



软件分析

# 过程间分析加速技术

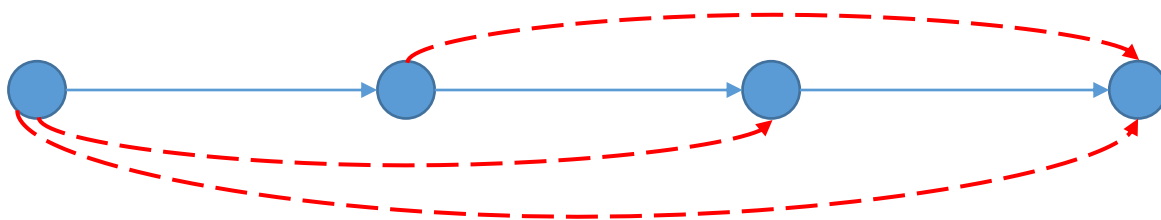
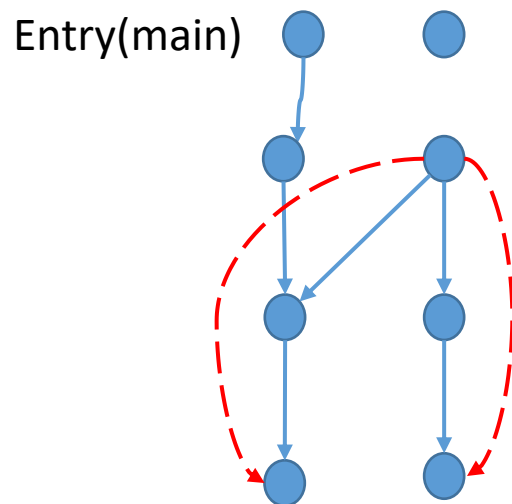
熊英飞

北京大学



# 求解算法缺陷

- 无效计算
  - 我们只关心从起始结点开始的可达性
  - CFL求解算法都会计算该过程内部的可达性
- 重复计算
  - 一条边可能从几个不同途径添加，导致重复计算

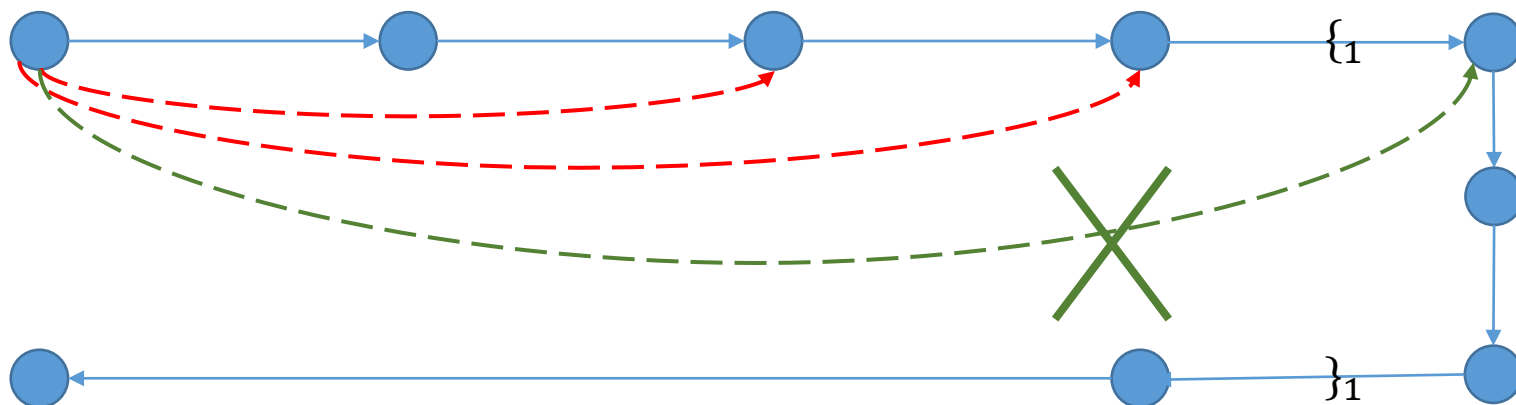




# 尝试1

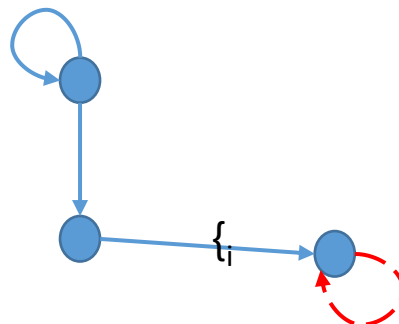
- 只添加从Entry出发的边
  - 不会对Entry不可达的路径进行无效计算
  - 因为固定顺序，不会产生重复计算

d@Entry(main)

