软件分析

过程间分析

熊英飞
北京大学
2017

复习：数据流分析

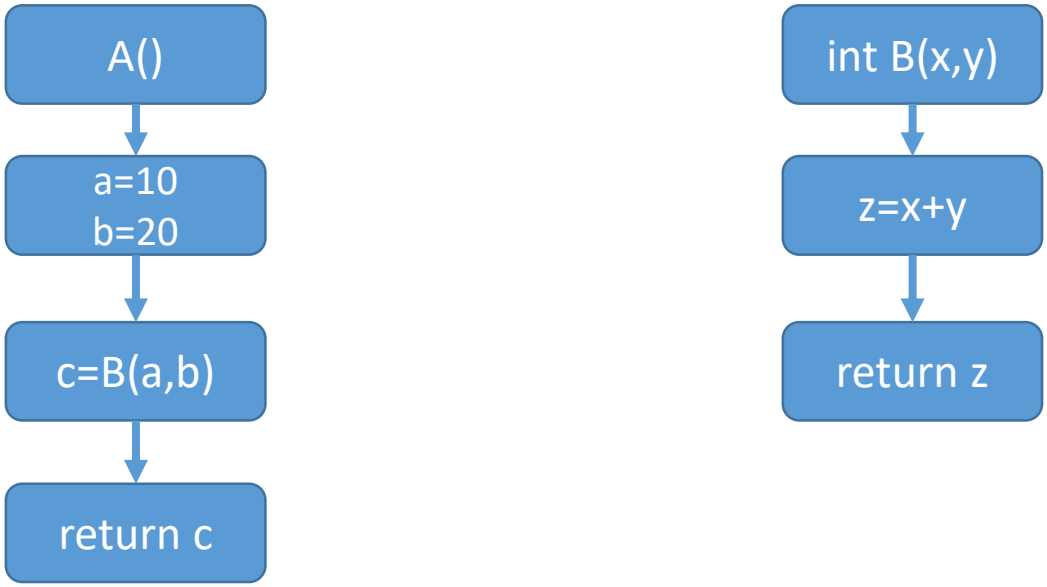
- 框架
  - 一个控制流图(V, E)
  - 一个有限高度的半格$(S, \sqcap)$
  - 一个entry的初值I
  - 一组结点转换函数，对任意$v \in V - entry$存在一个结点转换函数f_v

- 框架中的各元素代表什么含义？
- 求解算法如何工作？
- 为什么说该算法是正确的？为什么该算法是收敛的？

过程间分析

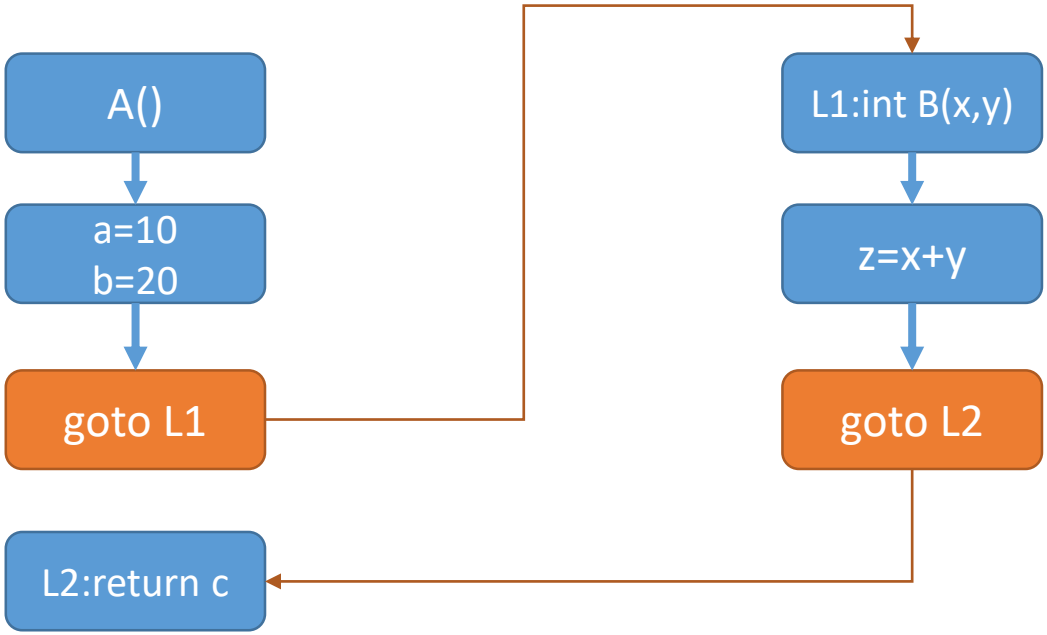
- 过程内分析Intra-procedural Analysis
  - 只考虑过程内部语句，不考虑过程调用
  - 目前的所有分析都是过程内的
- 过程间分析Inter-procedural Analysis
  - 考虑过程调用的分析
  - 有时又称为全程序分析Whole Program Analysis
  - 对于C, C++等语言，有时又称为链接时分析Link-time Analysis

