软件分析

基于摘要的过程间分析

熊英飞
北京大学
2014

Microsoft Research
微软亚洲研究院

复习：基于克隆的过程间分析

- 练习：一个程序有n个方法，每个方法最多在程序中会有m次调用，取最近k次调用作为上下文，问最坏情况下克隆后该程序中方法的个数。

- nm^k

基于摘要的过程间分析 Summary-based Interprocedural Analysis

- 构造Super CFG的开销比较大
- 由于clone/inline的作用，即使一个只依赖本地参数的方法被调用的时候输入参数完全相同，也要被分析两次
  - A(){…;sin(100);…;}
  - B(){…;sin(100);…;}
- 实践中常常采用基于摘要的过程间分析

基于摘要的过程间分析

- 摘要：关于过程行为的抽象信息，使得根据摘要可以直接导出函数调用语句的转换函数或求出转换后的值，而不用跳转到函数内部
  - 通常针对具体分析具体设计
  - 可以预先分析出每个过程的摘要
  - 也可以在分析过程中动态创建摘要

- 摘要通常是上下文敏感的
  - 如果在不同的调用点能通过摘要生成的转换函数得到不同的信息，则摘要是上下文敏感的
  - 也可以是摘要+上下文信息导出转换函数

摘要示例：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);
}
```

- 为id创建摘要
  - 常量->常量
  - 非常量->非常量

摘要实例

- 内存泄露修复的摘要
  - 针对特定malloc语句，每个过程根据可以分成三种类型
    - 必定返回该语句分配的内存，记为Malloc
    - 可能使用该语句分配的内存，记为Use
    - 可能释放该语句分配的内存，记为Free
  - 摘要=一个vector，每个元素是Malloc, Use, Free, None中的一个，分别对应一条malloc语句
