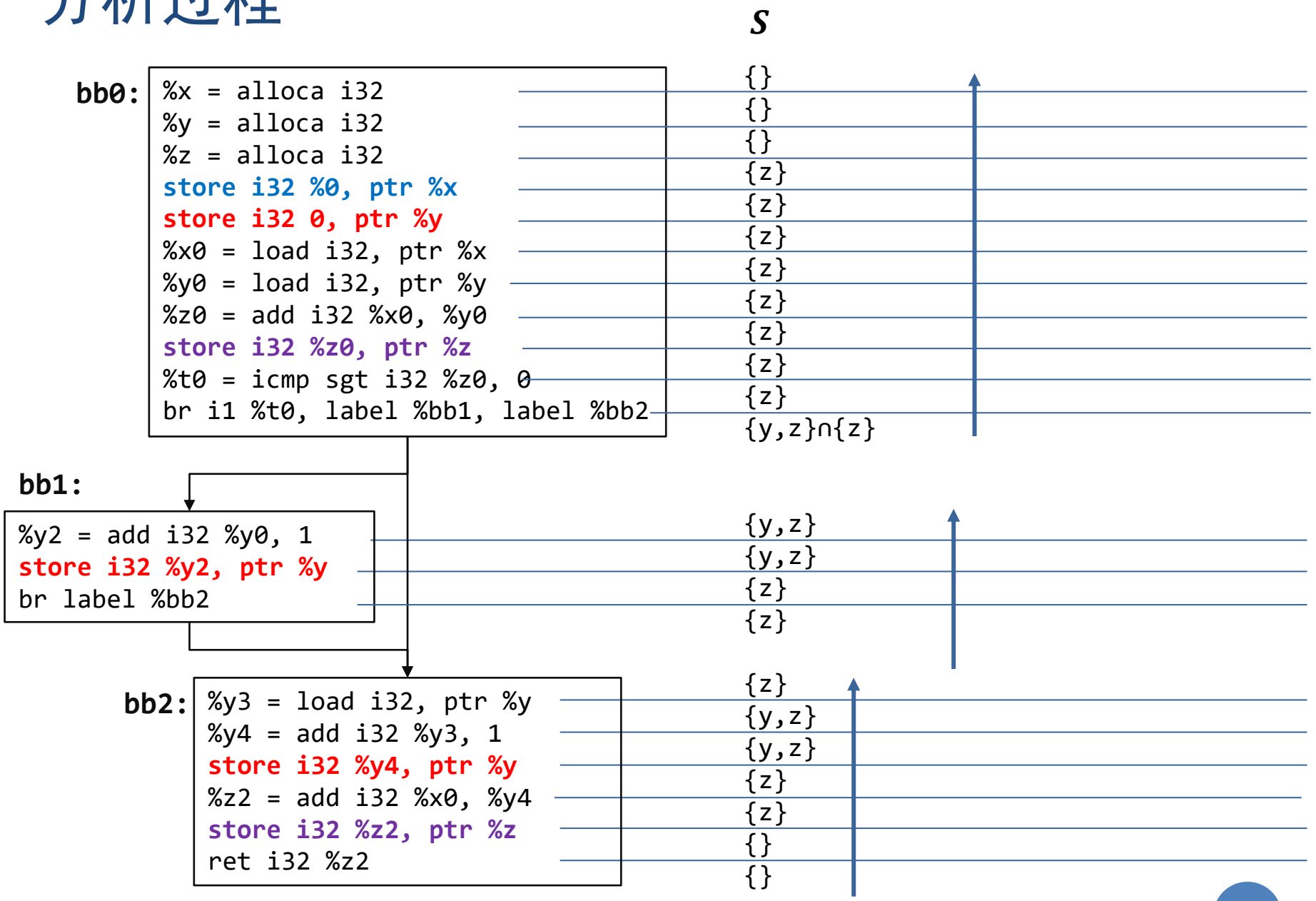
优化思路：可用store语句分析



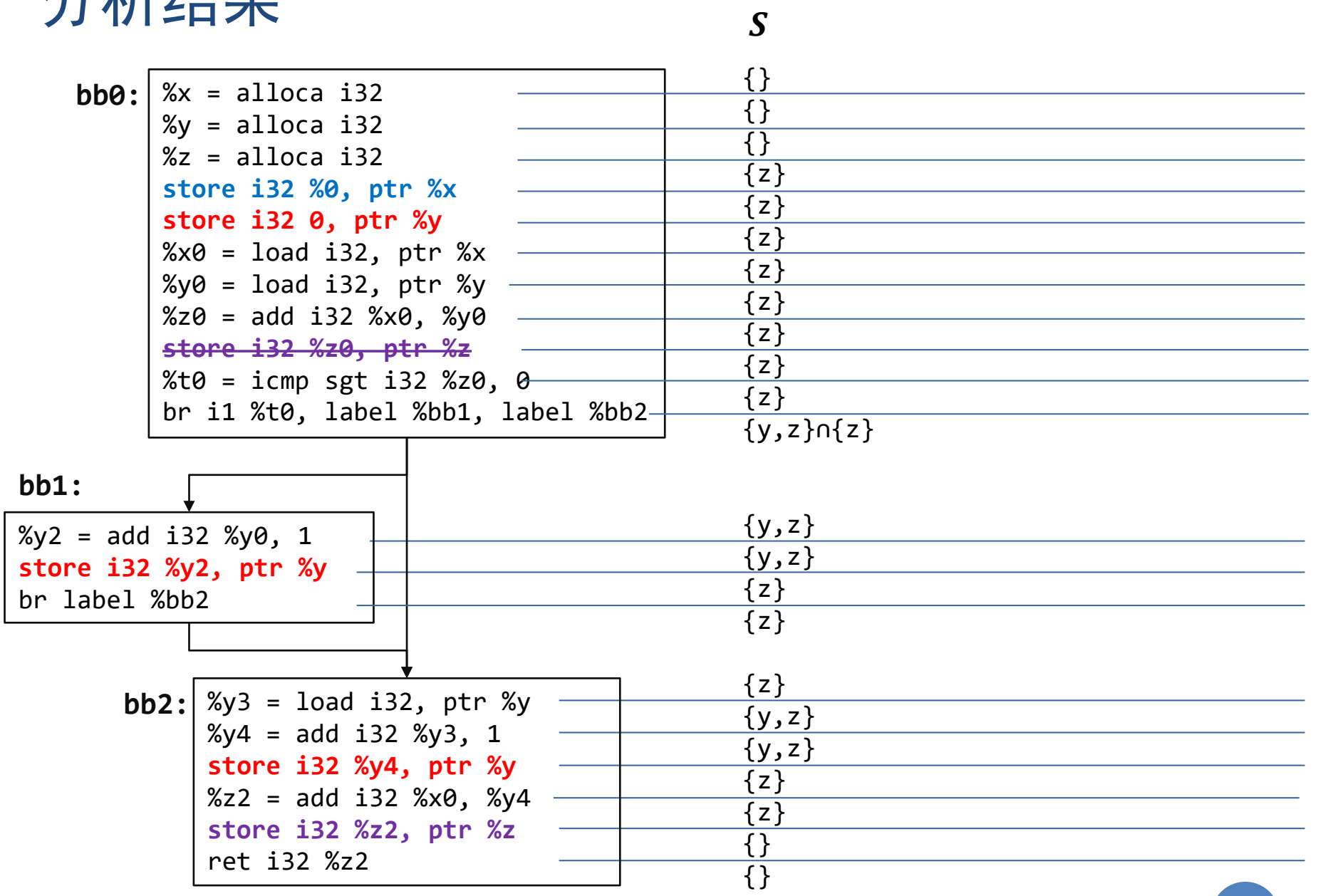
- 逆向遍历控制流图
- Transfer函数定义：
  - `store i32 %t, ptr %x`
    - $S = S \cup \{x\}$
  - `%t = load i32, ptr %x`
    - $S = S \backslash \{x\}$
  - `%x = alloca i32`
    - $S = S \backslash \{x\}$
- 遇到合并节点