过程间分析-示例



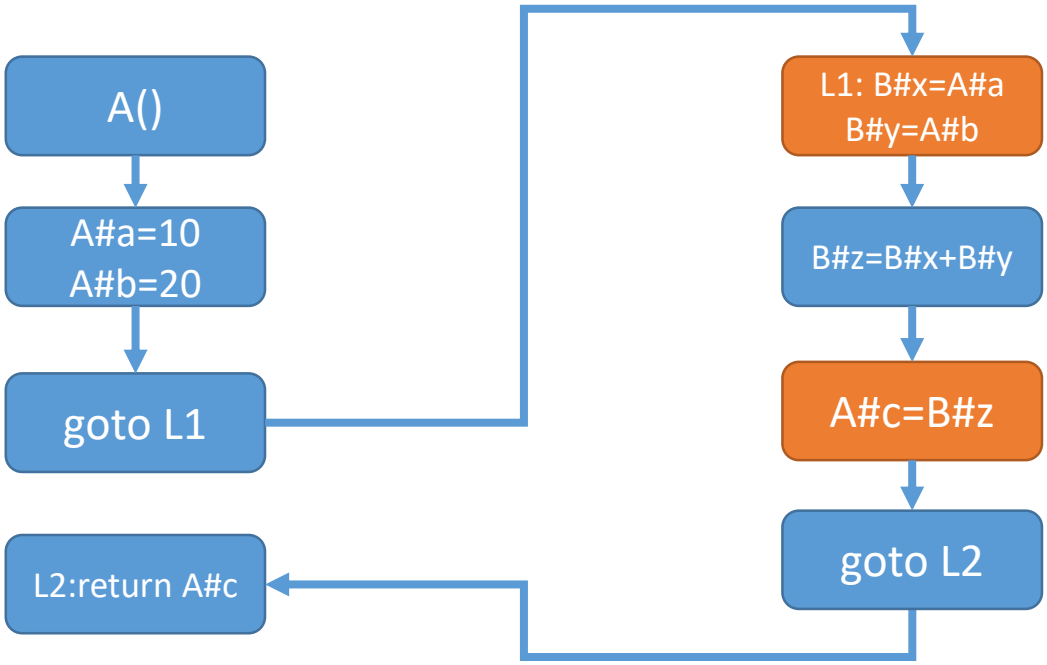
过程间分析-基本思路

- 把Call/Return语句当成goto语句



过程间分析-基本思路

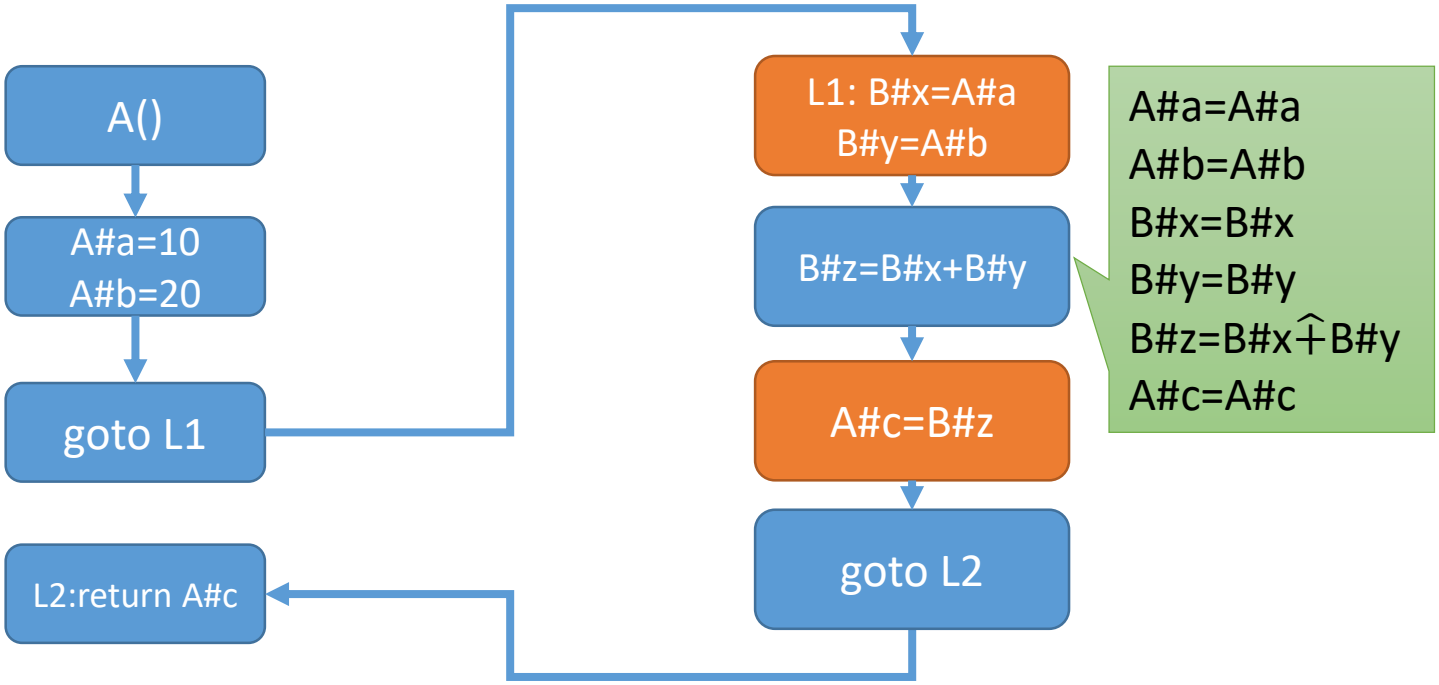
- 把Call/Return语句当成goto语句
- 对本地变量改名，并在调用和返回时添加赋值语句