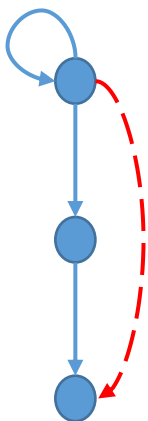




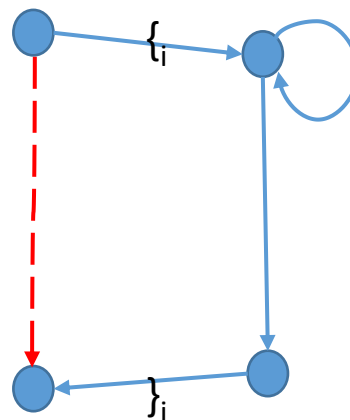
# 改进Dyck-CFL-Reachability求解规则



用回边标记当前  
可以分析的过程



过程内分析



调用返回总结



# 原始CFL-Reachability求解算法的复杂度

- $O(n^3)$
- $n$ 为结点数
- 假设文法的大小远远小于 $n$
- 图中最多有 $n*n$ 条边
  - 按规则a添加边的复杂度为 $O(n)$
  - 给定一条边, 检查规则b的复杂度为 $O(1)$
  - 给定一条边, 检查规则c的复杂度为 $O(n)$
- 总复杂度 $O(n^3)$

```
For(each node) {  
    根据规则(a)加边  
}  
ToVisit ← 所有边  
While(ToVisit.size > 0) {  
    从ToVisit中取出任意边  
    根据规则(b)加边  
    查看前后结点的边的组合, 根据  
    规则(c)加边  
    以上两步新加边加入ToVisit  
}
```