$$\mathrm{OUT}(n) = \bigcap_{n' \in \mathrm{successor}(n)} \mathrm{IN}(n')$$

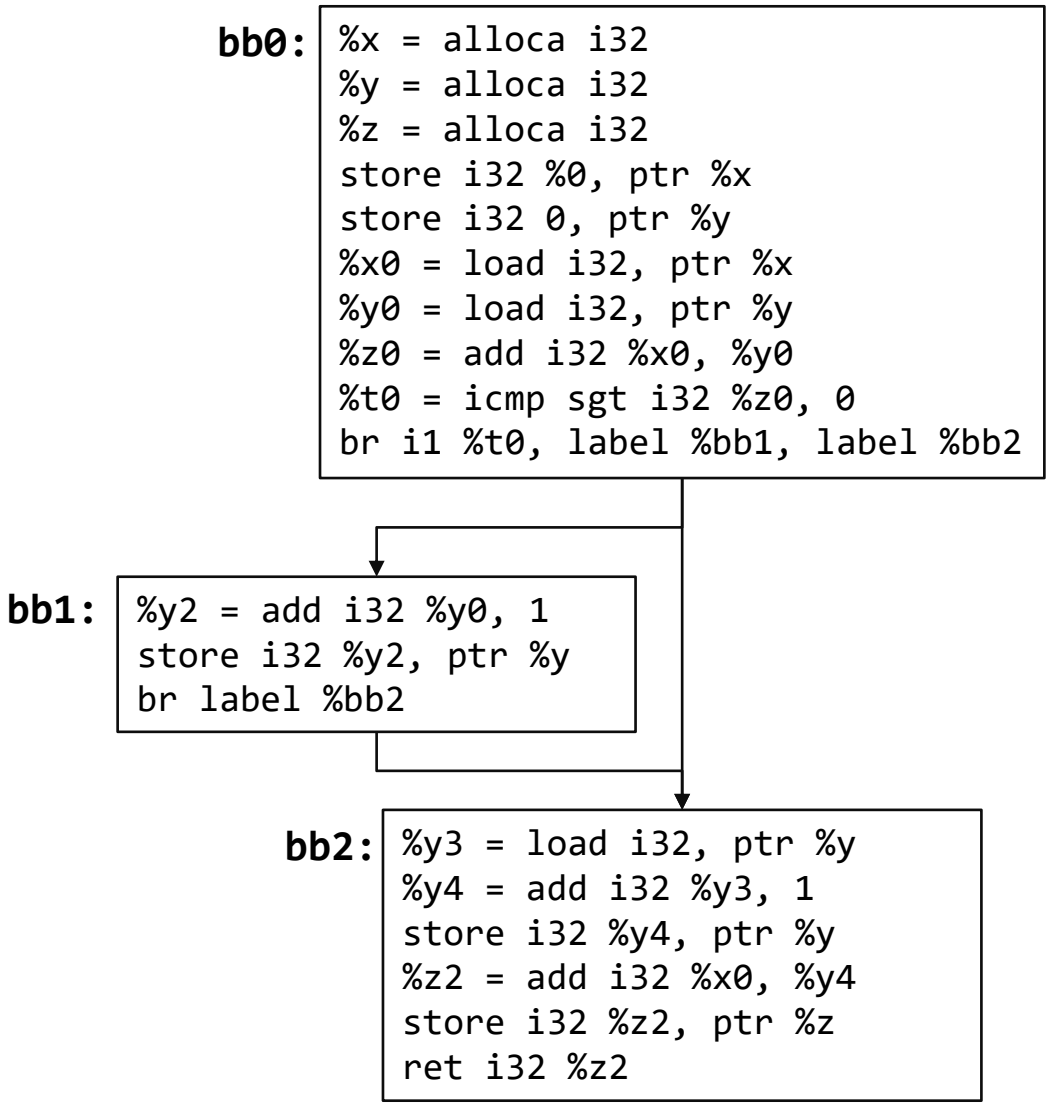
分析过程


S
bb0:
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 %0, ptr %x
store i32 0, ptr %y
%x0 = load i32, ptr %x
%y0 = load i32, ptr %y
%z0 = add i32 %x0, %y0
store i32 %z0, ptr %z
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
{}
{}
{}
{z}
{z}
{z}
{z}
{z}
{z}
{z}
{y,z}∩{z}
bb1:
%y2 = add i32 %y0, 1
store i32 %y2, ptr %y
br label %bb2
{y,z}
{y,z}
{z}
{z}
bb2:
%y3 = load i32, ptr %y
%y4 = add i32 %y3, 1
store i32 %y4, ptr %y
%z2 = add i32 %x0, %y4
store i32 %z2, ptr %z
ret i32 %z2
{z}
{y,z}
{y,z}
{z}
{z}
{}
{}

分析结果

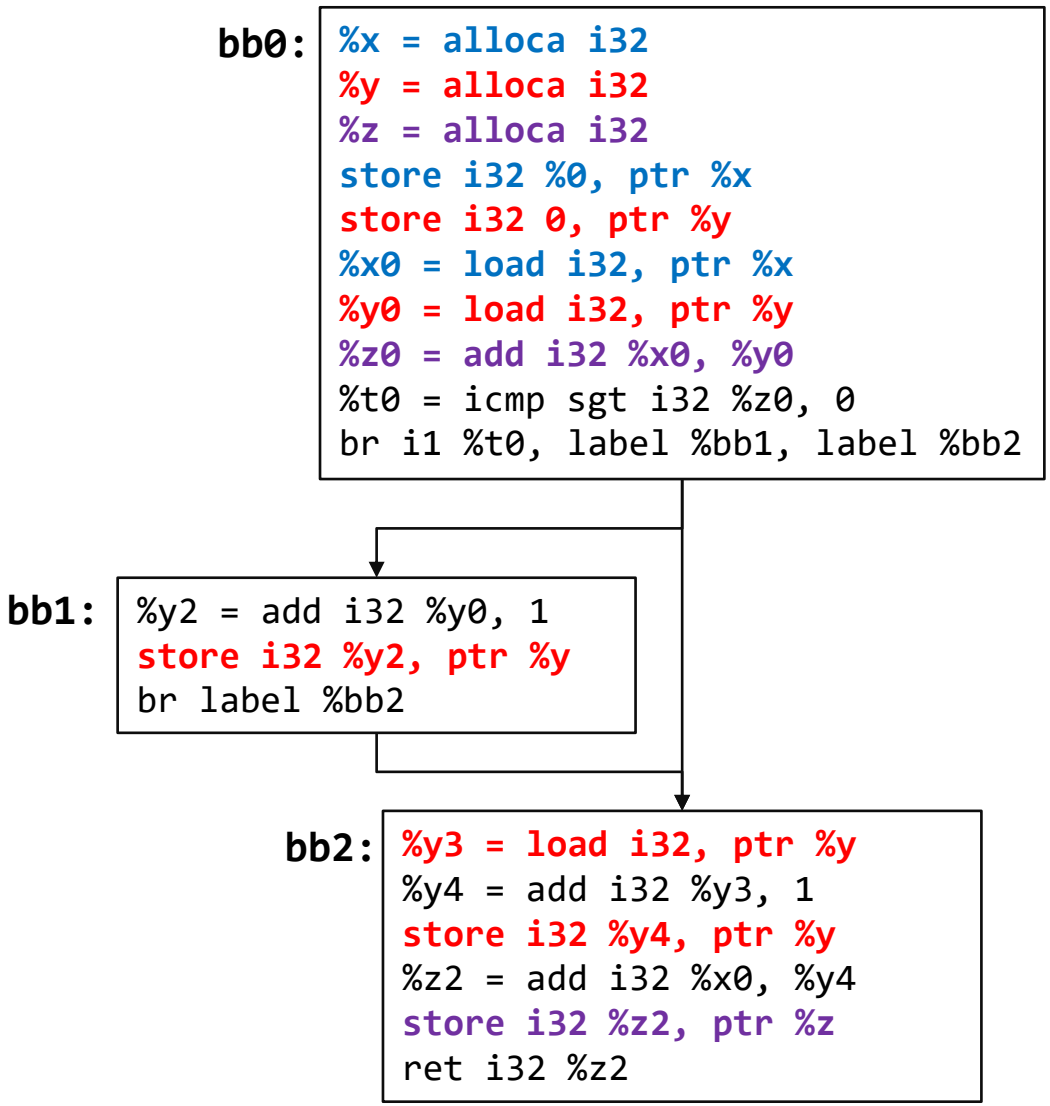

S
bb0:
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 %0, ptr %x
store i32 0, ptr %y
%x0 = load i32, ptr %x
%y0 = load i32, ptr %y
%z0 = add i32 %x0, %y0
store i32 %z0, ptr %z
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
{}
{}
{}
{z}
{z}
{z}
{z}
{z}
{z}
{z}
{y,z}∩{z}
bb1:
%y2 = add i32 %y0, 1
store i32 %y2, ptr %y
br label %bb2
{y,z}
{y,z}
{z}
{z}
bb2:
%y3 = load i32, ptr %y
%y4 = add i32 %y3, 1
store i32 %y4, ptr %y
%z2 = add i32 %x0, %y4
store i32 %z2, ptr %z
ret i32 %z2
{z}
{y,z}
{y,z}
{z}
{z}
{}
{}