这样形成的全局控制流图通常称为Super CFG

优化信息传递

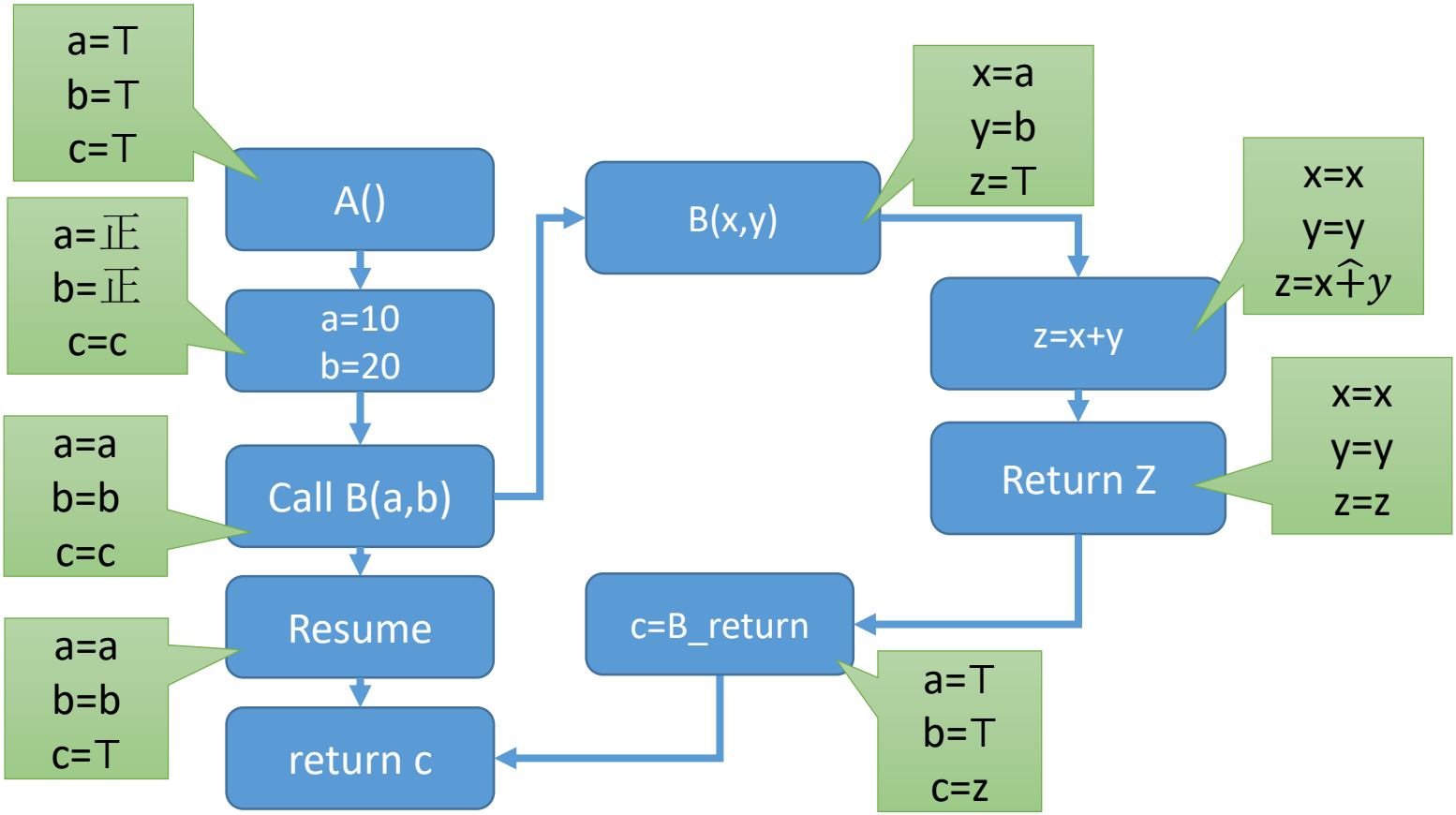
- 以上方案会导致分析B的时候也必须不断传递关于A的局部变量信息



符号分析时的转换函数

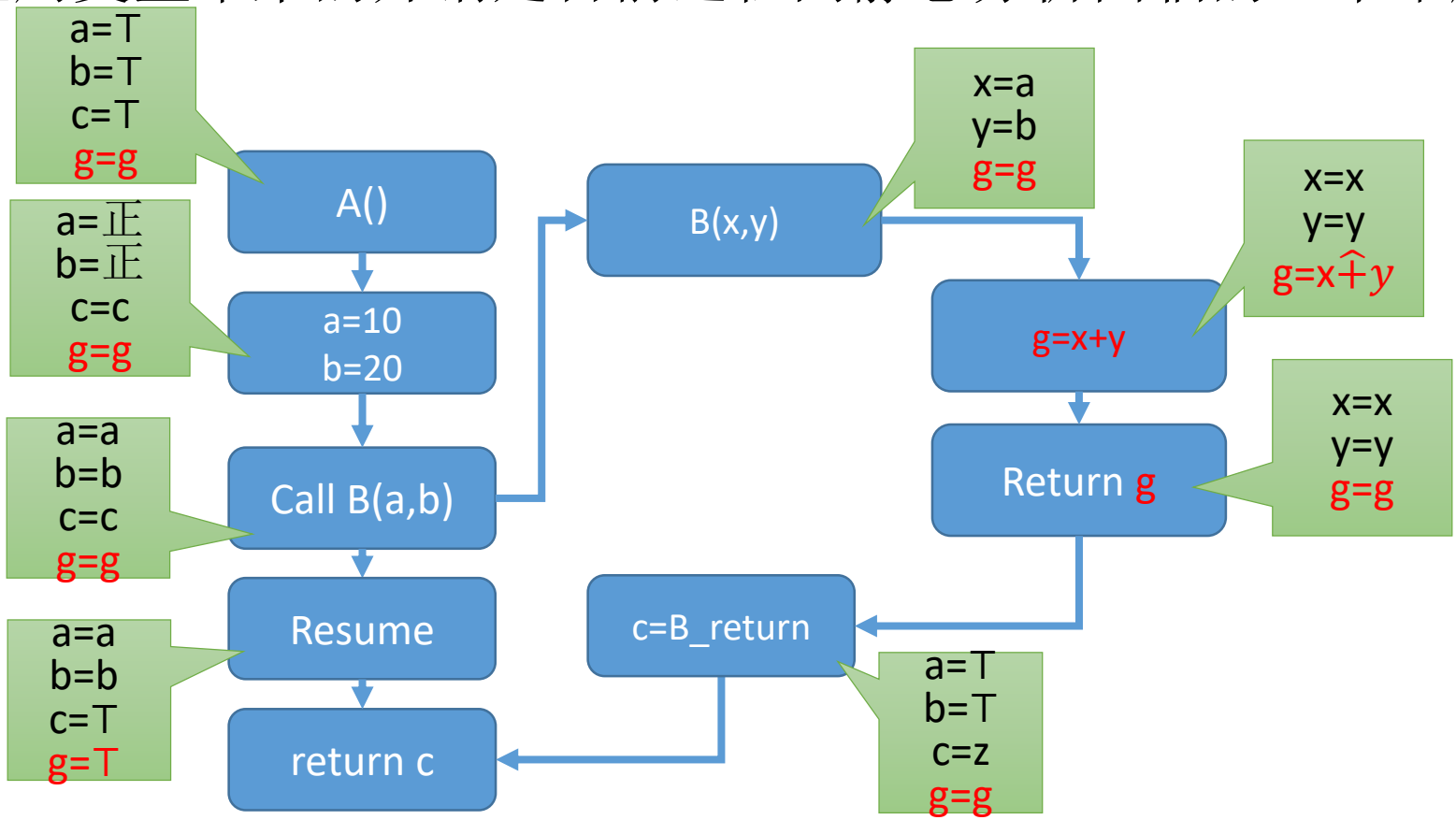
优化信息传递

- 在本地调用与返回之间添加边来传递本地信息



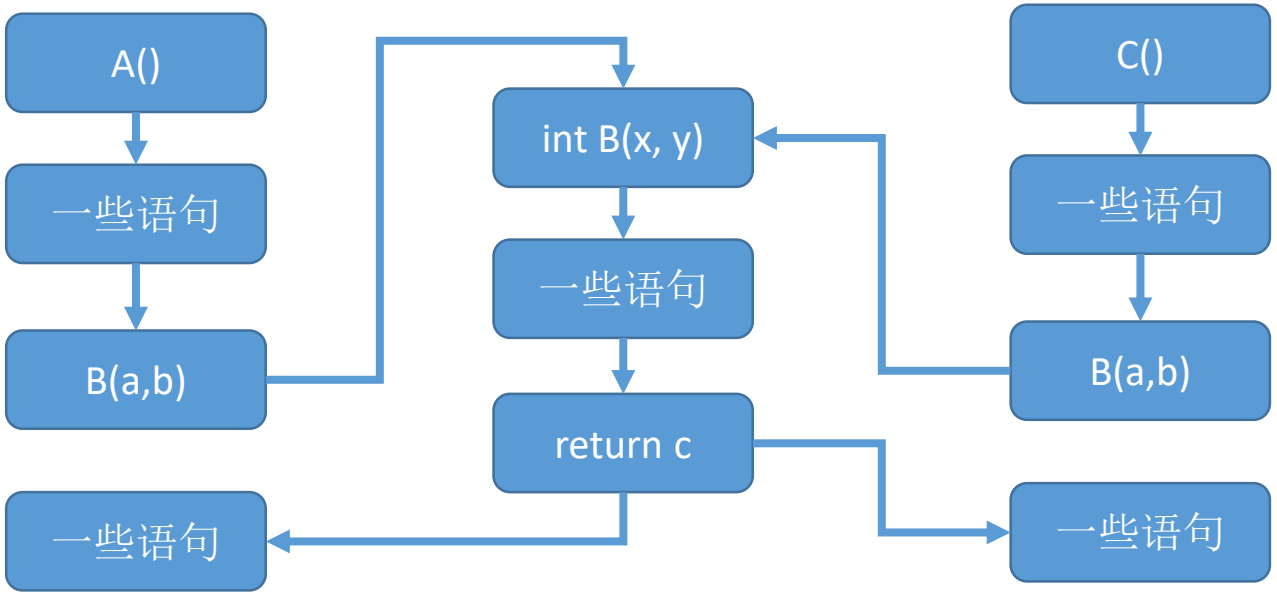
全局变量

- 全局变量需要加到所有使用的过程和这些过程的直接和间接调用者中
- 全局变量带来的开销是目前过程间静态分析面临的一个难题



过程间分析-精度问题

- 上述分析方法是否有精度损失？



- 导致不可能的路径变成可能
- 会将A()函数的分析结果引入C()函数，反之亦然

示例：常量分析

- 判断在某个程序点某个变量的值是否是常量

```
int id(int a) { return a; }
void main() {
  int x = id(100);
  int y = id(200);
}
```

- main执行结束时，x和y都应该是常量，但在super CFG中分析不出来这样的结果

精度损失对比

- 在过程内分析时，由于我们忽略了If语句的条件，同样会导致不可能的路径变成可能