# 改进后CFL-Reachability 求解算法的复杂度

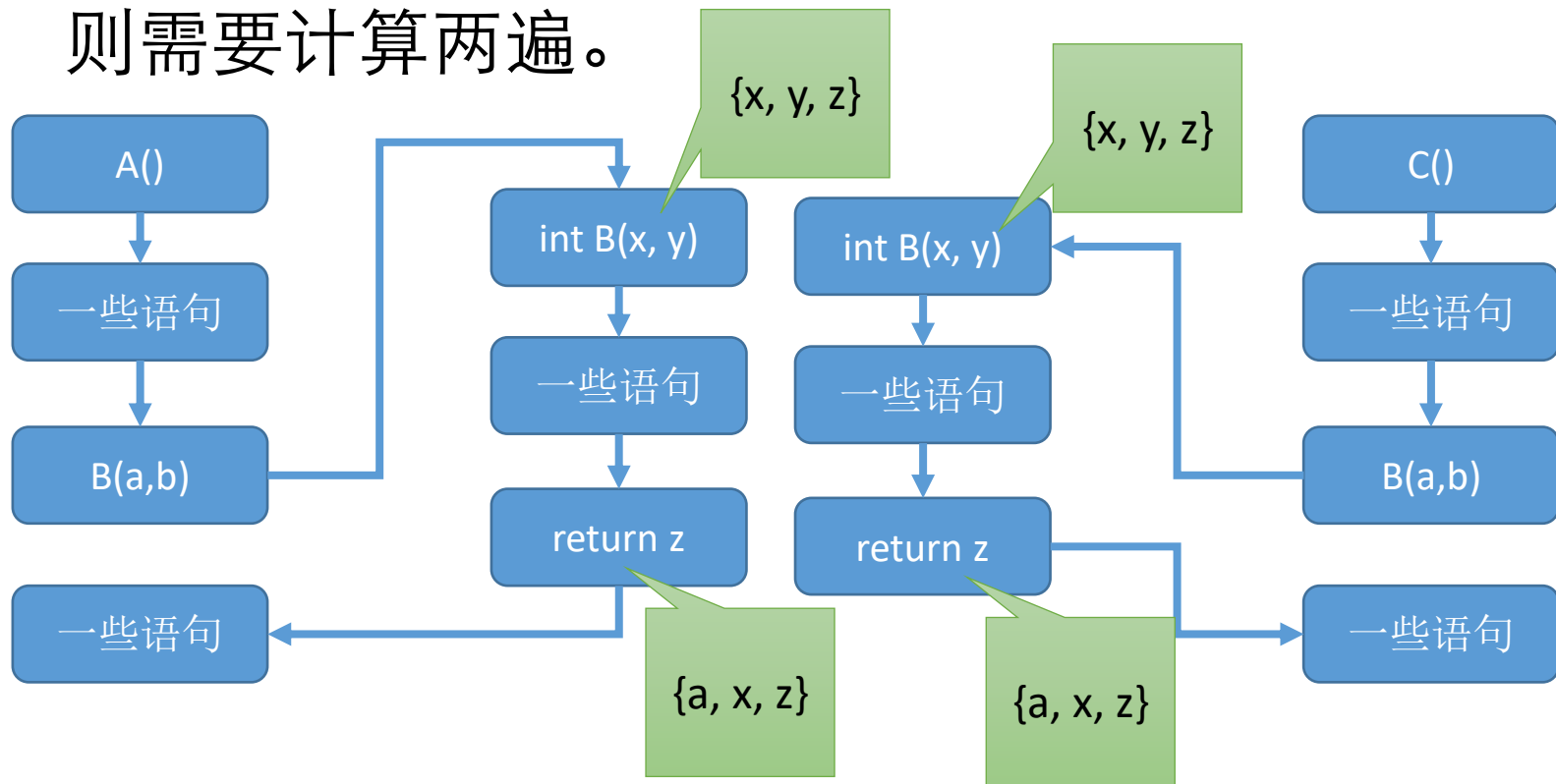


- Tom Reps证明
  - 算法复杂度为 $O(ED^3)$
  - E为控制流图上的边数，D为每个控制流图节点展开的节点数



# 换个角度理解该加速算法

- 在基于克隆的分析，如果同一个过程要分析两遍，则需要计算两遍。

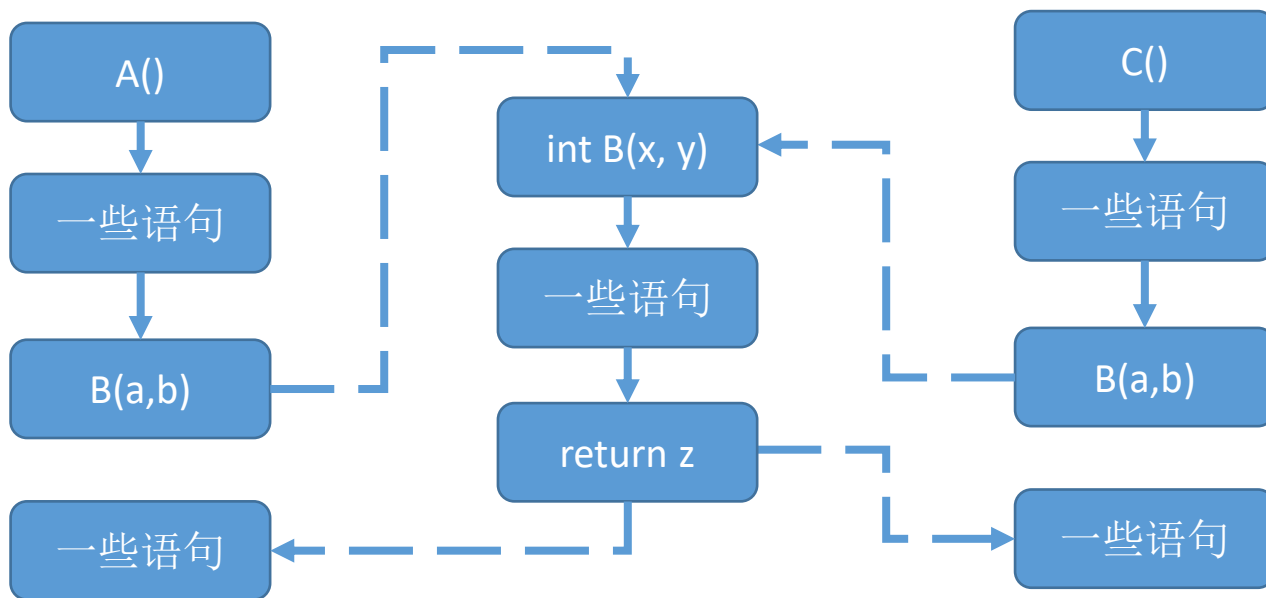






# 基于动态规划的分析

- 改进后的CFL-Reachability用边记录了之前的结果
- 如果初始状态任意子集在记录中存在，则重用记录，避免重新计算



# 过程间分析两种典型加速技术

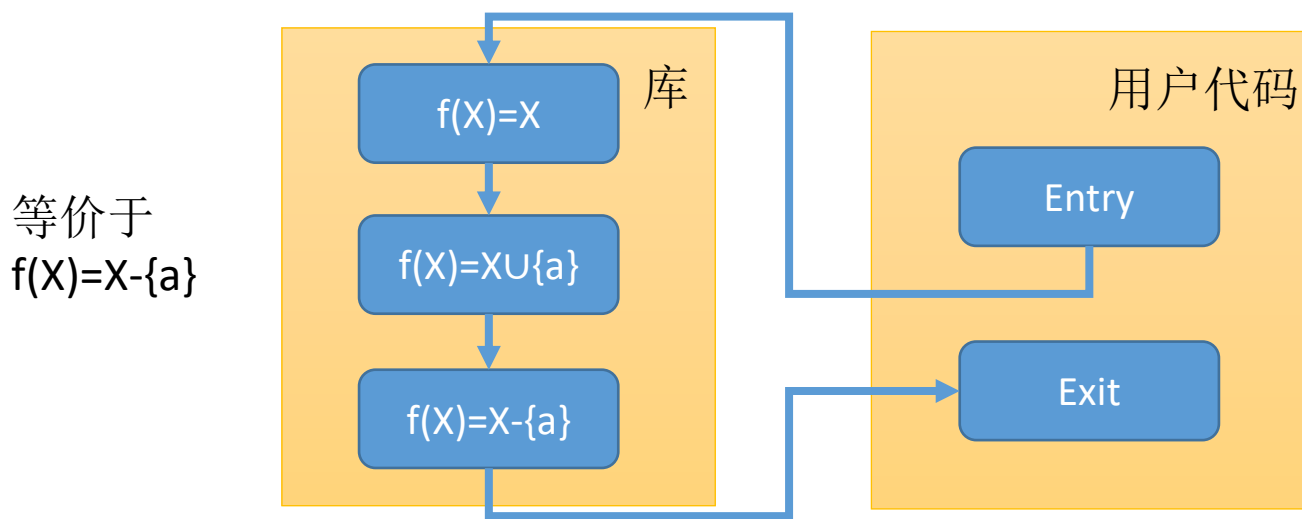


- 基于动态规划的加速技术
  - 通过记录之前计算过的信息来加速
  - 又叫做Top-down Summary、Tabulating Algorithm等
- 基于函数摘要的加速技术
  - 通过对函数内部的函数进行合并来加速
  - 通常用于提前对于函数库等进行分析
  - 在选择合适的函数表示的时候，也可以加快分析执行
  - 也叫作Bottom-up Summary、Functional Analysis、Modular Analysis等



