软件分析

# 过程间分析加速技术

熊英飞
北京大学
2017

# 复习

- 什么是上下文敏感性？
- 对比基于克隆的方法和基于CFL可达性的方法，他们的上下文敏感性有什么不一样？
  - 基于克隆的方法只能精确考虑最近k次调用的情况，基于CFL可达性的方法考虑所有上下文
  - 基于克隆的方法可以返回针对某个特定上下文的结果，基于CFL可达性的方法只考虑所有上下文的整合结果

# 补充术语：IFDS

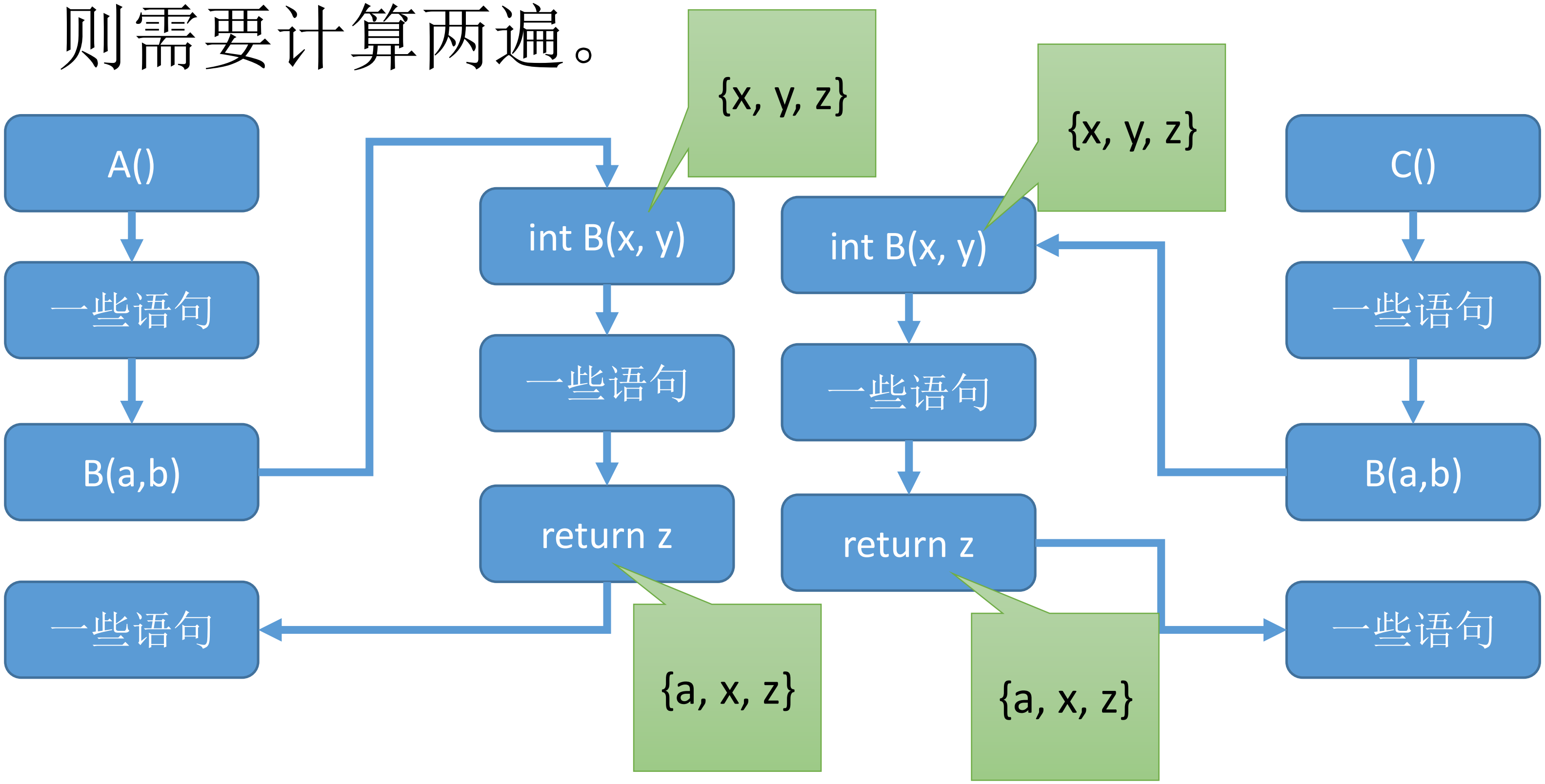
- 可以用CFL可达性解决的问题又称为IFDS问题
  - Interprocedural, finite, distributive, subset problems
- 后来通常称CFL可达性解决数据流问题的方法称为IFDS框架或者IFDS方法