优化结果

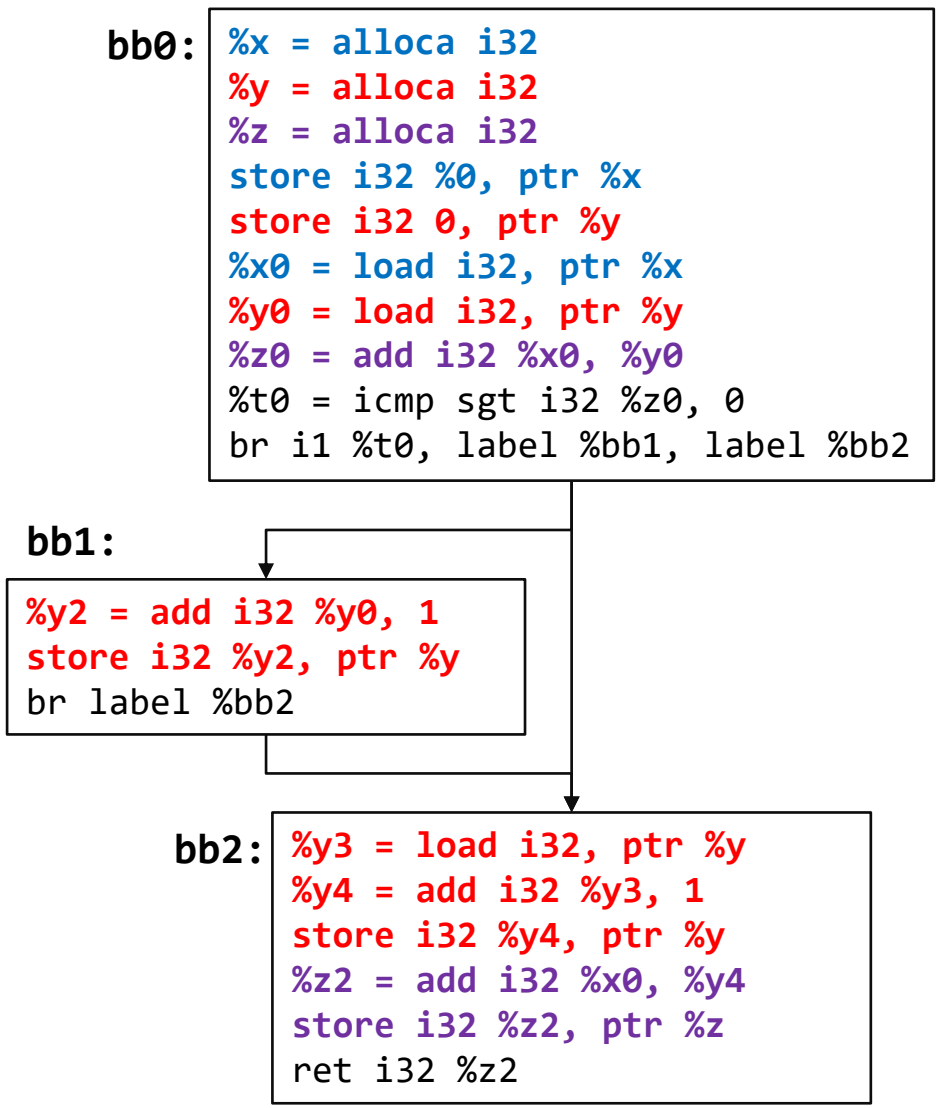

bb0:
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 %0, ptr %x
store i32 0, ptr %y
%x0 = load i32, ptr %x
%y0 = load i32, ptr %y
%z0 = add i32 %x0, %y0
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
bb1:
%y2 = add i32 %y0, 1
store i32 %y2, ptr %y
br label %bb2
bb2:
%y3 = load i32, ptr %y
%y4 = add i32 %y3, 1
store i32 %y4, ptr %y
%z2 = add i32 %x0, %y4
store i32 %z2, ptr %z
ret i32 %z2

二、纯寄存器表示

消除数据存取


bb0:
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 %0, ptr %x
store i32 0, ptr %y
%x0 = load i32, ptr %x
%y0 = load i32, ptr %y
%z0 = add i32 %x0, %y0
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
bb1:
%y2 = add i32 %y0, 1
store i32 %y2, ptr %y
br label %bb2
bb2:
%y3 = load i32, ptr %y
%y4 = add i32 %y3, 1
store i32 %y4, ptr %y
%z2 = add i32 %x0, %y4
store i32 %z2, ptr %z
ret i32 %z2