# 基于函数摘要的加速技术

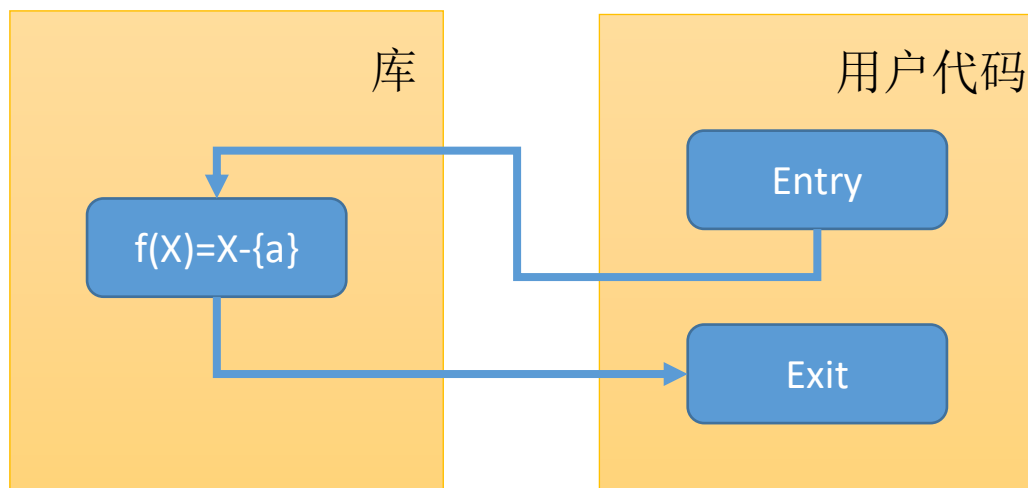
- 动机1：在数据流分析中，很多转换函数的效果可以互相抵消，但我们还是要针对每一个进行计算
- 动机2：程序分析中大量代码是库代码，往往分析一个很小的程序就要分析大量库代码。





# 基于函数摘要的加速技术

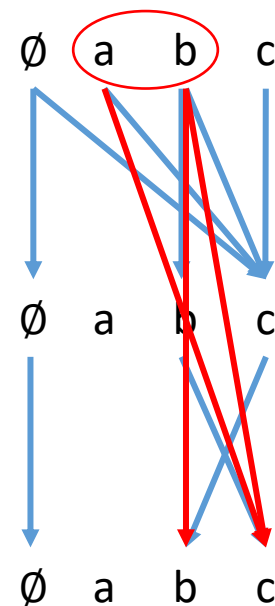
- 将一个过程摘要成一个转换函数
- 如果节省下来的冗余计算大于摘要花费，则加速了程序分析
- 库函数可以提前做成摘要，在分析用户代码的时候直接使用摘要



# CFL-Reachability与函数摘要



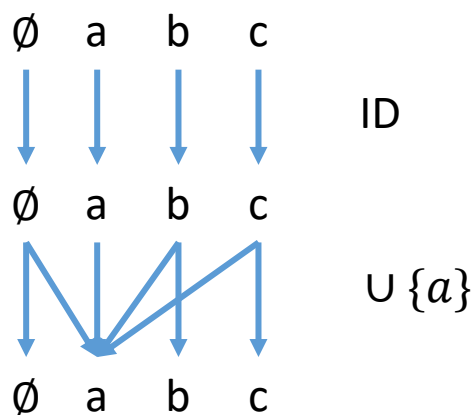
- CFL-Reachability再次解决了这个问题
- 过程入口点和出口点的可达性即为过程的函数摘要
- 只需要先对特定过程的图进行分析就能创建摘要





# CFL-Reachability的问题

- CFL-Reachability展开表示了转换函数，在摘要计算上并不高效
- 例：很多分析的格都由变量组成，特别是全局变量

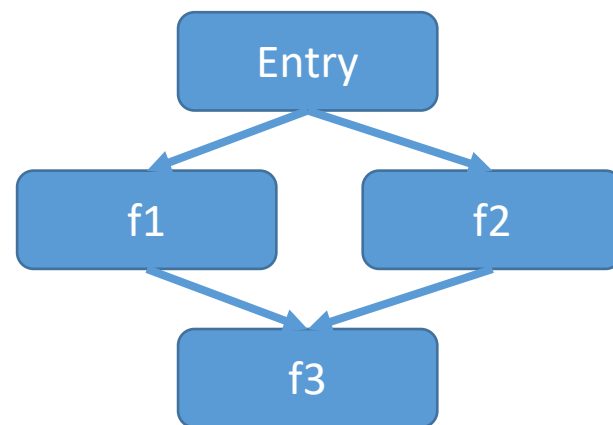


从函数定义上我们可以很容易合并这两个转换，但如果在图上计算摘要就要分别算每一个变量的可达性



# 基于函数摘要的加速技术

- 沿控制流图合并转换函数
  - $f_s = f_3 \circ (f_1 \sqcup f_2)$
- 需要给每个函数统一的抽象表示
- 需要定义该抽象表示上的 $\circ$ 和 $\sqcup$ 操作，这些操作是对该抽象表示封闭的，并且对任意 $x$ 满足如下条件：
  - $(f_2 \circ f_1)(x) = f_2(f_1(x))$
  - $(f_1 \sqcup f_2)(x) = f_1(x) \sqcup f_2(x)$





# 合并操作符-gen/kill标准型

- $f(x) = gen \cup (x - kill)$ , 半格操作为并集

$$\begin{aligned} f_2 \circ f_1(x) &= gen_2 \cup \left( (gen_1 \cup (x - kill_1)) - kill_2 \right) \\ &= (gen_2 \cup (gen_1 - kill_2)) \cup (x - (kill_1 \cup kill_2)) \end{aligned}$$