# 如何让过程间分析变得更快一些？

- 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

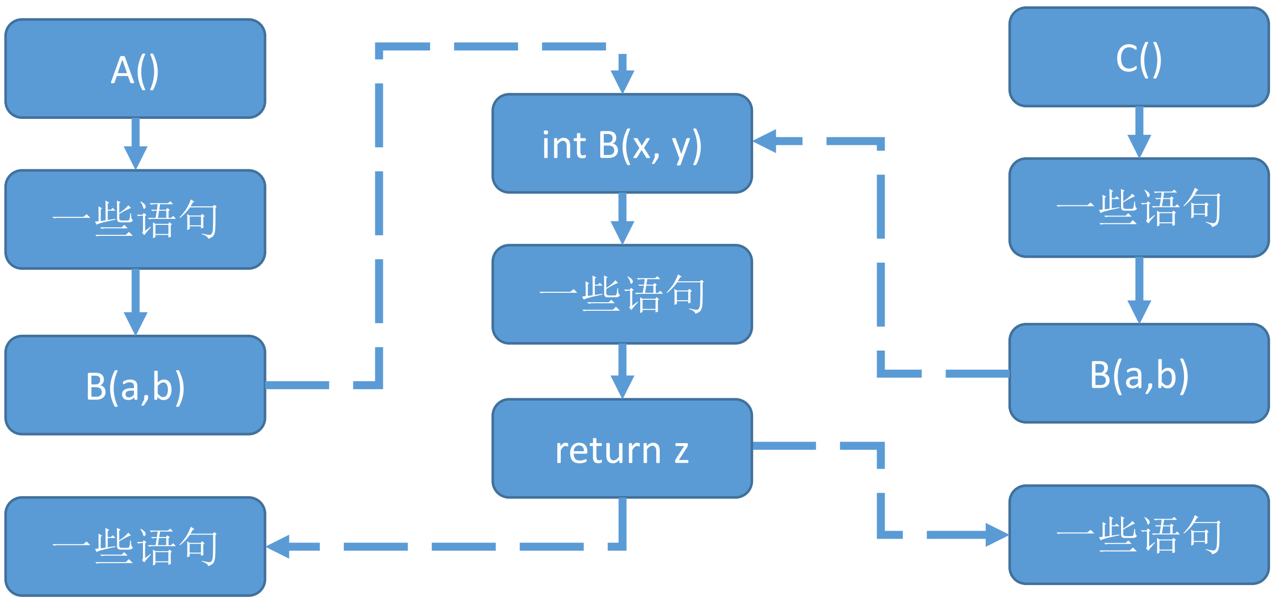
# 基于克隆的过程间分析的问题

- 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

- 采用动态规划/Memoization记录之前分析过的信息
  - {x, y, z}=>{a, x, z}
- 如果初始状态在记录中存在，则重用记录，否则克隆B过程并重新计算