- 过程内分析不精确的条件：存在两个条件互斥
- 过程间分析不精确的条件：一个过程被调用两次
- 后者远比前者普遍
- 在面向对象或者函数语言中，过程调用非常频繁，该方法将导致非常不精确的结果

程序分析的分类-敏感性

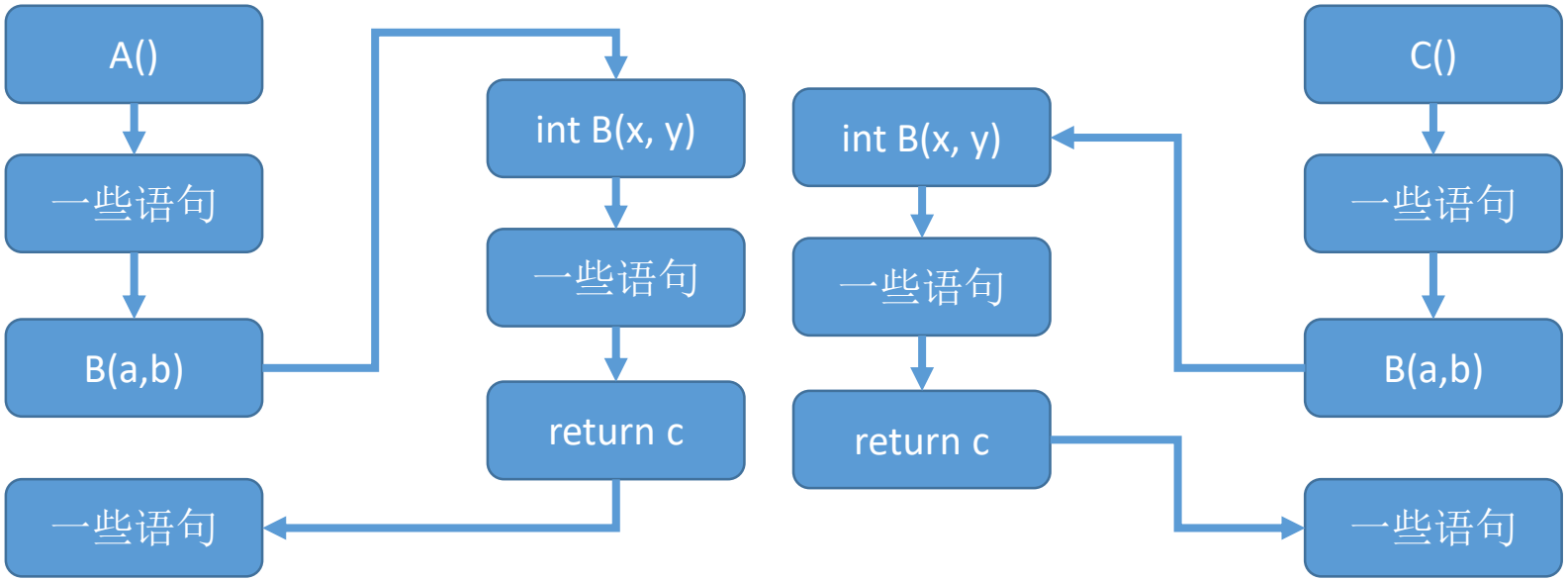
- 一般而言，抽象过程中考虑的信息越多，程序分析的精度就越高，但分析的速度就越慢
- 程序分析中考虑的信息通常用敏感性来表示
  - 流敏感性flow-sensitivity
  - 路径敏感性path-sensitivity
  - 上下文敏感性context-sensitivity
  - 字段敏感性field-sensitivity

上下文敏感性 Context-sensitivity

- 上下文非敏感分析Context-insensitive analysis
  - 在过程调用的时候忽略调用的上下文
- 上下文敏感分析Context-sensitive analysis
  - 在过程调用的时候考虑调用的上下文

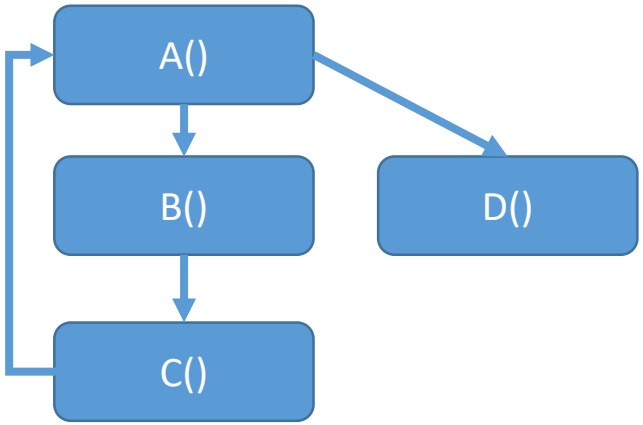
基于克隆的上下文敏感分析
Clone-based Context-Sensitive Analysis