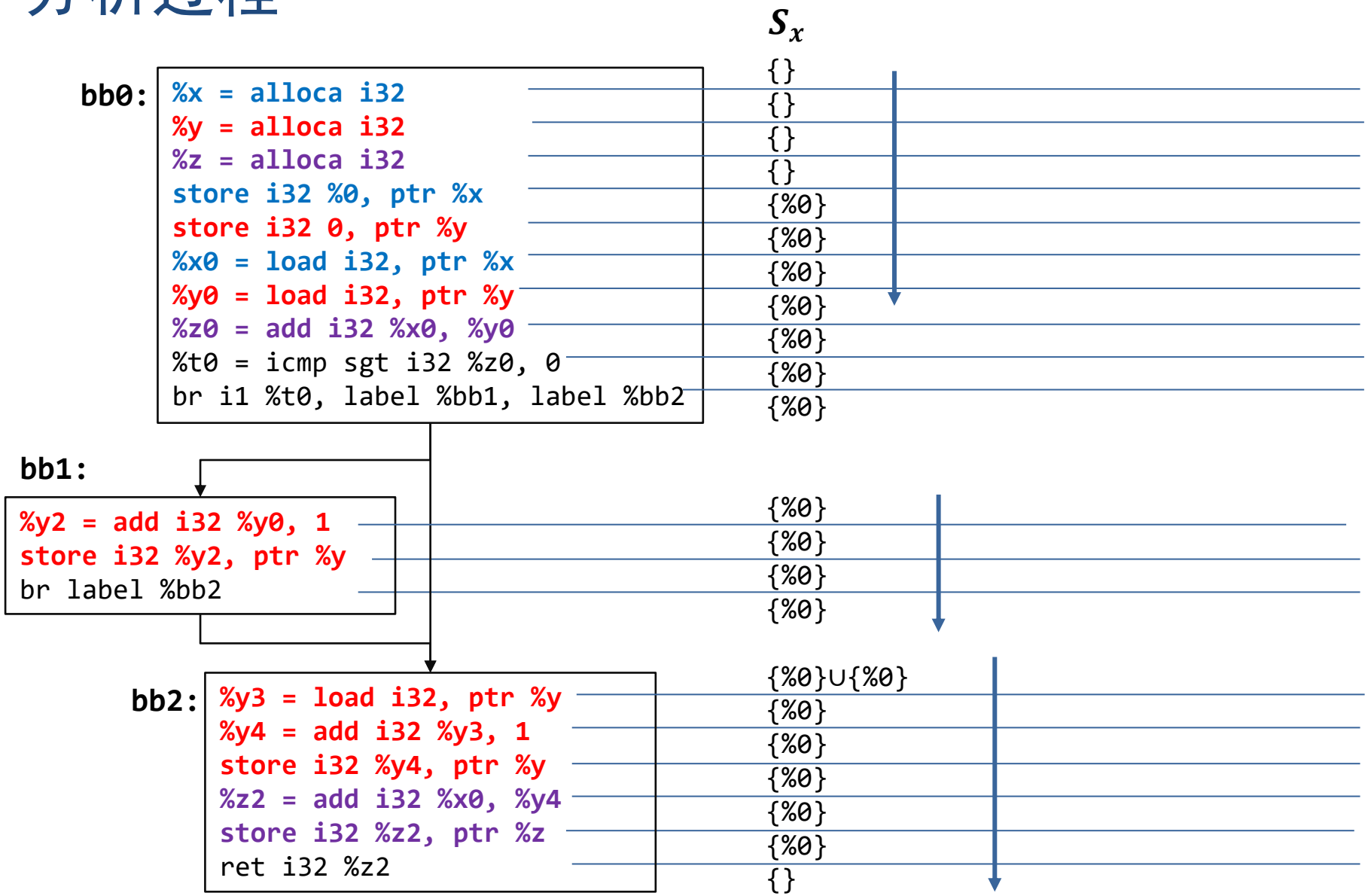
分析方法：内存数值流分析



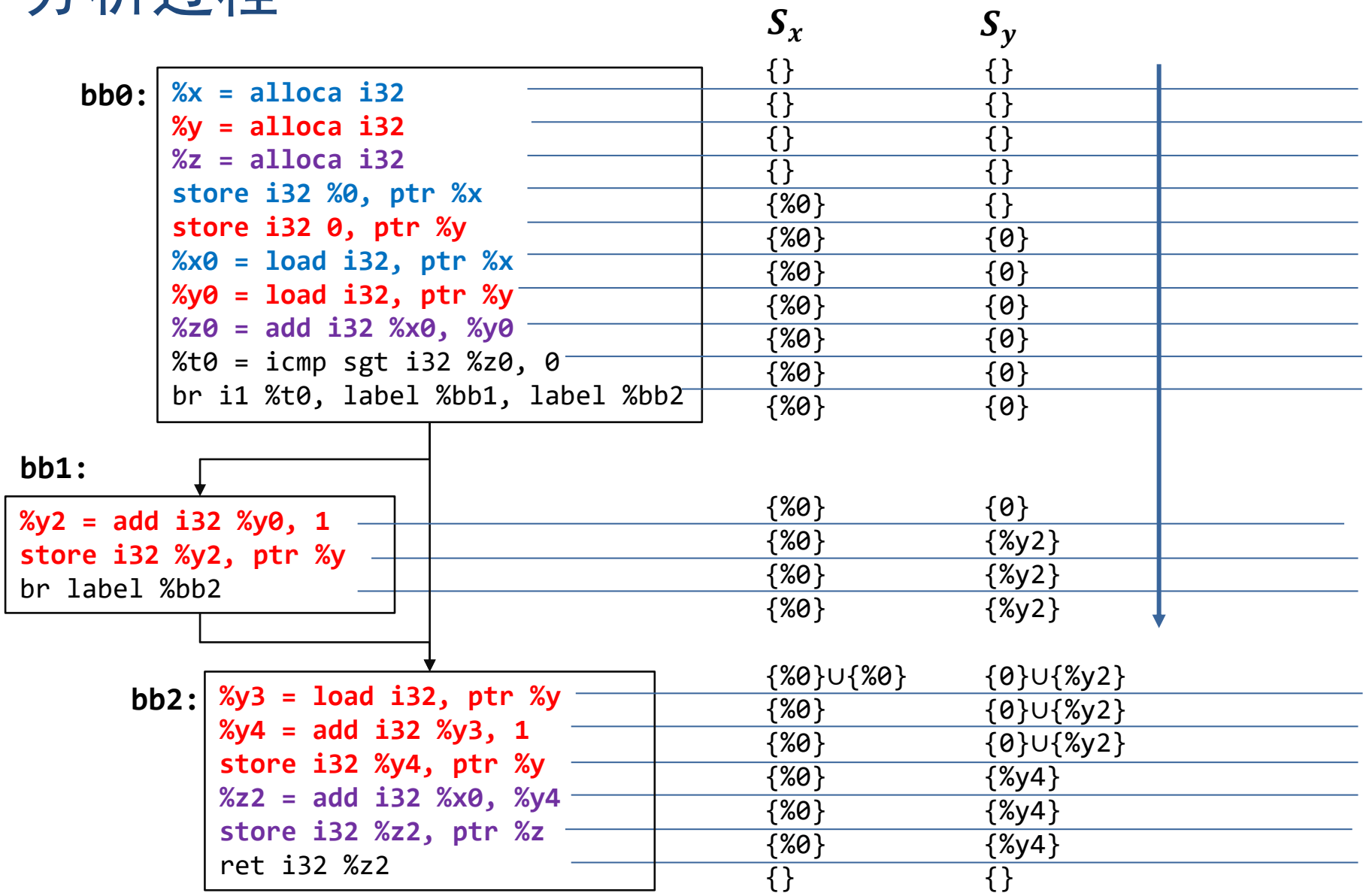
- 正向遍历控制流图
- Transfer函数定义：
  - store i32 %t, ptr %x
    - $S_x = \{t\}$
- 遇到合并节点

$$\mathrm{IN}(n) = \bigcup_{n' \in \mathrm{predecessor}(n)} \mathrm{OUT}(n')$$

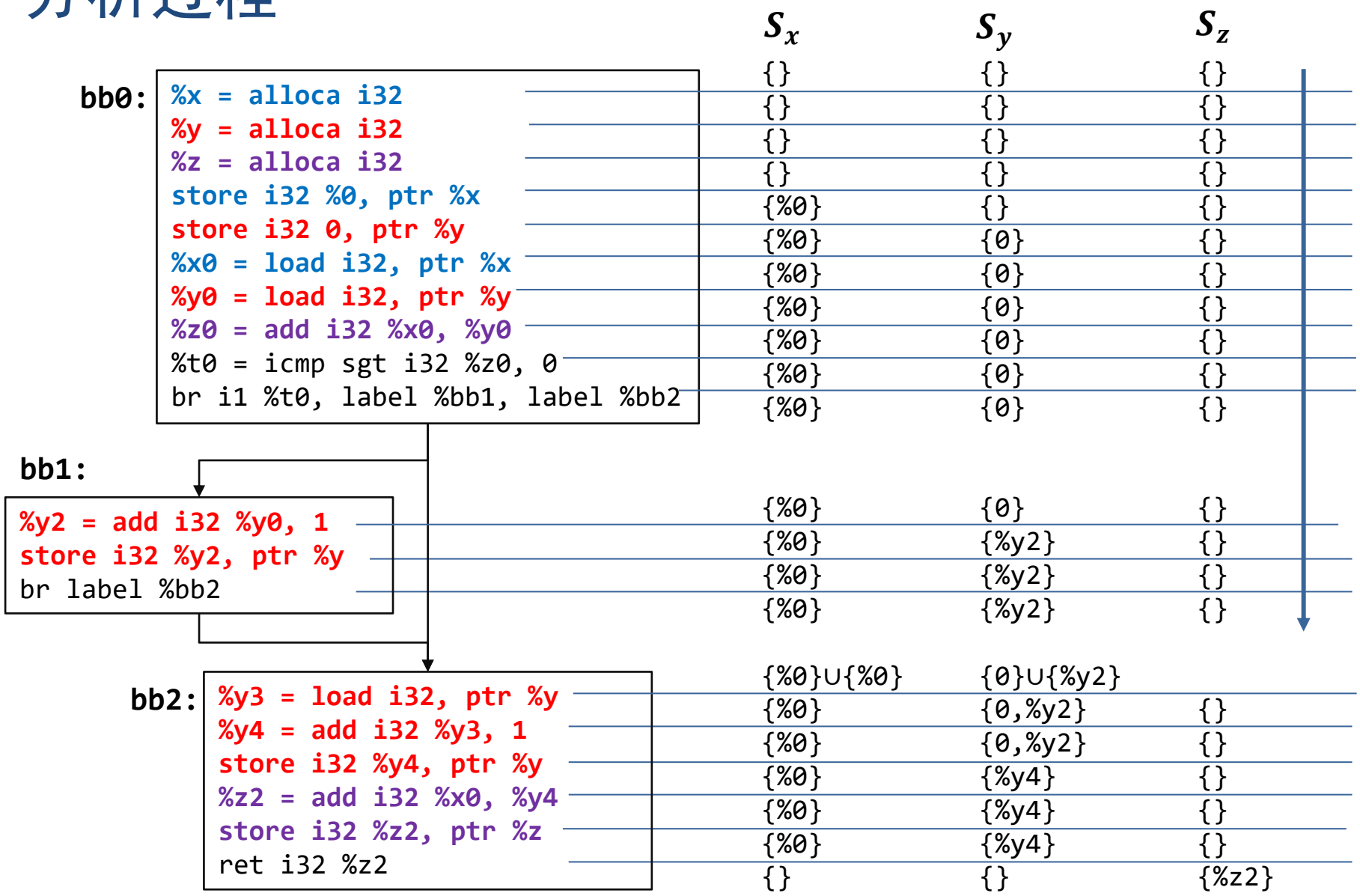
分析过程


S_x
bb0:
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 %0, ptr %x
store i32 0, ptr %y
%x0 = load i32, ptr %x
%y0 = load i32, ptr %y
%z0 = add i32 %x0, %y0
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
{}
{}
{}
{}
{%0}
{%0}
{%0}
{%0}
{%0}
{%0}
{%0}
bb1:
%y2 = add i32 %y0, 1
store i32 %y2, ptr %y
br label %bb2
{%0}
{%0}
{%0}
{%0}
bb2:
%y3 = load i32, ptr %y
%y4 = add i32 %y3, 1
store i32 %y4, ptr %y
%z2 = add i32 %x0, %y4
store i32 %z2, ptr %z
ret i32 %z2
{%0}∪{%0}
{%0}
{%0}
{%0}
{%0}
{%0}
{}