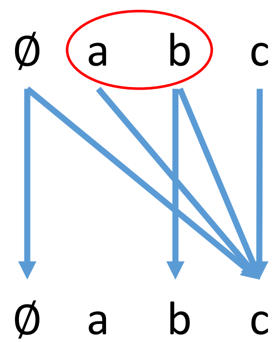
# 动态规划的问题

- 粒度太粗
  - 内存消耗大，加速效果不明显
- 递归调用不容易处理

- 存在算法处理递归调用的问题
- 但不容易处理前者的问题

# 复习：转换函数vs图可达性

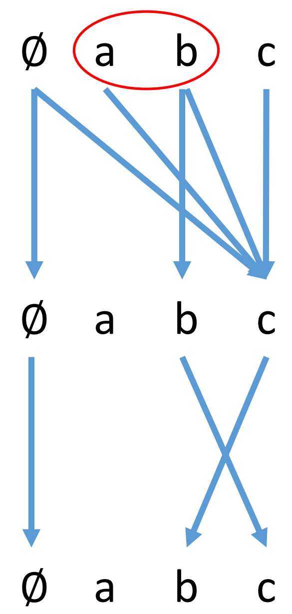
- 对于可分配函数f(X)=(X-{a})U{c}，其中X为{a,b,c}的子集
- 可以表示成图




- 求解f({a,b})变成了从 {a,b}出发的可达性问题
- 整个数据流分析转成可达性问题

# 改进动态规划记录

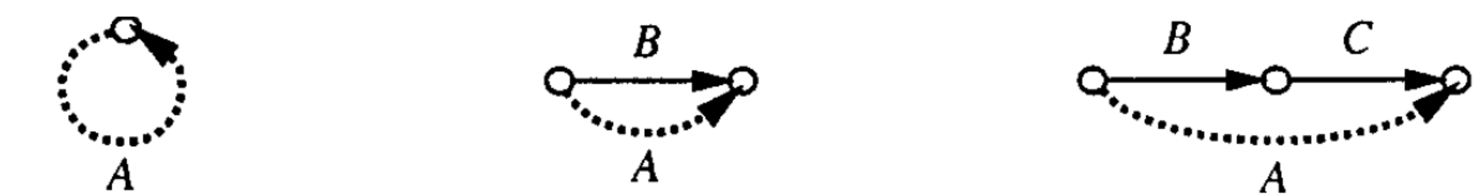
- 粗粒度记录：
  - {a, b}->{b, c}
- 细粒度记录：
  - a->{b}, b->{b, c}