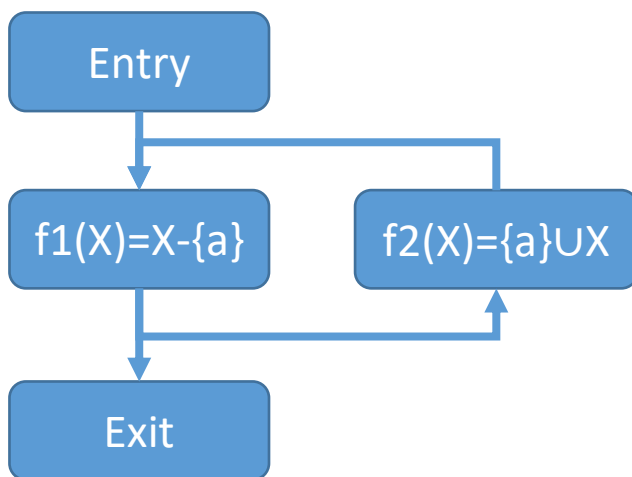
$$\begin{aligned} (f_1 \sqcup f_2)(x) &= f_1(x) \sqcup f_2(x) \\ &= (gen_1 \cup (x - kill_1)) \cup (gen_2 \cup (x - kill_2)) \\ &= (gen_1 \cup gen_2) \cup (x - (kill_1 \cap kill_2)) \end{aligned}$$

- 因此, 函数的抽象表示为集合的二元组  $(Gen, Kill)$ , 其中
  - $(g_2, k_2) \circ (g_1, k_1) = (g_2 \cup (g_1 - k_2), k_1 \cup k_2)$
  - $(g_1, k_1) \sqcup (g_2, k_2) = (g_1 \cup g_2, k_1 \cap k_2)$





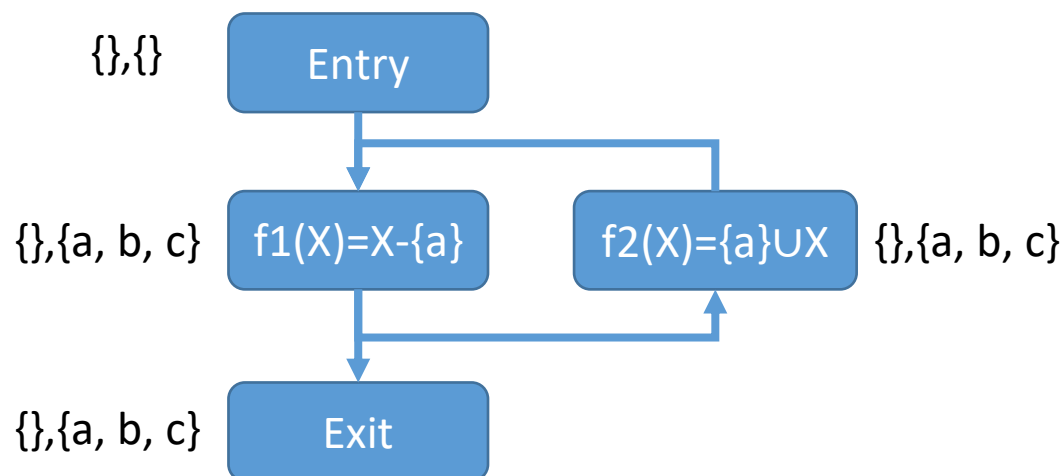
# 问题： 如何处理循环？



在函数上执行数据流分析



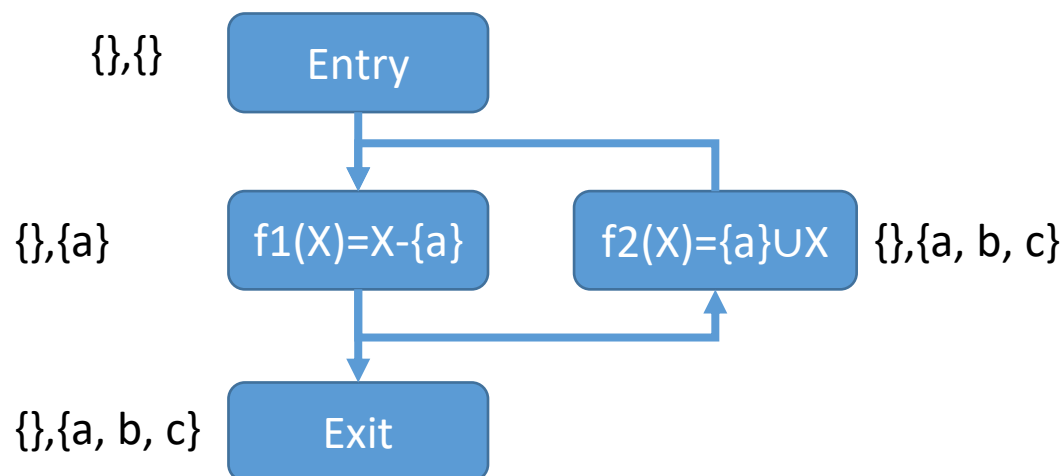
# 问题： 如何处理循环？



$$(g_2, k_2) \circ (g_1, k_1) = (g_2 \cup (g_1 - k_2), k_1 \cup k_2)$$
$$(g_1, k_1) \sqcap (g_2, k_2) = (g_1 \cup g_2, k_1 \cap k_2)$$