- 给每一处调用创建原函数的一份复制



Call Graph调用图

- 结点表示过程，边表示调用关系



基于克隆的上下文敏感分析

- 练习1：给定下面的程序，请画出克隆之后的函数调用图
  - p() {return q()*q();}
  - q() {return r()+r();}
  - r() {return 100;}

基于克隆的上下文敏感分析

- 练习1：给定下面的程序，请画出克隆之后的函数调用图
  - p() {return q()*q();}
  - q() {return r()+r();}
  - r() {return 100;}

- 练习2：能否画出下面程序的克隆后的函数调用图？
  - p(int n) {return n*p(n-1);}

基于克隆的上下文敏感分析

- 主要问题1：如果有深层次重复函数，就会出现指数爆炸
  - 在实践中，这个问题存在但并不致命
- 主要问题2：递归调用无法处理
  - 在实践中这个问题非常严重

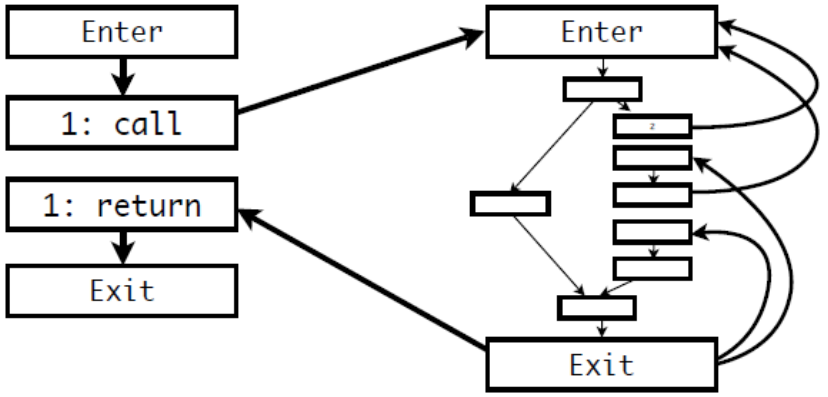
基于克隆的上下文敏感分析

- 解决方案1：只复制没有递归调用的函数
  - 仍然没有解决指数爆炸的问题
  - 如果有一个比较长的递归调用链，则该链上的函数都不能较精确的分析
    - p() { if(…) q(); else r();}
    - q() {if (…) r(); else s();}
    - r() {if (…) p(); else s();}
    - s() {…;p();…}
- 解决方案2：
  - 只使用最近k次调用区分上下文
  - 如果最近k次调用的位置都相同，则不复制，否则复制

Fibonacci函数示例 --Context-Insensitive

```
main() {
  1: fib(7);
}

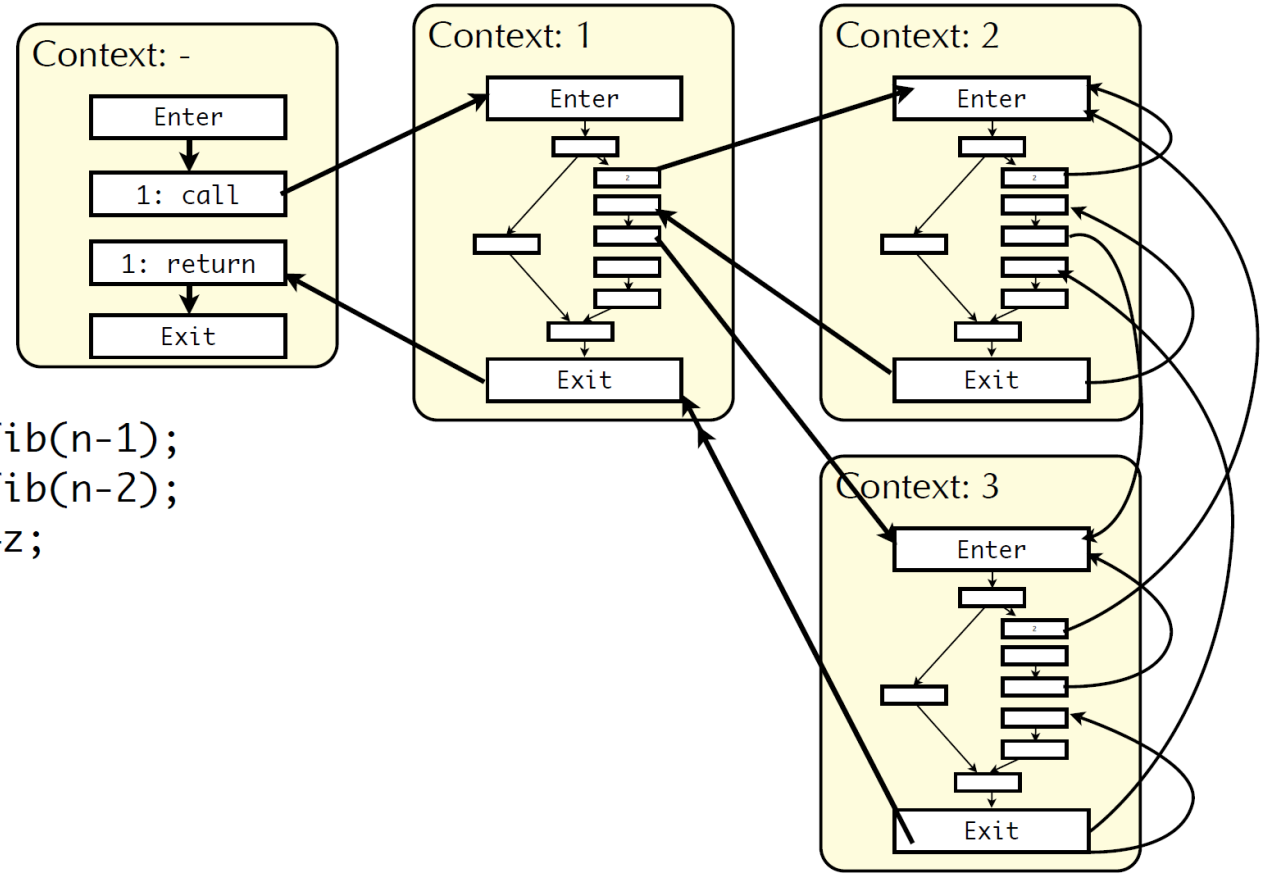
fib(int n) {
  if n <= 1
       x := 1
  else
    2: y := fib(n-1);
    3: z := fib(n-2);
       x:= y+z;
  return x;
}
```