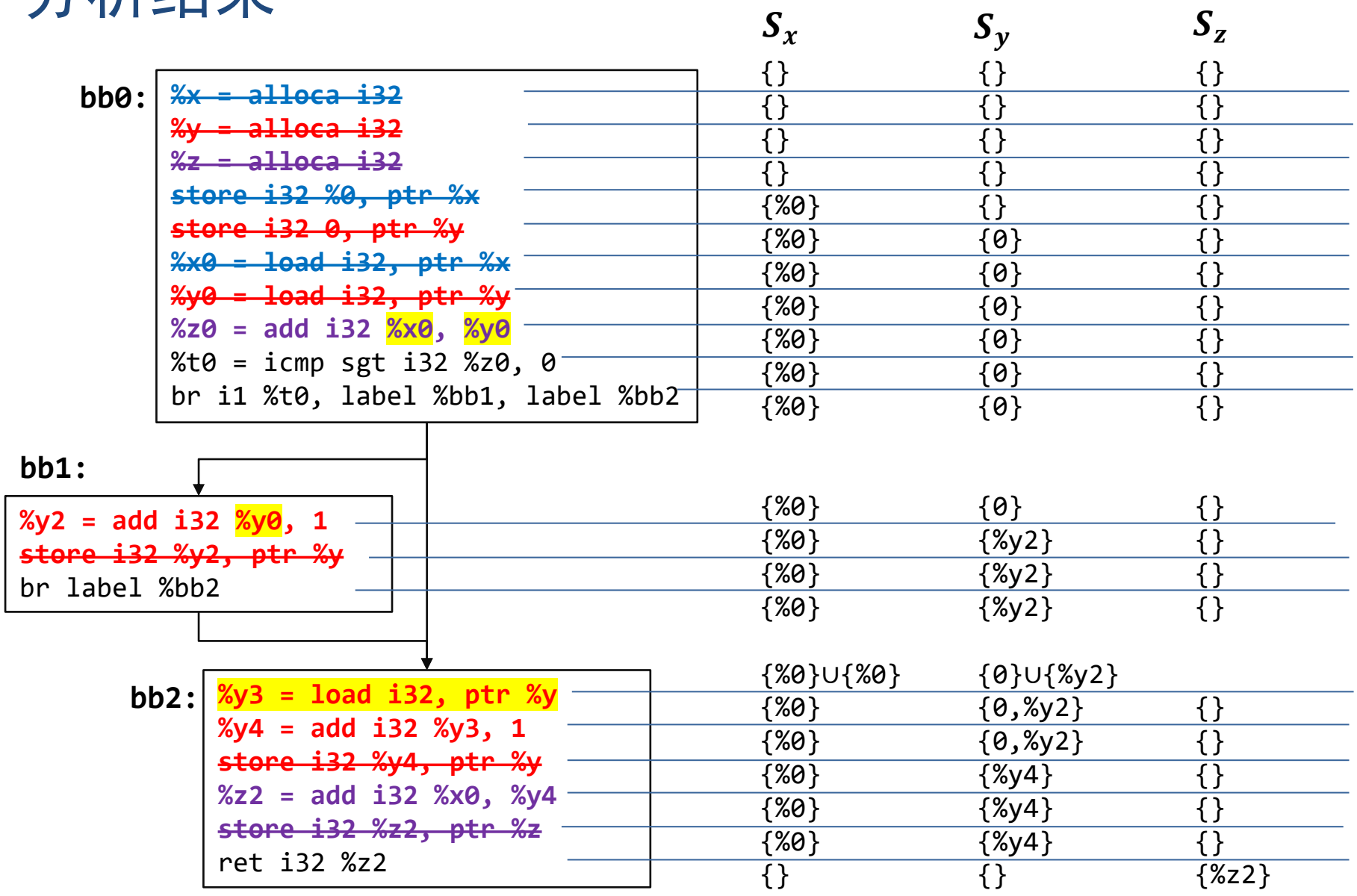
分析过程


S_x
S_y
bb0:
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 %0, ptr %x
store i32 0, ptr %y
%x0 = load i32, ptr %x
%y0 = load i32, ptr %y
%z0 = add i32 %x0, %y0
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
{} {}
{} {}
{} {}
{} {}
{%0} {}
{%0} {0}
{%0} {0}
{%0} {0}
{%0} {0}
{%0} {0}
{%0} {0}
bb1:
%y2 = add i32 %y0, 1
store i32 %y2, ptr %y
br label %bb2
{%0} {0}
{%0} {%y2}
{%0} {%y2}
{%0} {%y2}
bb2:
%y3 = load i32, ptr %y
%y4 = add i32 %y3, 1
store i32 %y4, ptr %y
%z2 = add i32 %x0, %y4
store i32 %z2, ptr %z
ret i32 %z2
{%0}∪{%0} {0}∪{%y2}
{%0} {0}∪{%y2}
{%0} {0}∪{%y2}
{%0} {%y4}
{%0} {%y4}
{%0} {%y4}
{} {}

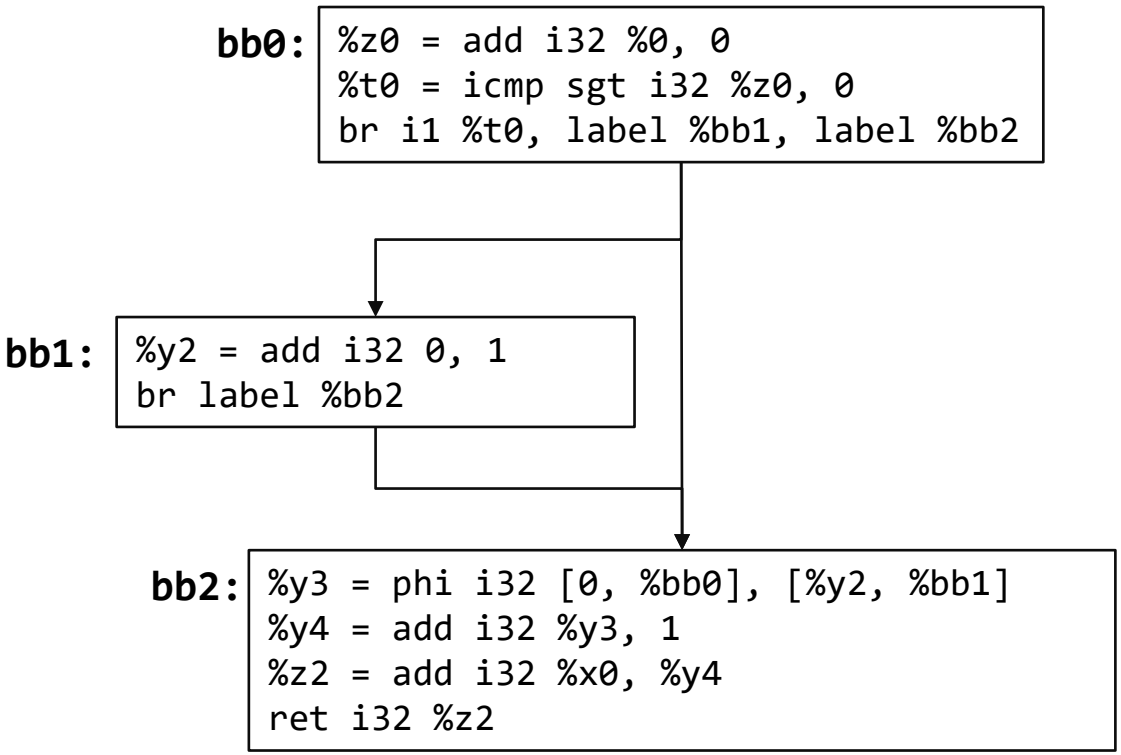
分析过程


S_x
S_y
S_z
bb0:
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 %0, ptr %x
store i32 0, ptr %y
%x0 = load i32, ptr %x
%y0 = load i32, ptr %y
%z0 = add i32 %x0, %y0
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
{} {} {}
{} {} {}
{} {} {}
{} {} {}
{%0} {} {}
{%0} {0} {}
{%0} {0} {}
{%0} {0} {}
{%0} {0} {}
{%0} {0} {}
{%0} {0} {}
bb1:
%y2 = add i32 %y0, 1
store i32 %y2, ptr %y
br label %bb2
{%0} {0} {}
{%0} {%y2} {}
{%0} {%y2} {}
{%0} {%y2} {}
bb2:
%y3 = load i32, ptr %y
%y4 = add i32 %y3, 1
store i32 %y4, ptr %y
%z2 = add i32 %x0, %y4
store i32 %z2, ptr %z
ret i32 %z2
{%0}∪{%0} {0}∪{%y2}
{%0} {0,%y2} {}
{%0} {0,%y2} {}
{%0} {%y4} {}
{%0} {%y4} {}
{%0} {%y4} {}
{} {} {%z2}

分析结果


S_x
S_y
S_z
bb0:
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 %0, ptr %x
store i32 0, ptr %y
%x0 = load i32, ptr %x
%y0 = load i32, ptr %y
%z0 = add i32 %x0, %y0
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
{} {} {}
{} {} {}
{} {} {}
{} {} {}
{%0} {} {}
{%0} {0} {}
{%0} {0} {}
{%0} {0} {}
{%0} {0} {}
{%0} {0} {}
{%0} {0} {}
bb1:
%y2 = add i32 %y0, 1
store i32 %y2, ptr %y
br label %bb2
{%0} {0} {}
{%0} {%y2} {}
{%0} {%y2} {}
{%0} {%y2} {}
bb2:
%y3 = load i32, ptr %y
%y4 = add i32 %y3, 1
store i32 %y4, ptr %y
%z2 = add i32 %x0, %y4
store i32 %z2, ptr %z
ret i32 %z2
{%0}∪{%0} {0}∪{%y2}
{%0} {0,%y2} {}
{%0} {0,%y2} {}
{%0} {%y4} {}
{%0} {%y4} {}
{%0} {%y4} {}
{} {} {%z2}

纯寄存器表示


bb0:
%z0 = add i32 %0, 0
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
bb1:
%y2 = add i32 0, 1
br label %bb2
bb2:
%y3 = phi i32 [0, %bb0], [%y2, %bb1]
%y4 = add i32 %y3, 1
%z2 = add i32 %x0, %y4
ret i32 %z2