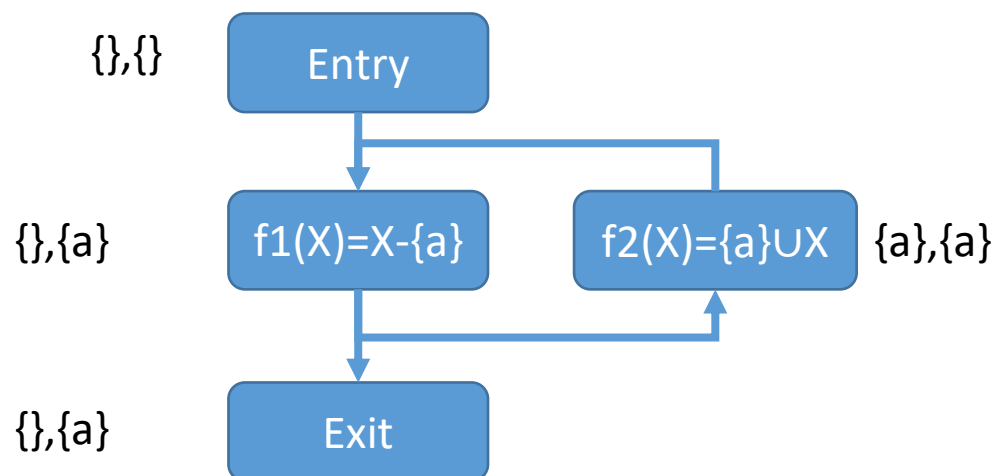
# 问题： 如何处理循环？



$$(g_2, k_2) \circ (g_1, k_1) = (g_2 \cup (g_1 - k_2), k_1 \cup k_2)$$
$$(g_1, k_1) \sqcap (g_2, k_2) = (g_1 \cup g_2, k_1 \cap k_2)$$



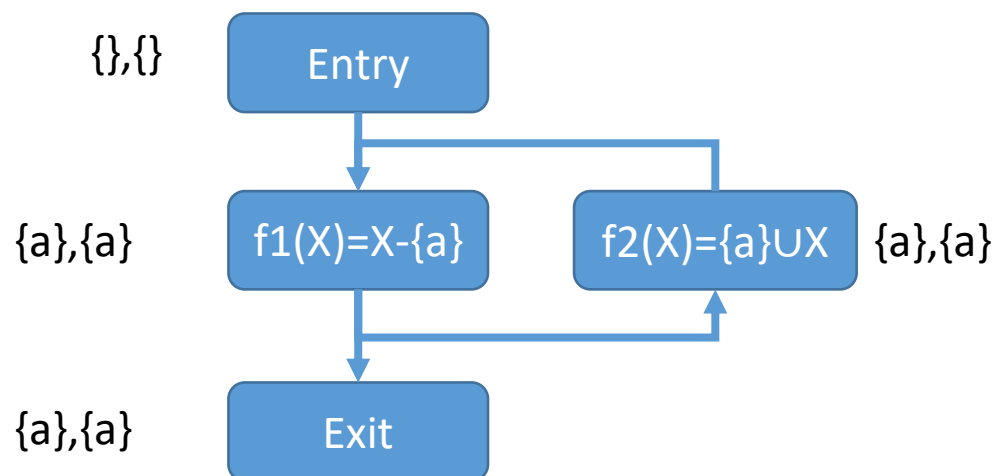
# 问题： 如何处理循环？



$$(g_2, k_2) \circ (g_1, k_1) = (g_2 \cup (g_1 - k_2), k_1 \cup k_2)$$
$$(g_1, k_1) \sqcap (g_2, k_2) = (g_1 \cup g_2, k_1 \cap k_2)$$



# 问题： 如何处理循环？



$$(g_2, k_2) \circ (g_1, k_1) = (g_2 \cup (g_1 - k_2), k_1 \cup k_2)$$
$$(g_1, k_1) \sqcap (g_2, k_2) = (g_1 \cup g_2, k_1 \cap k_2)$$

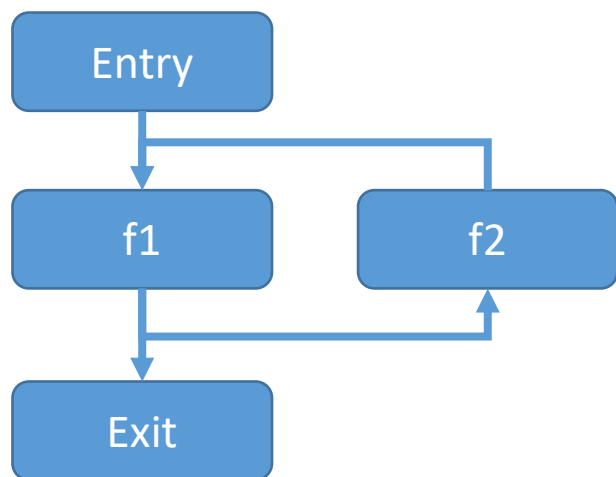


# 函数的数据流分析——半格

- 半格格元素为 Gen/Kill 标准型的抽象表示，其中 Gen 和 Kill 都只包含为原始分析中半格元素，是有限集合
- 交汇运算  $\sqcap$  为函数上的并操作，该操作满足幂等性、交换性、结合性
  - 证明：由  $(g_1, k_1) \sqcap (g_2, k_2) = (g_1 \cup g_2, k_1 \cap k_2)$  可见，交汇运算可以分解成一个集合并和一个集合交，由两种运算都满足幂等性、交换性、结合性可知原结论成立。
- 最大元 T 中，Gen 为空集，Kill 为全集