Fibonacci函数示例
--深度1的Clone-based analysis

```
main() {
  1: fib(7);
}

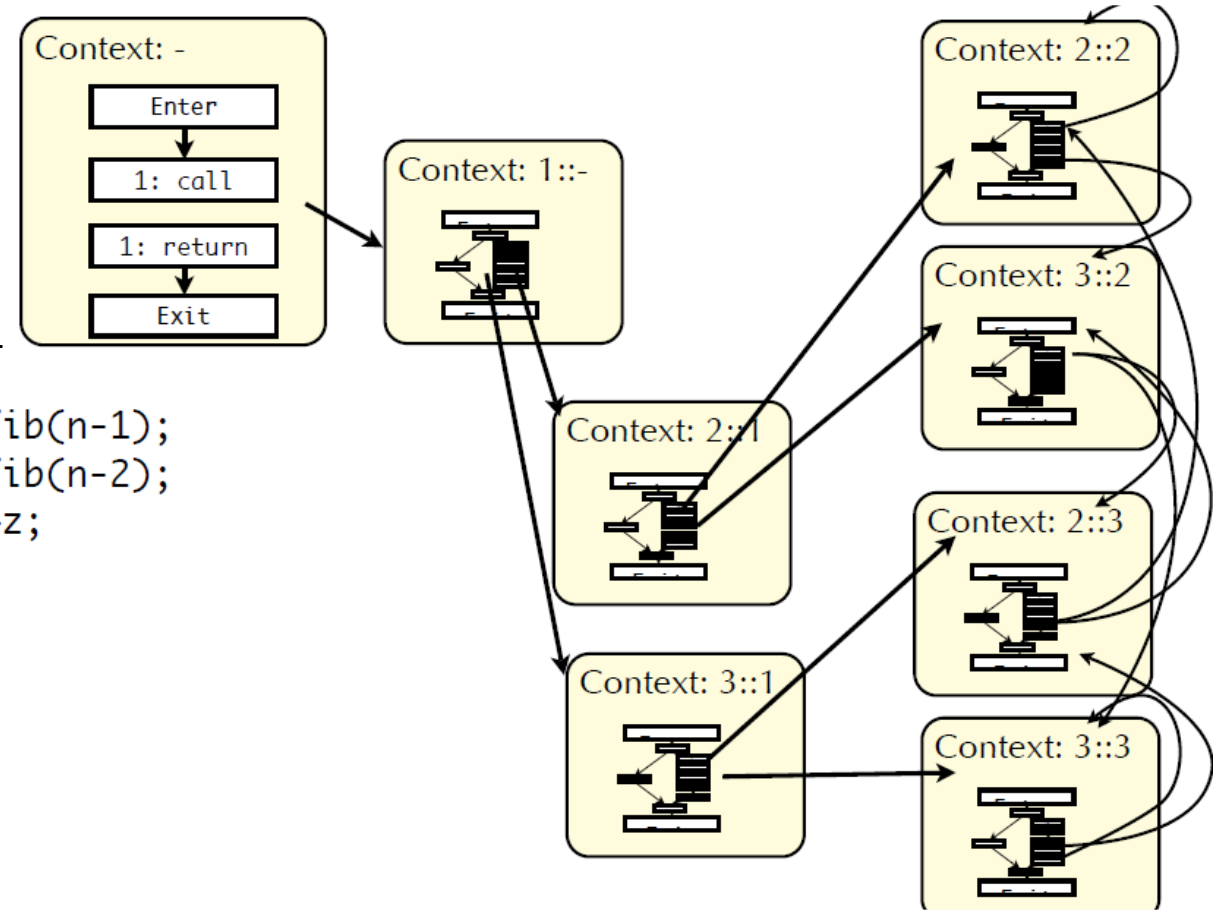
fib(int n) {
  if n <= 1
       x := 1
  else
    2: y := fib(n-1);
    3: z := fib(n-2);
       x:= y+z;
  return x;
}
```



Fibonacci函数示例
--深度2的Clone-based analysis

```
main() {
  1: fib(7);
}

fib(int n) {
  if n <= 1
       x := 1
  else
    2: y := fib(n-1);
    3: z := fib(n-2);
       x:= y+z;
  return x;
}
```



其他可能的上下文

- 基于函数名字而不是调用位置的上下文
  - 在Fib的例子中，2::2和2::3都变成了fib::fib
  - 不如调用位置精确，但能减少复制量
- 基于对象类型的类型的上下文
  - 在OO语言中，对于x.p()的调用，根据x的不同类型区分上下文
  - 可以对OO函数重载进行一些更精细的分析
- 基于系统状态的上下文
  - 根据分析需要，对系统的当前状态进行分类
  - 当函数以不同状态调用时，对函数复制

克隆背后的主要思想

- 当直接在控制流图上分析达不到所要求的精度时，通过复制控制流图的结点来提高分析精度

克隆思想用于过程内分析

- 实例1：
  - 已知c1和c2的条件互斥，给定如下的代码
    - if(c1) x(); else y(); z();if (c2) m(); else n();
  - 该代码上的数据流分析有何不精确？
    - 会考虑x();z();m();这样的执行序列
  - 如何通过克隆手段来解决该不精确？
    - Contex: 前一个if走的是true还是false分支
    - 把代码变换成
      if(c1)
      {x(); z(); if (c2) m(); else n();}
      else
      {y(); z(); if (c2) m() else n();}

克隆思想用于过程内分析

- 实例2：对于循环，可以展开一定层数k，这样对于前k层会有比较精确的结果

- 实例3：根据对系统状态的分类同样可以在语句级别而不是过程级别进行复制

内联Inline

- 另一种实现克隆的方法是内联
- 内联：把被调函数的代码嵌入到调用函数中，对参数进行改名替换
- 实际效果和克隆等效

精确的上下文敏感分析

- 精确的上下文敏感分析
  - =考虑最近任意多次调用的上下文
  - =对于任意分析中考虑的路径，路径中的调用边和返回边全部匹配（称为可行路径）

- 能否做到精确的上下文敏感分析？

精确的上下文敏感分析

- 历史上提出过多种精确上下文敏感分析方法
  - Functional
  - Dataflow Facts-based Summary
  - CFL-reachability
- 近年来的研究逐渐集中在CFL-Reachability上
  - 理解上比较直观
  - 能够优化出高效算法
  - 能覆盖任意的具备分配性的数据流分析（但不能覆盖所有已有方法）
  - 基于该模型讨论清楚了若干可判定性问题