练习：将下列代码转化为纯寄存器表示

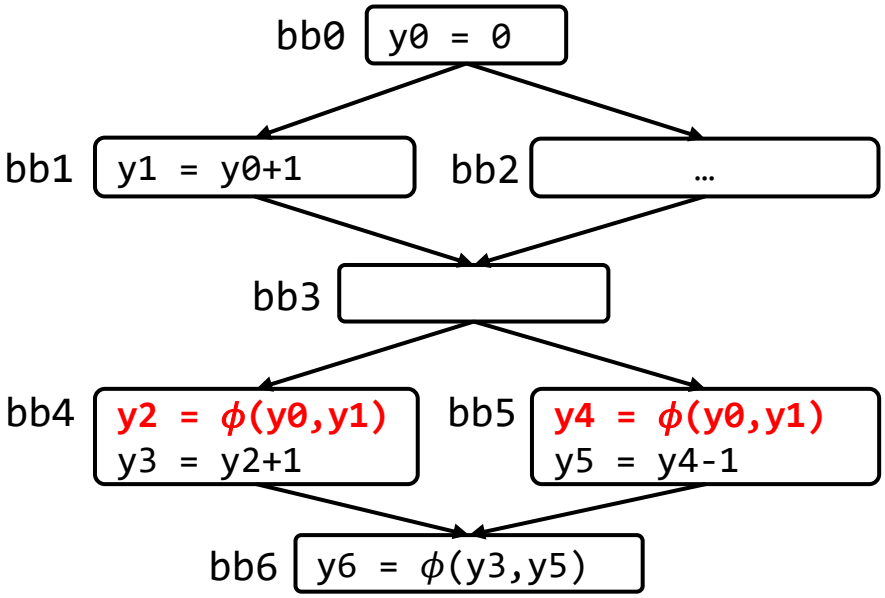
```
define i32 @fac(i32 %0) {
bb0:
    %n = alloca i32
    %r = alloca i32
    store i32 %0, ptr %n
    store i32 1, ptr %r
    br label %bb1
bb1:
    %t1 = load i32, ptr %n
    %t2 = icmp sgt i32 %t1, 0
    br i1 %t2, label %bb2, label %bb3
bb2:
    %t3 = load i32, ptr %r
    %t4 = load i32, ptr %n
    %t5 = mul i32 %t3, %t4
    store i32 %t5, ptr %r
    %t6 = load i32, ptr %n
    %t7 = sub i32 %t6, 1
    store i32 %t7, ptr %n
    br label %bb1
bb3:
    %t8 = load i32, ptr %r
    ret i32 %t8
}
```

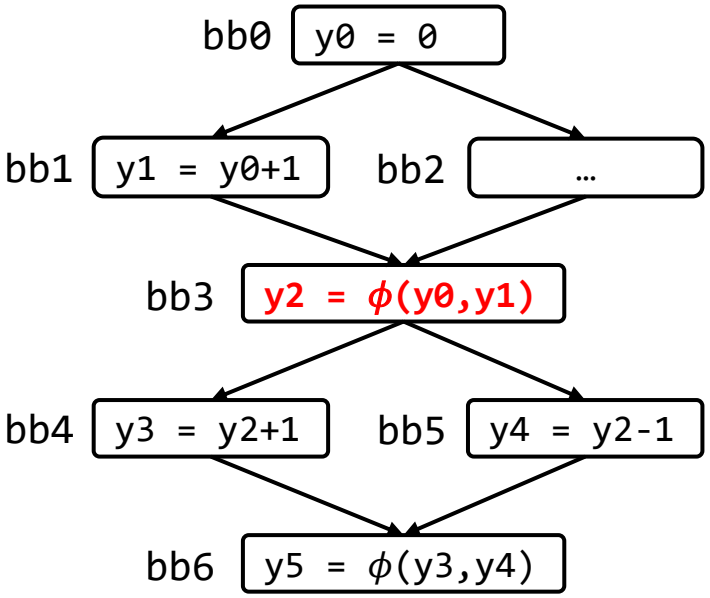
数据流分析方法小结

	May Analysis (U)	Must Analysis (∩)
前向分析	纯寄存器表示	精简load
逆向分析	活跃性分析	精简store

三、Phi指令优化

哪个phi指令方案更优？

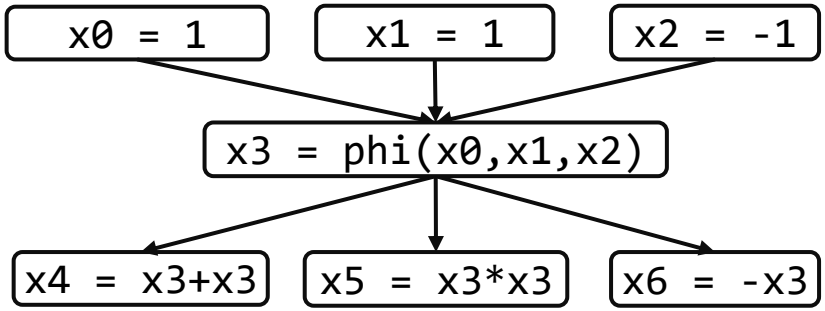




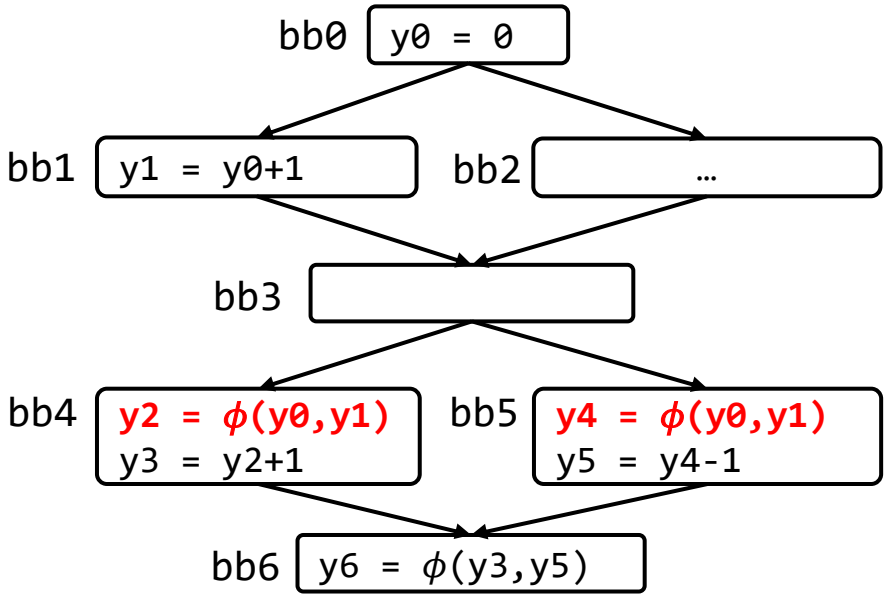
SSA简化def-use关系

- 原始程序的def-use关系数量是$O(m \times n)$
- SSA的def-use数量减少为$O(m + n)$

```
match v1:
    0 => { x = 0; }
    1 => { x = 1; }
    _ => { x = -1; }
...
match v2:
    0 => { x = x + x; }
    1 => { x = x * x; }
    _ => { x = -x; }
```