- 给定新输入状态数据
  - {a, b, c}
- 粗粒度记录要重新计算a, b, c
- 细粒度记录只需要重新计算c

# 复习：求解CFL-Reachability的算法

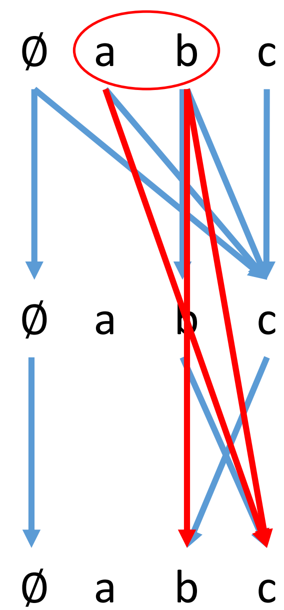
- 按如下三种模式不断添加边，直到没有边需要添加



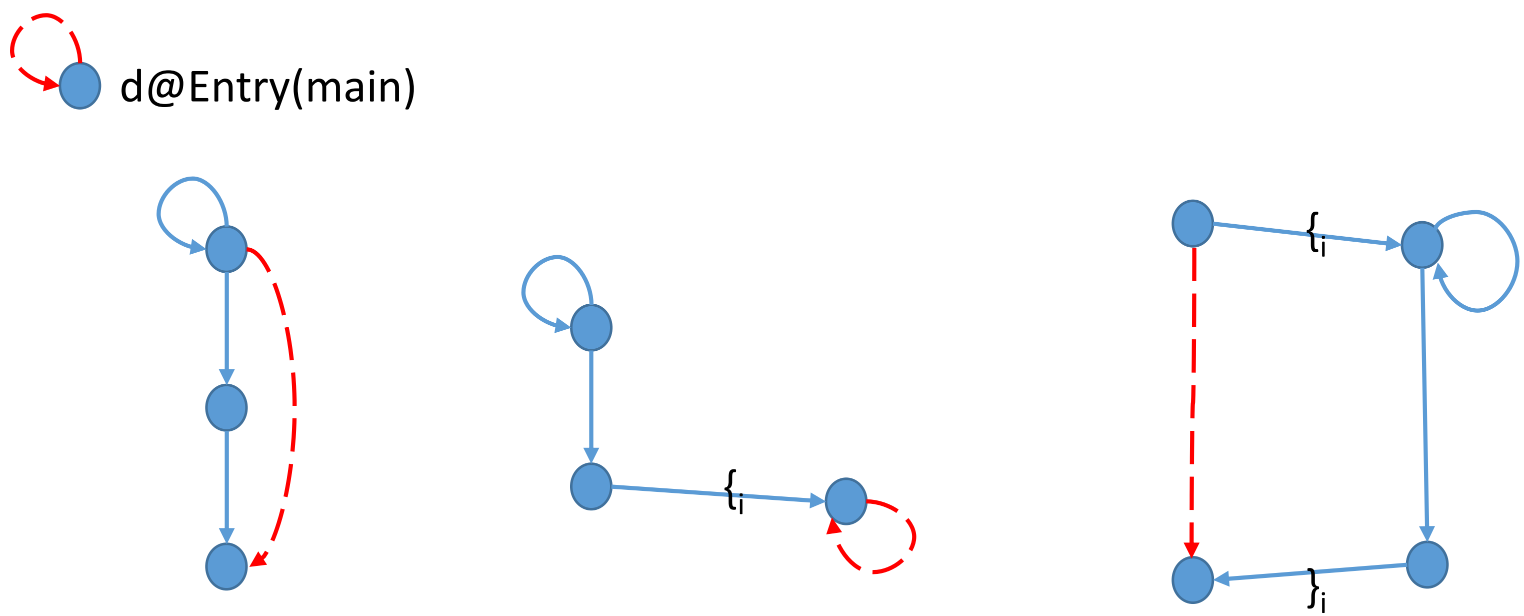

(a) $A \rightarrow \varepsilon$ (b) $A \rightarrow B$ (c) $A \rightarrow B\ C$

# CFL-Reachabilty求解算法与动态规划

- 新添加的边实际等价于细粒度记录
- 采取CFL-Reachability求解算法实际已经实现了动态规划的加速
- 但CFL-Reachability求解算法存在一个缺陷
  - 不管一个点是否从起始点可达，CFL求解算法都会计算该过程内部的可达性



# 改进Dyck-CFL-Reachability求解规则




- 用回边标记当前可以分析的过程
- 直接针对Dyck-CFL产生边

# 原始CFL-Reachability求解算法的复杂度

- $O(n^3)$
- n为结点数量
- 假设文法的大小远远小于n

- 上页三条规则实际可用worklist算法实现
  - 首先根据规则a添加边
  - 每添加一条边，检查是否有边可以根据规则b, c添加
- 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$O(n^3)$