发明人：威斯康星大学
Thomas Reps教授

Dyck-CFL

括号匹配的上下文无关语言

$S \rightarrow \{_1\ S\ \}_1$

$\quad | \ \{_2\ S\ \}_2$

$\quad | \ \ldots$

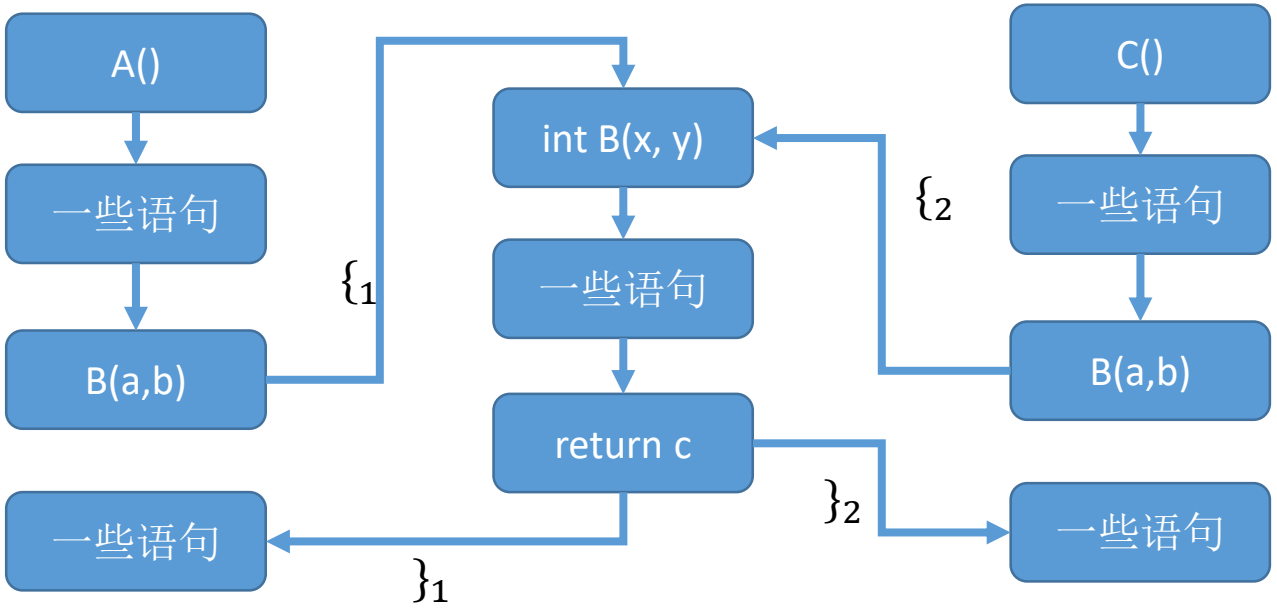
$\quad | \ SS$

$\quad | \ \epsilon$

注意区分上下文无关语言和上下文敏感性中的上下文

Dyck-CFL与上下文敏感性

- 给系统中的每一处调用分配唯一一对括号



- 给定一条路径，如果该路径上的符号组成Dyck-CFL的句子，则该路径是可行路径

复习：数据流分析的分配性 Distributivity

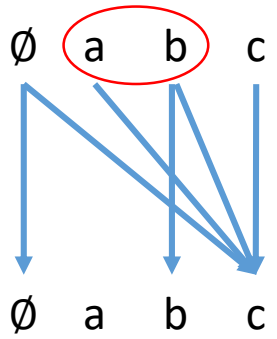
- 什么叫数据流分析的分配性？
- 一个数据流分析满足分配性，如果
  - $\forall v \in V, x, y \in S: f_v(x) \sqcap f_v(y) = f_v(x \sqcap y)$
- 符号分析是否满足分配性？
- GEN/KILL标准型$f(S) = (S - KILL) \cup \text{GEN}$是否满足分配性？

数据流分析的分配性

- 推论
  - 给定任意满足分配性的数据流分析X，其entry结点的初值是I^X。
  - 令Y和Z为仅有初值不同的数据流分析，其中Y和Z的初值分别是I^Y、I^Z，且$I^X = I^Y \sqcap I^Z$。
  - 令DATA_v^X为通过X分析得出的v结点的值
  - 则：$\mathrm{DATA}_v^X = \mathrm{DATA}_v^Y \sqcap \mathrm{DATA}_v^Z$
- 证明
  - 在任意执行路径上，结论成立
  - 根据数据流分配性的性质，在所有执行路径上，结论成立