# 函数的数据流分析——半格



- 每个程序点上的数据流分析结果表示从Entry到该节点所有路径的函数摘要
- Entry的初值为 $(\{\}, \{\})$ ，即等价变换
- $f_i$ 的转换函数为
  - $f_{f_i}((g, k)) = f_i \circ (g, k)$



# 转换函数的单调性

- 引理：任意结点上的转换函数都是单调的
  - 如果  $(g_1, k_1) \sqcup (g_2, k_2) = (g_1, k_1)$ ，需要证明  $(g_3, k_3) \circ (g_1, k_1) \sqcup (g_3, k_3) \circ (g_2, k_2) = (g_3, k_3) \circ (g_1, k_1)$
  - 由前提，可知  $g_1 \cup g_2 = g_1, k_1 \cap k_2 = k_1$
  - $(g_3, k_3) \circ (g_1, k_1) \sqcup (g_3, k_3) \circ (g_2, k_2)$   
 $= (g_3 \cup (g_1 - k_3), k_1 \cup k_3) \sqcup (g_3 \cup (g_2 - k_3), k_2 \cup k_3)$   
 $= (g_3 \cup (g_1 \cup g_2 - k_3), (k_1 \cap k_2) \cup k_3)$   
 $= (g_3 \cup (g_1 - k_3)), k_1 \cup k_3)$   
 $= (g_3, k_3) \circ (g_1, k_1)$





# 转换函数的分配性

- 引理：任意结点上的转换函数满足分配性
  - $(g_3, k_3) \circ ((g_1, k_1) \sqcup (g_2, k_2))$   
 $= (g_3, k_3) \circ (g_1 \cup g_2, k_1 \cap k_2)$   
 $= (g_3 \cup (g_1 \cup g_2 - k_3), (k_1 \cap k_2) \cup k_3)$   
 $= (g_3 \cup (g_1 - k_3), k_3 \cup k_1) \sqcup (g_3 \cup (g_2 - k_3), k_3 \cup k_2)$   
 $= (g_3, k_3) \circ (g_1, k_1) \sqcup (g_3, k_3) \circ (g_2, k_2)$