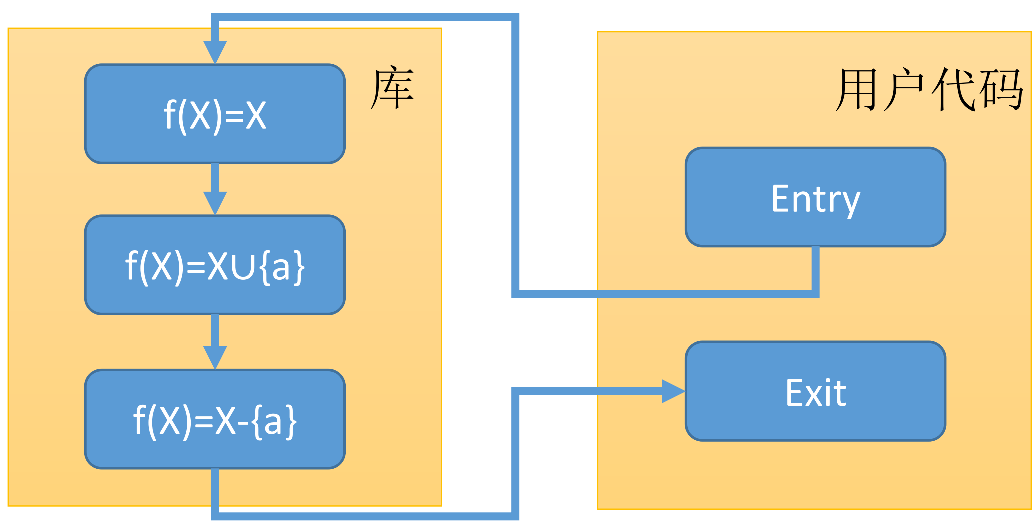
# 改进后CFL-Reachability求解算法的复杂度

- 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

# 基于函数摘要的加速技术

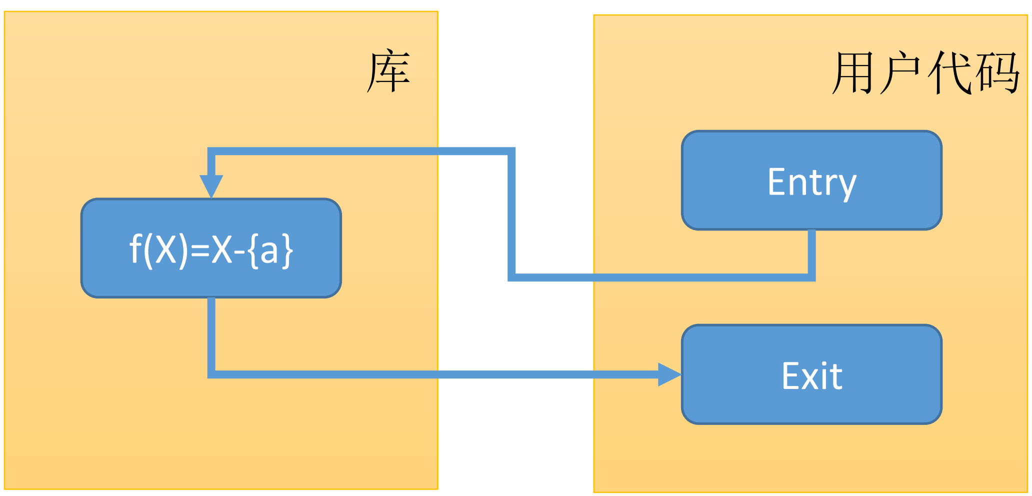
- 动机1：在数据流分析中，很多转换函数的效果可以互相抵消，但我们还是要针对每一个进行计算
- 动机2：程序分析中大量代码是库代码，往往分析一个很小的程序就要分析大量库代码。