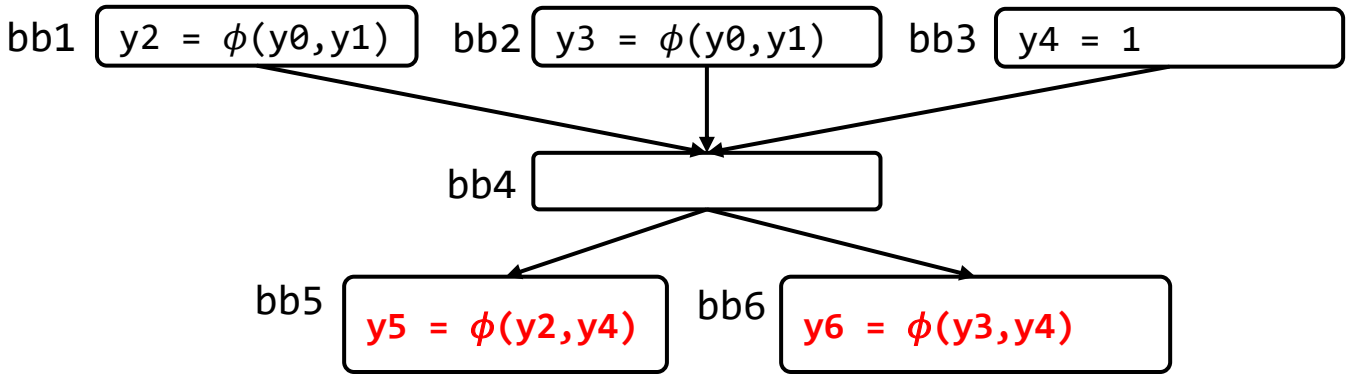


逆向分析：提取公共Phi表达式

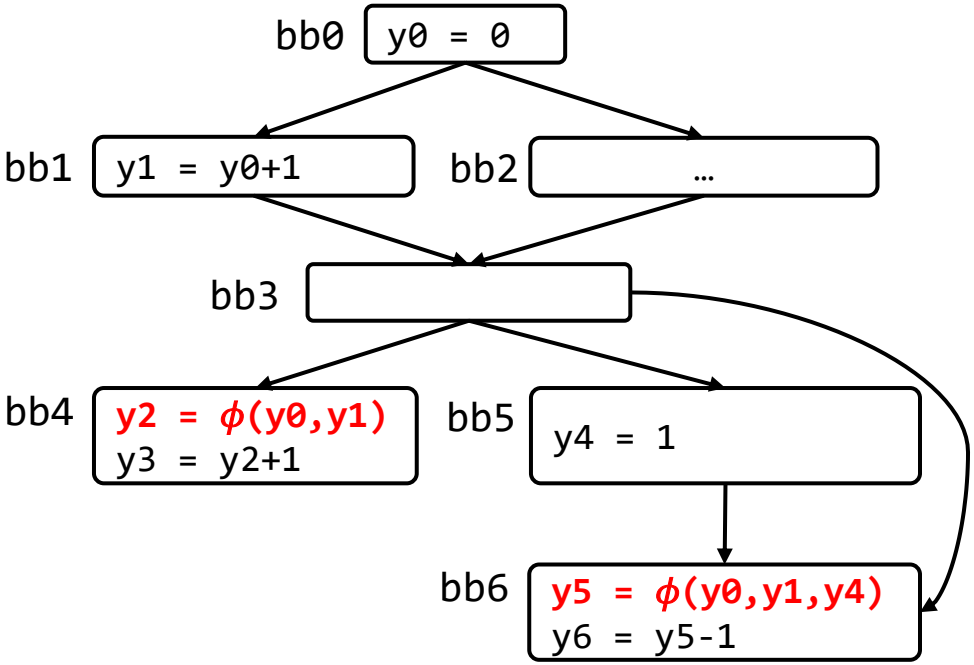


如何设计？
潜在问题？

潜在问题

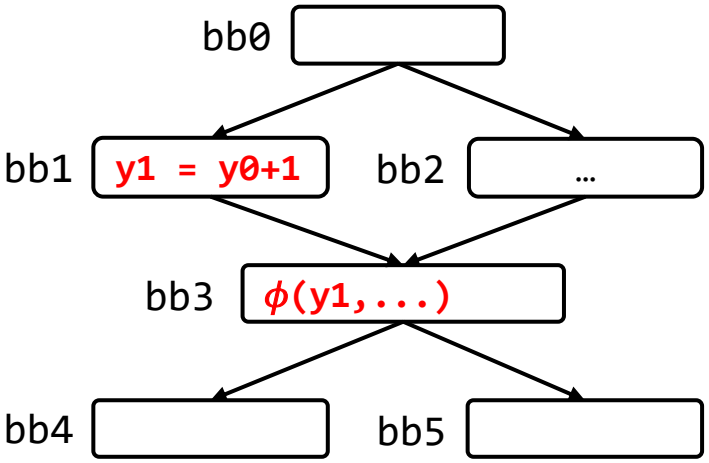


潜在问题


bb0
y0 = 0
bb1
y1 = y0+1
bb2
…
bb3
bb4
y2 = ϕ(y0,y1)
y3 = y2+1
bb5
y4 = 1
bb6
y5 = ϕ(y0,y1,y4)
y6 = y5-1

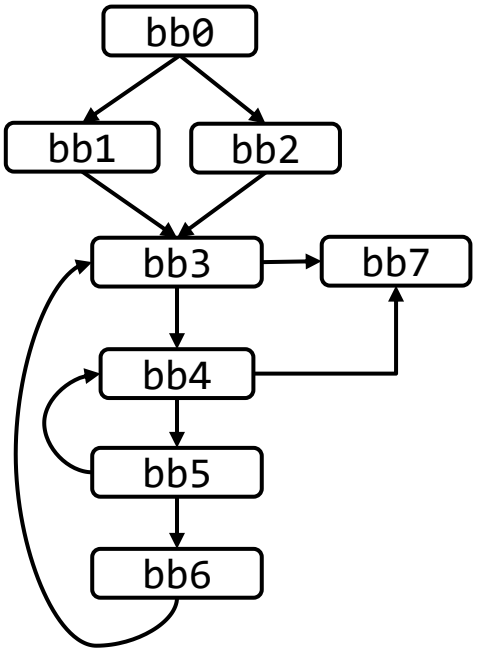
基于支配边界的Phi指令优化方法

- 正向分析：根据赋值确定后续Phi指令插入点。
  - 基于支配边界确定



支配的基本概念

- 给定有向图$G(V,E)$与起点v_0，如果从v_0到某个点v_j均需要经过点v_i，则称v_i支配v_j或v_i是v_j 的一个支配点
  - $v_i \in Dom(v_j)$
- 如果$v_i \neq v_j$，则称v_i严格支配v_j，$v_i \in IDom(v_j)$



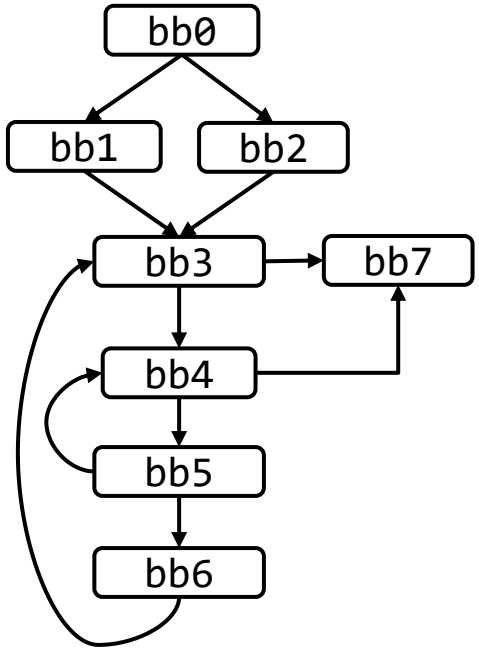
控制流图

$$Dom(bb_0) = \{bb_0\}$$
$$Dom(bb_1) = \{bb_0, bb_1\}$$
$$Dom(bb_2) = \{bb_0, bb_2\}$$
$$Dom(bb_3) = \{bb_0, bb_3\}$$
$$Dom(bb_4) = \{bb_0, bb3, bb_4\}$$
$$Dom(bb_5) = \{bb_0, bb3, bb_4, bb_5\}$$
$$Dom(bb_6) = \{bb_0, bb3, bb_4, bb5, bb_6\}$$
$$Dom(bb_7) = \{bb_0, bb_3, bb_7\}$$

$$Dom(v) = \begin{cases} \{v\}, & if\ v = v_0 \\ \{v\} \cup \left(\bigcap_{p \in pred(v)} Dom(p) \right), & if\ v \neq v_0 \end{cases}$$

支配边界

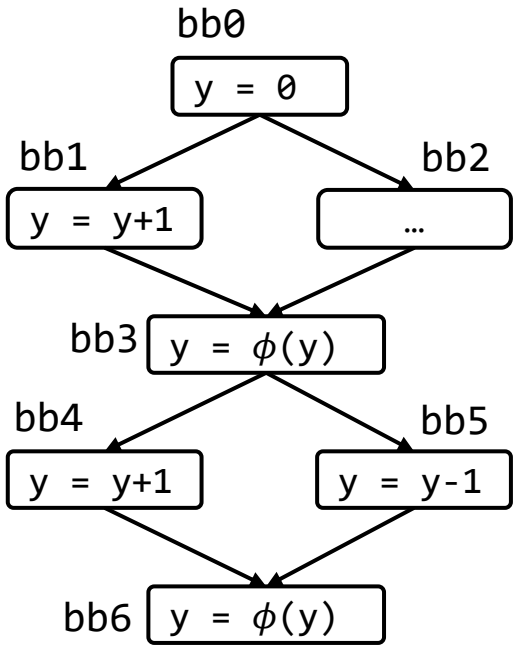
- v_i的支配边界是所有满足条件的v_j的集合
  - v_i支配v_j的一个前序节点
  - v_i并不严格支配v_j



$$DF(bb_0) = \{\}$$
$$DF(bb_1) = \{bb_3\}$$
$$DF(bb_2) = \{bb_3\}$$
$$DF(bb_3) = \{bb_3\}$$
$$DF(bb_4) = \{bb_3, bb_4, bb_7\}$$
$$DF(bb_5) = \{bb_3, bb_4\}$$
$$DF(bb_6) = \{bb_3\}$$
$$DF(bb_7) = \{\}$$