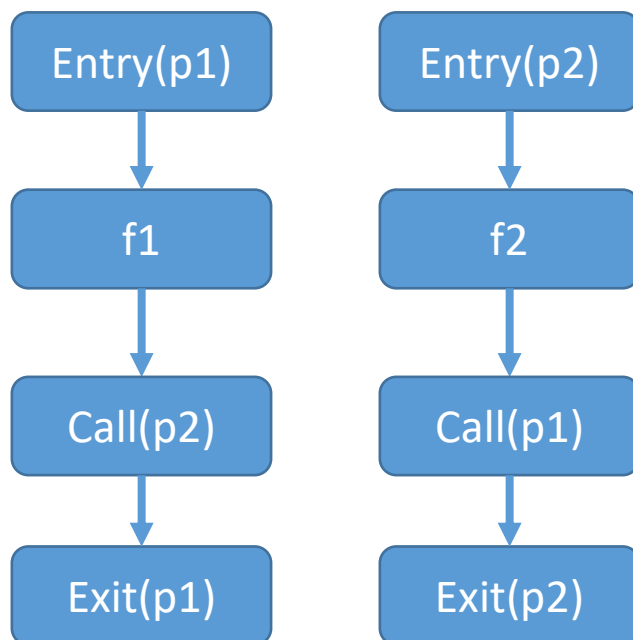


# 正确性

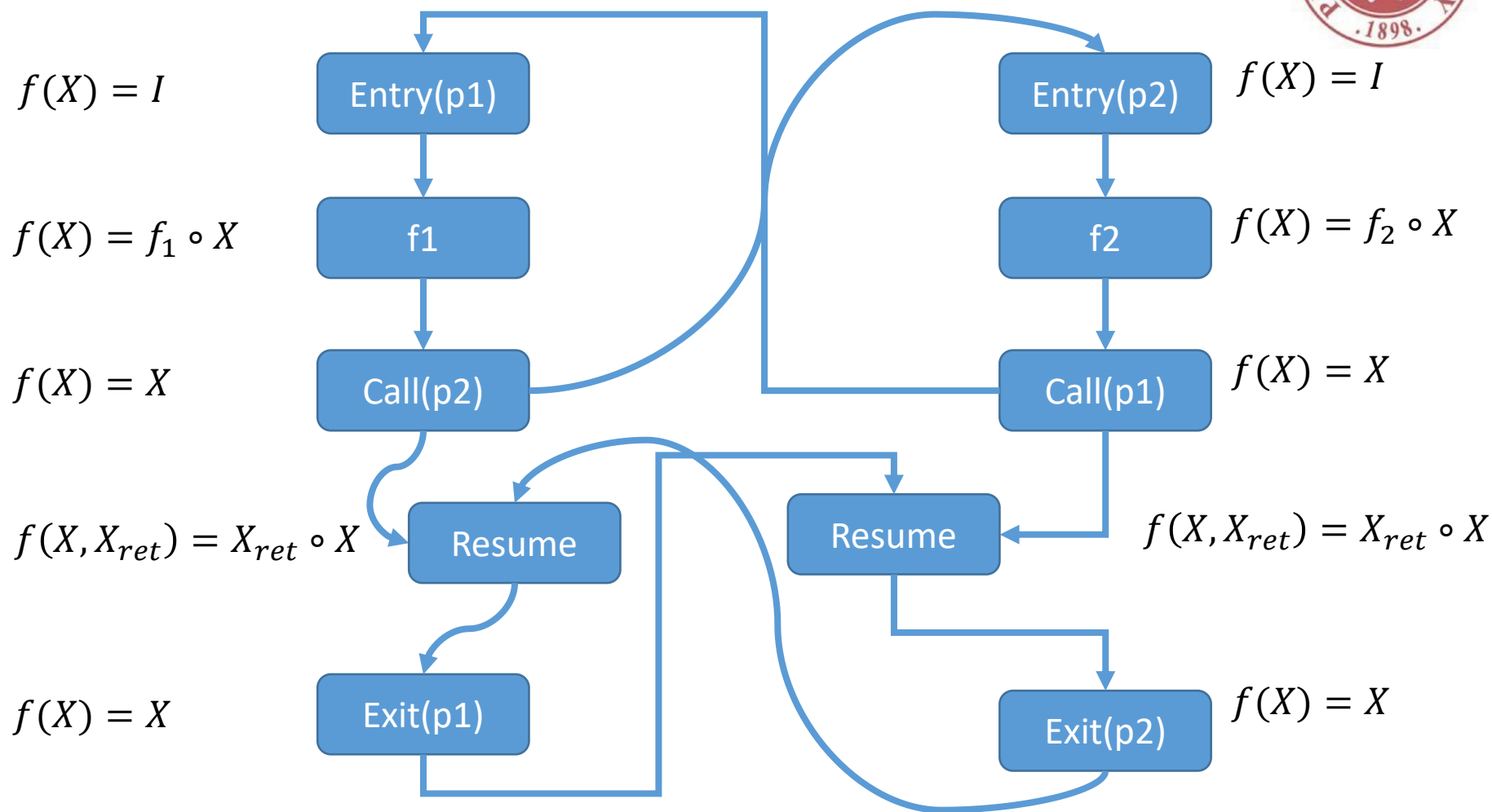
- 定理：用以上方法做出来的函数摘要进行数据流分析，分析结果和原数据流分析完全相同
  - 证明：
    - 容易证明单条路径上最后一个结点的分析结果（函数摘要）和原分析等价
    - 多条路径函数摘要的合并也和原分析等价
      - 因为数据流分析标准型满足分配性，所以原分析结果等价于单条路径上完成原分析之后再合并
    - 根据转换函数的分配性，即函数摘要和原分析等价



# 函数调用如何处理？



# 函数调用如何处理?

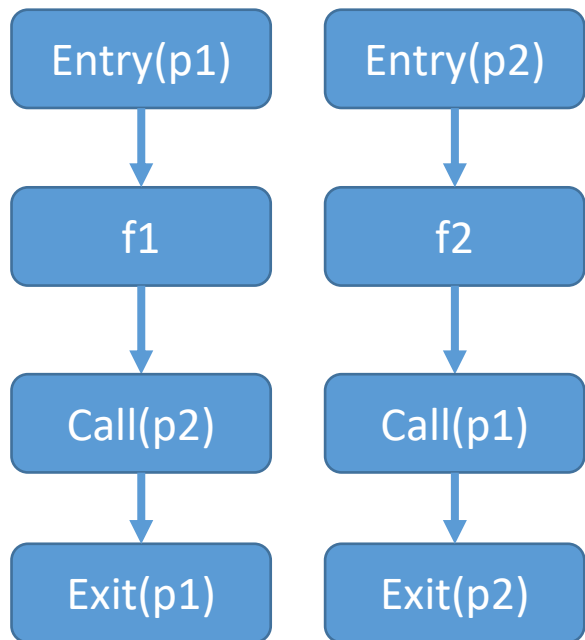


$I = (\{ \}, \{ \})$

是否需要考虑上下文敏感性?



# 另一个角度：看成方程组



$$I = (\{ \quad \}, \{ \quad \})$$

$$OUT_{f1} = f_1 \circ I$$

$$OUT_{call(p2)} = OUT_{p2} \circ OUT_{f1}$$

$$OUT_{p1} = OUT_{call(p2)}$$

$$OUT_{f2} = f_2 \circ I$$

$$OUT_{call(p1)} = OUT_{p1} \circ OUT_{f2}$$

$$OUT_{p2} = OUT_{call(p1)}$$



# 函数摘要的方法 vs 基于CFL可达性的方法

- 函数摘要可直接完成精确的过程间分析
  - 从main入口到关心的程序点之间做一个摘要，然后传入分析初值即可
- 对于输入集合比较大，而单个转换函数对集合改变较小的分析，基于摘要的方法可能达到较好效果
- 基于摘要的方法只能计算出口点的信息，不能知道中间点的信息



# 作业

- 把原始数据流分析中的并集换成交集，Gen/Kill 标准型上的交汇运算和组合运算还能定义出来吗？



# 参考资料

- Two Approaches to Interprocedural Data Flow Analysis
  - Micha Sharir and Amir Pnueli
  - New York University Technical Report, 1978
- Precise Interprocedural Dataflow Analysis via Graph Reachability
  - Thomas W. Reps, Susan Horwitz, Shmuel Sagiv
  - POPL 1995