等价于
f(X)=X-{a}



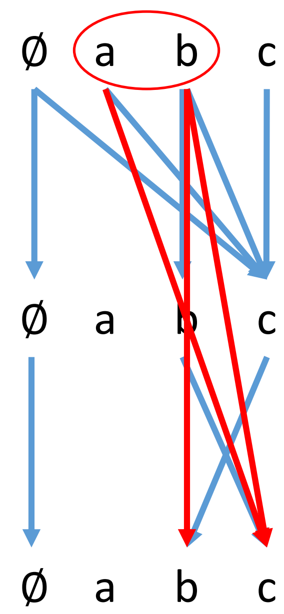
# 基于函数摘要的加速技术

- 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



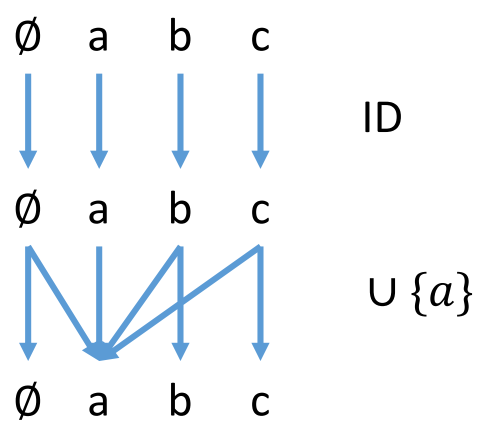
# CFL-Reachability与函数摘要

- 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

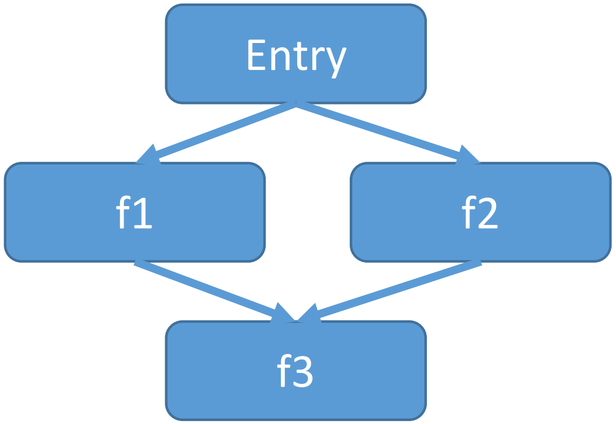
- CFL-Reachability展开表示了转换函数，在摘要计算上并不高效
- 例：很多分析的格都由变量组成，特别是全局变量




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
  - $f_s = f_3 \circ (f_1 \sqcap f_2)$
- 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$\sqcap$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：
  - $f_2 \circ f_1(x) = f_2(f_1(x))$
  - $f_1 \sqcap f_2(x) = f_1(x) \sqcap f_2(x)$



# 合并操作符-gen/kill标准型

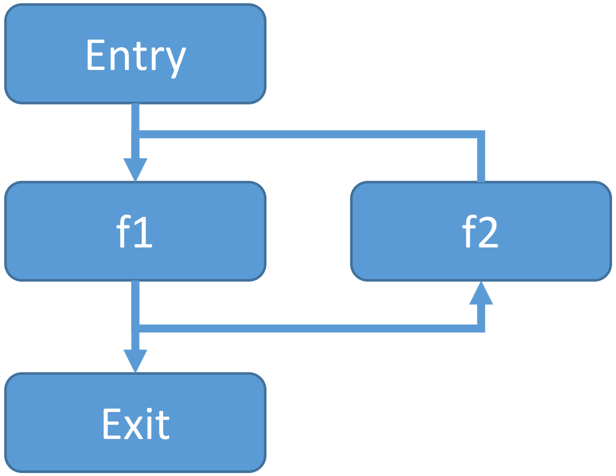
- $f(x) = gen \cup (x - kill)$，半格操作为并集

$$f_2 \circ f_1(x) = gen_2 \cup \Big( \big(gen_1 \cup (x - kill_1)\big) - kill_2 \Big)$$
$$= \big(gen_2 \cup (gen_1 - kill_2)\big) \cup \big(x - (kill_1 \cup kill_2)\big)$$

$$(f_1 \sqcap f_2)(x) = f_1(x) \sqcap f_2(x)$$
$$= \big(gen_1 \cup (x - kill_1)\big) \cup \big(gen_2 \cup (x - kill_2)\big)$$
$$= (gen_1 \cup gen_2) \cup \big(x - (kill_1 \cap kill_2)\big)$$