优化思路：基于支配边界优化Phi指令

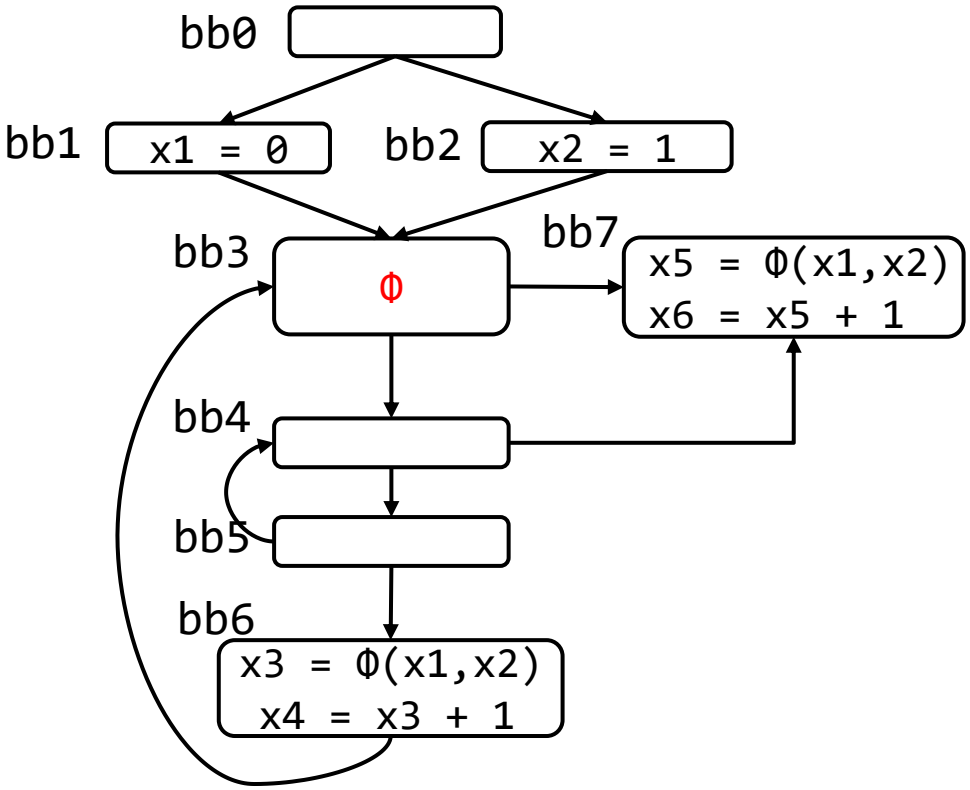
- bb0支配bb2，bb1和bb2的支配边界都是bb3
- 如果bb1和bb2中都没有def(x)，bb3不需要phi(x)，可直接使用bb0中的def(x)
- 如果bb1中有def(y)，bb3中一定需要phi(y)



利用支配边界设置phi指令

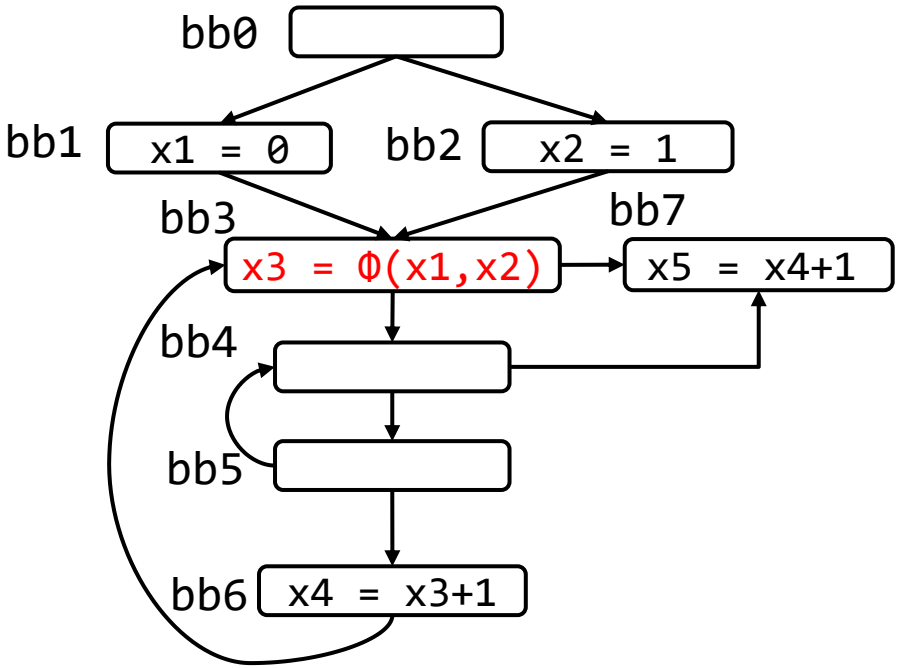
- 初始化：枚举所有变量的def-sites
  - def-sites(x) = {bb1,bb2,bb6,bb7}
- 为每个变量在bb_j增加phi节点：
  - $bb_i \in$ def-sites(x)
  - $bb_j \in DF(bb_i)$
- 在bb3增加phi指令的phi(x)

$$DF(bb_0) = \{\}$$
$$DF(bb_1) = \{bb_3\}$$
$$DF(bb_2) = \{bb_3\}$$
$$DF(bb_3) = \{bb_3\}$$
$$DF(bb_4) = \{bb3, bb_4, bb_7\}$$
$$DF(bb_5) = \{bb3, bb_4\}$$
$$DF(bb_6) = \{bb_3\}$$
$$DF(bb_7) = \{\}$$

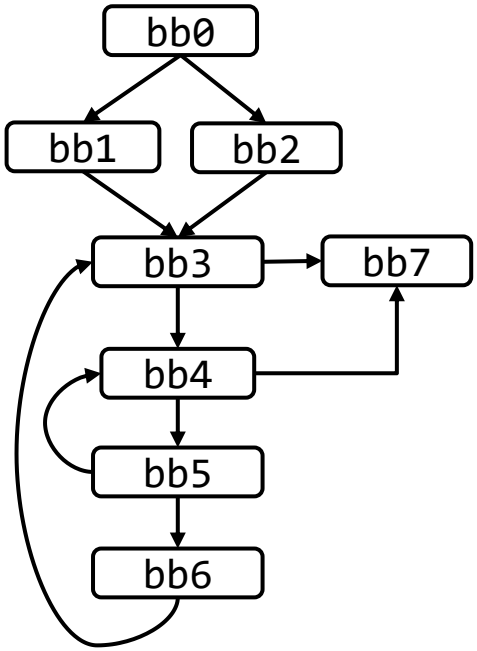


优化结果

- 重新编号
- 删除只有一个元素的phi指令



如何计算支配边界？



$$Predecessor(bb_0) = \{\}$$
$$Predecessor(bb_1) = \{bb_0\}$$
$$Predecessor(bb_2) = \{bb_0\}$$
$$Predecessor(bb_3) = \{bb_1, bb_2, bb_6\}$$
$$Predecessor(bb_4) = \{bb3, bb_5\}$$
$$Predecessor(bb_5) = \{bb_4\}$$
$$Predecessor(bb_6) = \{bb_5\}$$
$$Predecessor(bb_7) = \{bb_3, bb_4\}$$

$$Dom(bb_0) = \{bb_0\}$$
$$Dom(bb_1) = \{bb_0, bb_1\}$$
$$Dom(bb_2) = \{bb_0, bb_2\}$$
$$Dom(bb_3) = \{bb_0, bb_3\}$$
$$Dom(bb_4) = \{bb_0, bb3, bb_4\}$$
$$Dom(bb_5) = \{bb_0, bb3, bb_4, bb_5\}$$
$$Dom(bb_6) = \{bb_0, bb3, bb_4, bb5, bb_6\}$$
$$Dom(bb_7) = \{bb_0, bb_3, bb_7\}$$

如何计算支配边界？

$Predecessor(bb_0) = \{\}$
$Predecessor(bb_1) = \{bb_0\}$
$Predecessor(bb_2) = \{bb_0\}$
$Predecessor(bb_3) = \{bb_1, bb_2, bb_6\}$
$Predecessor(bb_4) = \{bb3, bb_5\}$
$Predecessor(bb_5) = \{bb_4\}$
$Predecessor(bb_6) = \{bb_5\}$
$Predecessor(bb_7) = \{bb_3, bb_4\}$

$Dom(bb_0) = \{bb_0\}$
$Dom(bb_1) = \{bb_0, bb_1\}$
$Dom(bb_2) = \{bb_0, bb_2\}$
$Dom(bb_3) = \{bb_0, bb_3\}$
$Dom(bb_4) = \{bb_0, bb3, bb_4\}$
$Dom(bb_5) = \{bb_0, bb3, bb_4, bb_5\}$
$Dom(bb_6) = \{bb_0, bb3, bb_4, bb5, bb_6\}$
$Dom(bb_7) = \{bb_0, bb_3\}$

$\forall\, bb_i \in Dom(bb_0) - IDom(bb_1),$
$bb_1 \in DF(bb_i)$

$DF(bb_0) = \{\}$
$DF(bb_1) = \{bb_3\}$
$DF(bb_2) = \{bb_3\}$
$DF(bb_3) = \{bb_3\}$
$DF(bb_4) = \{bb3, bb_4, bb_7\}$
$DF(bb_5) = \{bb3, bb_4\}$
$DF(bb_6) = \{bb_3\}$
$DF(bb_7) = \{\}$

练习

- 分析右侧代码：
  1) 计算支配树
  2) 计算支配边界
  3) 修改为SSA