静态单赋值（SSA）

1 构造 SSA

1.1 插入 ϕ 指令

- 在控制流图中，如果从起点到 n_2 节点的每条路径都经过 n_1 节点，那么就称 n_1 支配（dominates）n_2。
- 如果 n_1 支配 n_2 并且 $n_1 \neq n_2$，那么就称 n_1 严格支配（strictly dominates）n_2。
- 如果 u 支配 v，控制流图中有一条边从 v 到达 w，而且 u 不能严格支配 w，那么就称 w 是 u 的一个支配边界（dominance frontier）。

  $\mathrm{DF}(u)$ 表示 u 的所有支配边界构成的控制流图节点集合。
- 计算插入 ϕ 指令的位置
  - $S \leftarrow \mathrm{def}-\mathrm{of}(x)$
  - $\mathrm{DF}^{(0)}(S) \triangleq \varnothing$
  - $\mathrm{DF}^{(n+1)}(S) \triangleq \mathrm{DF}(S \cup \mathrm{DF}^{(n)}(S))$
  - $\mathrm{DF}^{(\infty)}(S) \triangleq \bigcup_{n\in\mathbb{N}} \mathrm{DF}^{(n)}(S)$

1.2 变量重命名

严格支配关系是有传递性的

- u 是 v 的直接支配节点（immediate dominator, idom）当且仅当 u 严格支配 v 并且不存在 w 是的 u 严格支配 w 并且 w 严格支配 v。

  使得
- 定理：直接支配关系构成了一棵以控制流图起点为根节点的树。这一树结构称为支配树（dominance tree）
- 利用支配树完成变量重命名

  Phi指令只考虑def，其他指令先处理use再处理def
  - 对支配树做深度优先遍历；
  - 当遍历进入一个节点 n 时，依次处理节点 n 中的每条指令 i；

    依次处理！
  - 先处理 i 的 use 变量（ϕ 指令除外），再处理 i 的 def 变量；
  - 处理完 n 中所有指令后，处理 n 的后继节点（控制流图中从 n 出发一步可达的节点）中所有 ϕ 指令的 ~~def~~ 变量。

    use
- UpdateReachingDef(x, i)

  没有严格dominate（不是dominate tree上的祖先节点），就pop栈顶
  - r ← x.RD
  - While (the location of r's def does not dominates i) do r ← r.PD
  - x.RD ← r
- 处理节点 n 中非 ϕ 指令 i 的 use 变量 x

  Phi指令的use是其他地方的东西来补

- 执行 UpdateReachingDef(x, i)
- 将 i 中的 x 改为 x.RD

增加phi指令的时候只需要看def

- 处理节点 n 中指令 i 的 def 变量 x
  - 执行 UpdateReachingDef(x, i)
  - 创建 x 变量的新版本 x0
  - 将 i 中的 x 改为 x0 放到栈当中去
  - x0.PD ← x.RD, x.RD ← x0
- 处理 n 的后继节点中 ϕ 指令 i 的 use 变量 x
  - 执行 UpdateReachingDef(x, i)
  - 将 i 中的对应 n 节点的 x 改为 x.RD
- 利用支配树计算支配边界
  - (1) 将所有节点的支配边界 DF(u) 都初始化为空集
  - (2) 依次考虑每一条控制流图中的边，假设它是从 u 到 v 的边
  - (2.1) 只要 $u \neq \text{idom}(v)$，就执行以下两项操作
  - (2.2) 将 v 加入 DF(u)
  - (2.3) $u \leftarrow \text{idom}(u)$

2 消去 SSA

- 计算 ϕ 网
  - 对于每一个变量 v，将 phiweb(x) 初始化为单元集 $\{x\}$
  - 对于每一条 ϕ 指令 $x_0 = \phi(x_1, x_2, ..., x_k)$, 将 phiweb$(x_i)$ 都合并起来
- 最终删除 ϕ 指令时，在同一个 phiweb(x) 中的所有变量应当合并为一个变量。

Live的区域不能重叠才能合并｜和

Phi指令不是函数！是虚拟符号

添加move指令可以保证phi web得到的控制流图中的变量不重叠。