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

# 问题：如何处理循环？

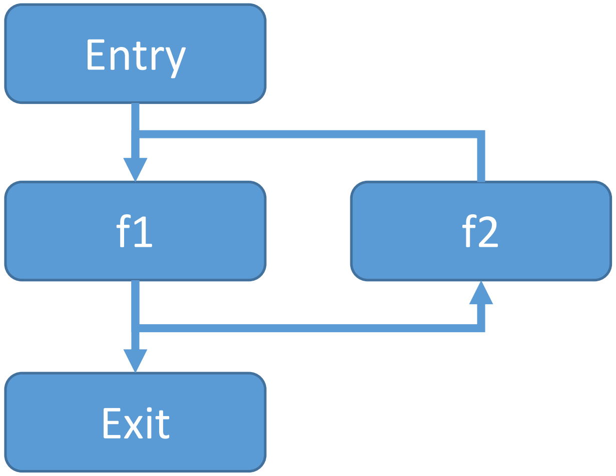



在函数上执行数据流分析

# 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⊓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({}, {})
- $f_i$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$(g_1,k_1)\sqcap(g_2,k_2)=(g_1,k_1)$，需要证明$(g_3,k_3)\circ(g_1,k_1)\sqcap(g_3,k_3)\circ(g_2,k_2)$=$(g_3,k_3)\circ(g_1,k_1)$
  - 由前提，可知$g_1\cup g_2=g_1, k_1\cap k_2=k_1$
  - $(g_3,k_3)\circ(g_1,k_1)\sqcap(g_3,k_3)\circ(g_2,k_2)$

  =$(g_3\cup(g_1-k_3), k_1\cup k_3)\sqcap(g_3\cup(g_2-k_3), k_2\cup k_3)$

  =$(g_3\cup(g_1\cup g_2-k_3), (k_1\cap k_2)\cup \mathrm{k}_3)$

  =$(g_3\cup(g_1-k_3)), k_1\cup \mathrm{k}_3)$

  =$(g_3,k_3)\circ(g_1,k_1)$

# 转换函数的分配性

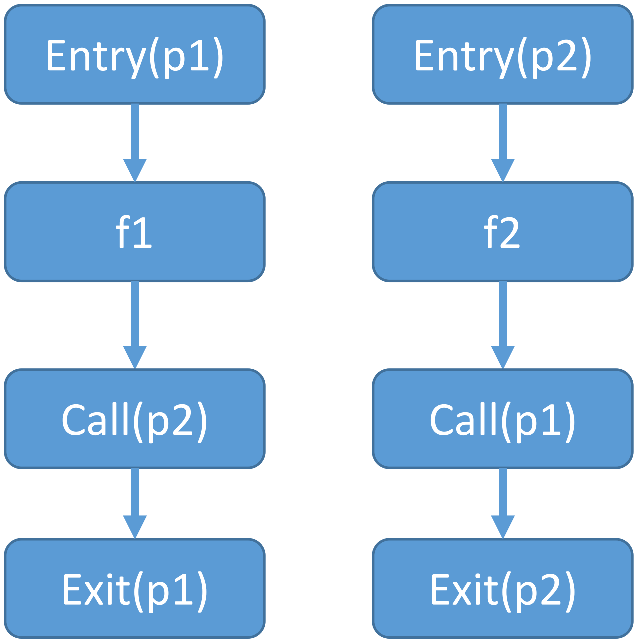
- 引理：任意结点上的转换函数满足分配性
  - $(g_3, k_3) \circ \left((g_1, k_1) \sqcap (g_2, k_2)\right)$
    $=(g_3, k_3) \circ (g_1 \cup g_2, k_1 \cap k_2)$
    $=(g_3 \cup (g_1 \cup g_2 - k_3), (k_1 \cap k_2) \cup \mathrm{k}_3)$

  - $(g_3, k_3) \circ (g_1, k_1) \sqcap (g_3, k_3) \circ (g_2, k_2)$
    $=(g_3 \cup (g_1 - k_3), k_1 \cup k_3) \circ (g_3 \cup (g_2 - k_3), k_2 \cup k_3)$
    $=(g_3 \cup (g_1 \cup g_2 - k_3), (k_1 \cap k_2) \cup \mathrm{k}_3)$