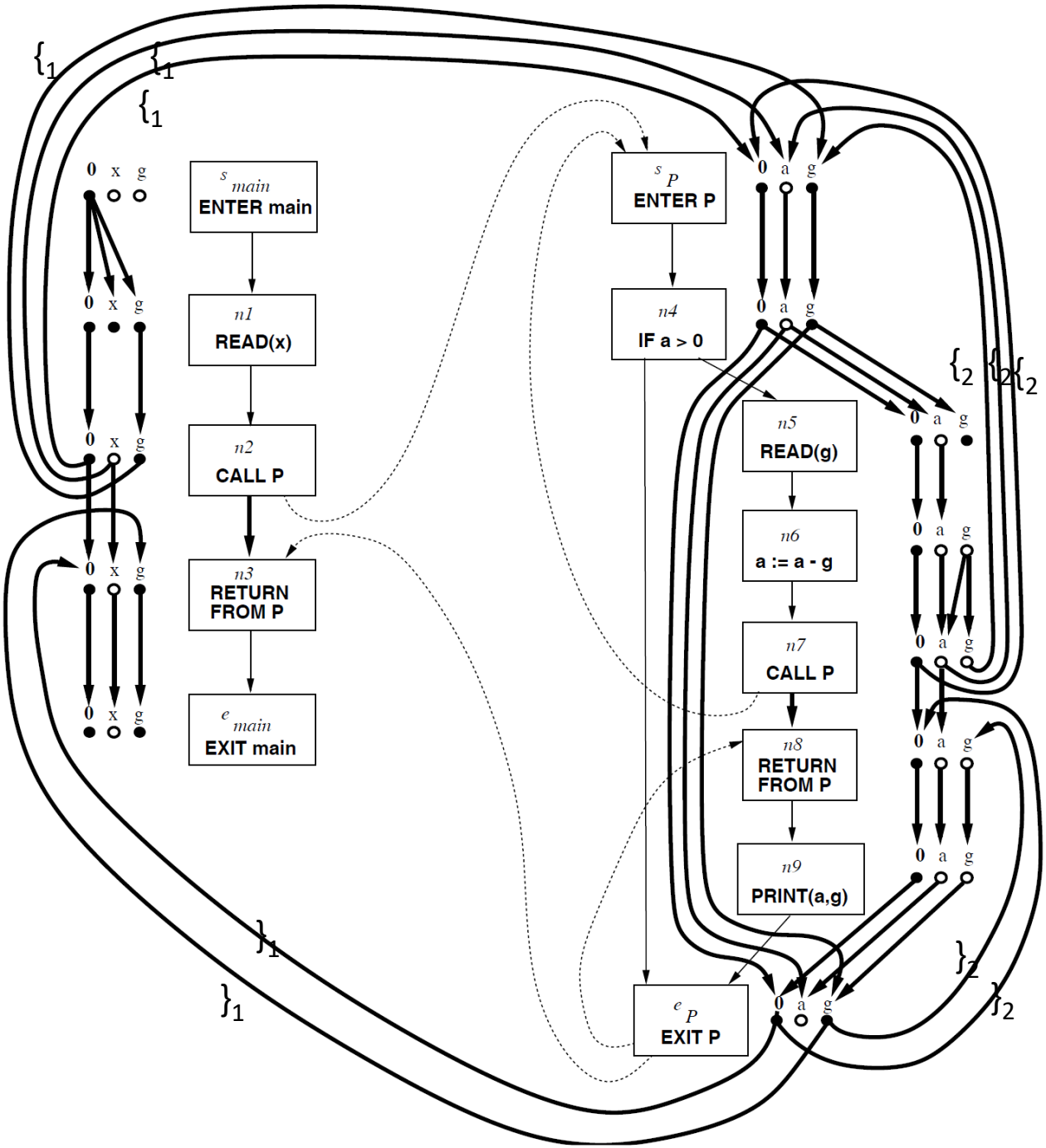
转换函数vs图可达性

- 对于可分配函数f(X)=(X-{a})∪{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
- 结合上页推论我们可以把整个数据流分析转成可达性问题

例：未初始化变量分析


s main
ENTER main
n1
READ(x)
n2
CALL P
n3
RETURN FROM P
e main
EXIT main
s P
ENTER P
n4
IF a > 0
n5
READ(g)
n6
a := a - g
n7
CALL P
n8
RETURN FROM P
n9
PRINT(a,g)
e P
EXIT P
0 x g
0 a g
{1
}1
{2
}2

上下文无关语言可达性问题

- 给定一个图，其中每条边上有标签
- 给定一个用上下文无关文法描述的语言L
- 对于图中任意结点v1、v2，确定是否存在从v1到v2的路径p，使得该路径上的标签组成了L中的句子。

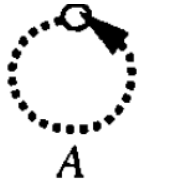
计算方法

- 把原文法改写为右边只有最多两个符号的形式

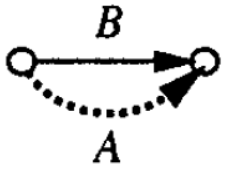
$$
\begin{aligned}
S \rightarrow\ & \{_1 E_1 && E_1 \rightarrow S\}_1 \\
\mid\ & \{_2 E_2 && E_2 \rightarrow S\}_2 \\
\mid\ & \ldots \\
\mid\ & \epsilon
\end{aligned}
$$

计算方法

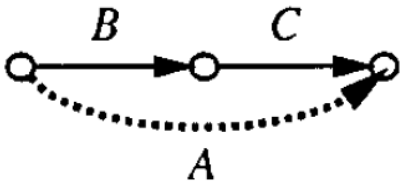
- 按如下三种模式不断添加边，直到没有边需要添加



(a) $A \rightarrow \varepsilon$



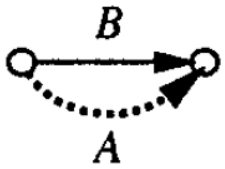
(b) $A \rightarrow B$



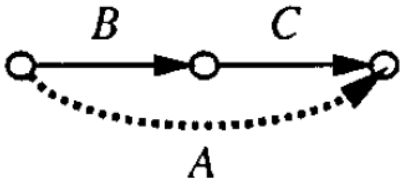
(c) $A \rightarrow B\ C$

例子

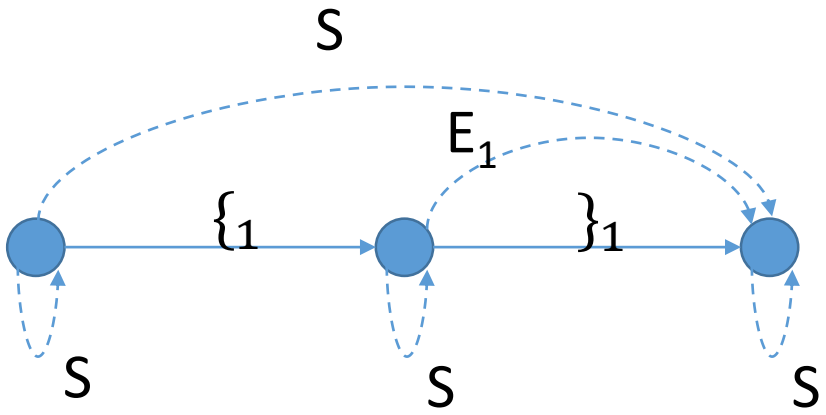
(a) $A \rightarrow \varepsilon$



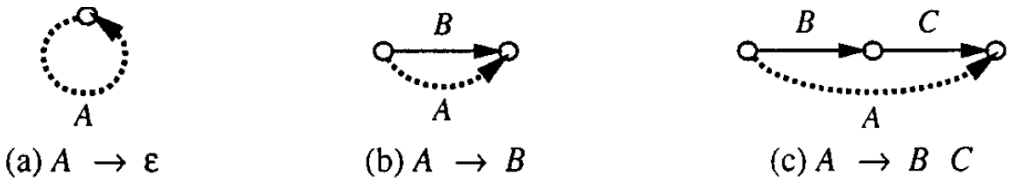
(b) $A \rightarrow B$



(c) $A \rightarrow B\ C$



伪码



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
}

Soundness

- 给定任意控制流图上的可行路径，该路径上算出的结果一定包含在CFL-Reachability算出的结果中。
  - 证明：
    - 根据之前的分析，对于单条路径，CFL-Reachability算出的可达性一定等价于传递函数算出的值
    - 由于该路径是可行路径，所以起结果一定仍然可达

Precision

- 给定任意n，以前n次调用位置作为上下文的基于克隆的分析所产生的结果一定包括CFL-Reachability分析所产生的结果
  - 证明：
    - CFL-Reachability上的任意一条可达路径所产生的结果一定包含在克隆分析中
    - CFL-Reachability分析的结果就是所有可达路径的结果的合并

精确的上下文敏感分析意义有多大？

- 学术界有过多次试验，结论不完全一致
- 总体观点
  - 在面向对象程序中，上下文敏感分析比上下文非敏感分析精确很多
  - 精确的上下文敏感分析比不精确的上下文敏感分析通常更精确一些
  - 精确的上下文敏感分析所额外增加的开销是可接受的

作业（截止10月17日）

- 设计一个过程间的Reaching Definition分析
  - 给出半格定义和变换函数定义，特别是call语句和return语句的变换函数
  - 采用基于克隆的方法分析下面的程序，k至少应该设置成多少才能达到精确的上下文敏感性？
    - ```
      int g;
      main(){
        int a = 10;
        g=a+1;
        h(a); h(a);
      }
      void h(int a) {
        x(a);  g++;  x(a);
      }
      void x(int a) {
        output(a);
      }
      ```
      ```