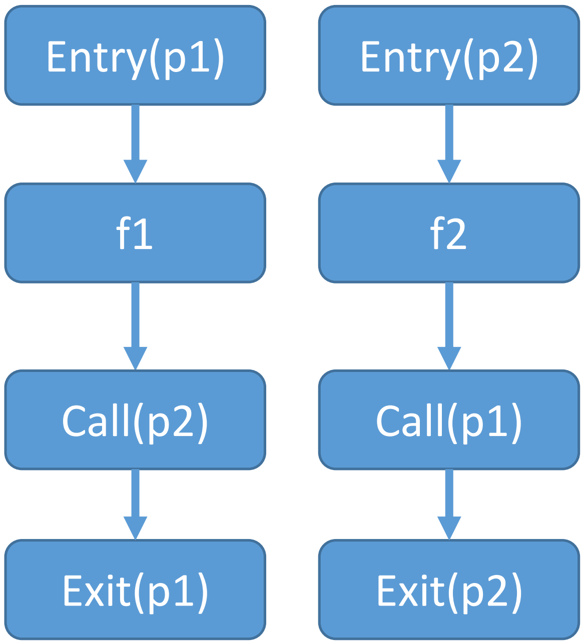
# 正确性

- 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：在满足分配性的情况下，数据流分析结果等同于理想值
    - 分析出来的函数摘要等同于把每条路径上的函数组合，然后再对结果区并，即原始数据流分析的理想值
    - 原始数据流分析的结果也等于理想值

# 函数调用如何处理？



# 将数据流分析看成方程组




$$
\begin{aligned}
&\mathrm{I} = (\{\quad\},\{\quad\}) \\
&\mathrm{DATA}_{\mathrm{f1}} = \mathrm{f}_1 \circ \mathrm{I} \\
&\mathrm{DATA}_{\mathrm{call(p2)}} = \mathrm{DATA}_{\mathrm{p2}} \circ \mathrm{DATA}_{\mathrm{f1}} \\
&\mathrm{DATA}_{\mathrm{p1}} = \mathrm{DATA}_{\mathrm{call(p2)}} \\
&\mathrm{DATA}_{\mathrm{f2}} = \mathrm{f}_2 \circ \mathrm{I} \\
&\mathrm{DATA}_{\mathrm{call(p1)}} = \mathrm{DATA}_{\mathrm{p1}} \circ \mathrm{DATA}_{\mathrm{f2}} \\
&\mathrm{DATA}_{\mathrm{p2}} = \mathrm{DATA}_{\mathrm{call(p1)}}
\end{aligned}
$$

- 用不动点算法求解方程组。
- 注意这里不需要上下文敏感性。

# 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

# 下节课

- 唐浩同学介绍Java分析框架SOOT的使用
- 请大家预先安装好SOOT
- 方法一：
  - 下载虚拟机（链接: http://pan.baidu.com/s/1dDuLPpJ 密码: xmwy），用VMWare加载（用户名：root；密码：123）。
  - 下载soot (https://ssebuild.cased.de/nightly/soot/lib/soot-trunk.jar)，放在/root/workspace/Test/目录下面（已经下载完毕）。
  - 预习小实验1：命令行运行commandline-example中给出的两个例子，观察输出文件；
  - 预习小实验2：打开桌面上的eclipse，运行core.NaiveSootExample。
- 方法二：
  - 配置JDK环境（例如OpenJDK1.7），课前请记下JDK的安装目录；
  - 可能需要安装eclipse。
  - 下载小实验的源码Test.zip（链接: http://pan.baidu.com/s/1pJsz9yF 密码: zpgi），内含soot-trunk.jar。
  - 预习小实验1-2（注意替换实验参数中的相应目录，Windows用户注意将目录分隔符替换为”;”）

# 作业（截止10月17日）

- 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？还能组成半格吗？给出你的证明。