  - 之后的分析只需要针对过程内部进行
- 符号分析的摘要
  - 将原函数转换成一个抽象域上的运算
    - 如：f(int a){ int b=a+10; b *= 20; return b; }
    - 摘要为：f(a)=a+正

摘要的构建

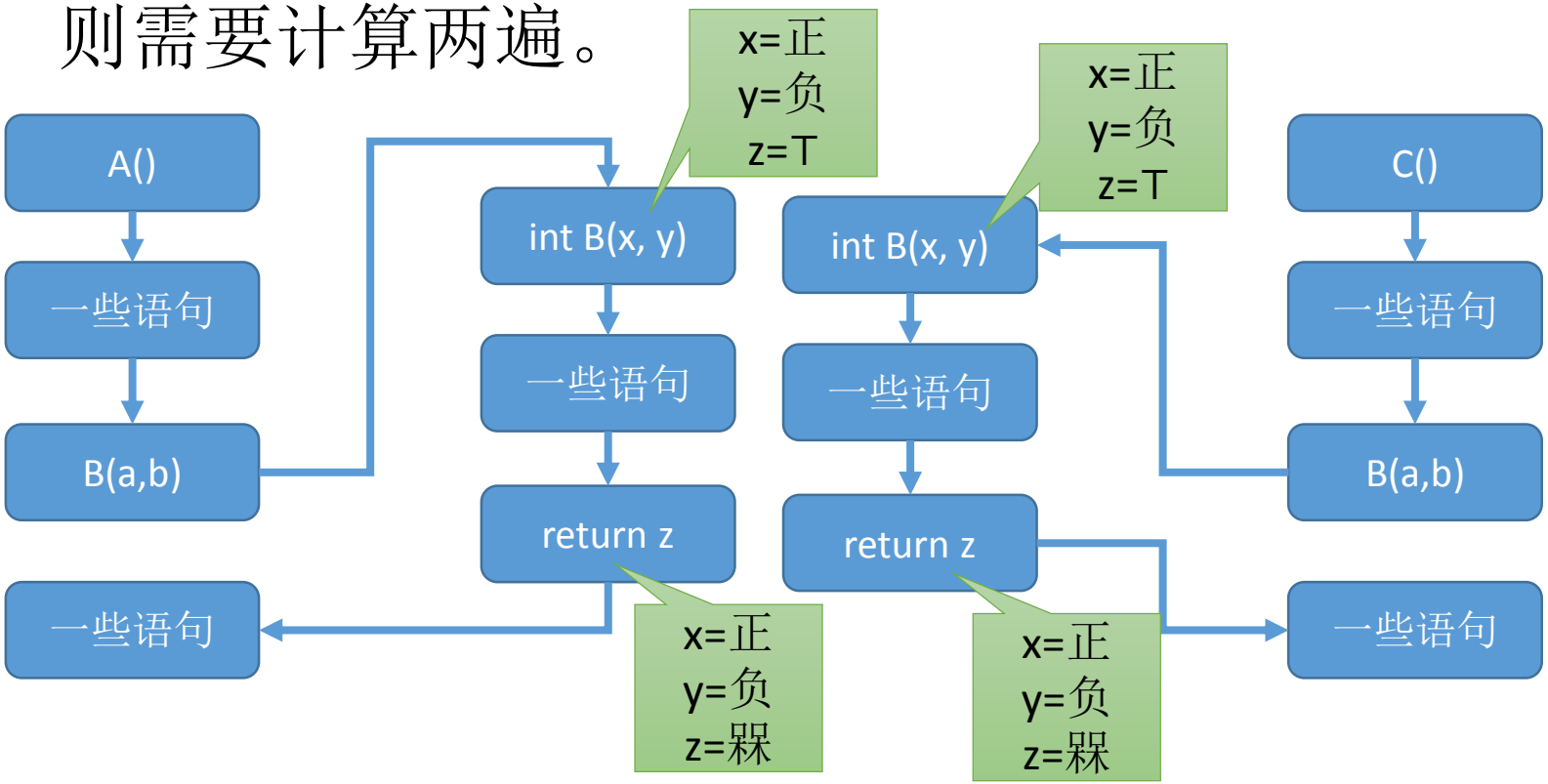
- 一个过程的摘要可能与该过程的调用者和其他调用的过程都有关
  - 假设过程a调用b
  - 被调者依赖调用者：如果a把内存x作为参数n传给了b，并且b使用了参数，则b的摘要中包括Use(x)
  - 调用者依赖被调者：如果b返回一个新分配的内存x，然后a又返回了x，则a的摘要中包括Malloc(x)
- 在调用图上迭代分析：当过程p的摘要变化时，更新调用图上p所有后继和所有前驱的摘要
- 终止性的证明：
  - 摘要构成一个高度有限的半格
  - 每次更新的摘要都比之前要小

基于摘要分析的通用算法

- 给定一个过程内数据流分析算法，如何自动的导出过程间数据流分析算法
- Top-down Analysis自顶向下的分析
  - 又叫tabulating分析
- Bottom-up Analysis自底向上的分析
  - 又叫functional分析

自顶向下的分析

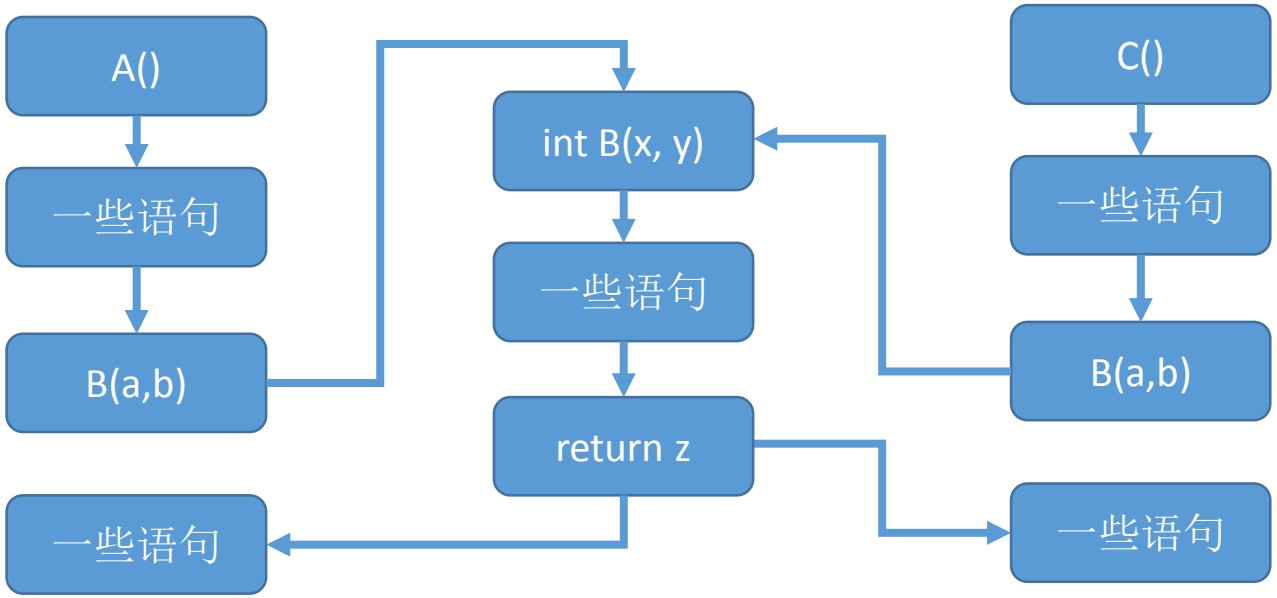
- 在基于克隆的分析，如果同一个过程要分析两遍，则需要计算两遍。



Microsoft Research 微软亚洲研究院

自顶向下的分析

- 采用动态规划/Memoization技术构造摘要
- 过程B的摘要为
  - {x=正,y=负,z=T}=>{x=正,y=负,z=踩}
- 如果初始状态在摘要中存在，则重用摘要，否则克隆B过程并重新计算



动态规划/Memoization的问题

- 粒度太粗
  - 内存消耗大，加速效果不明显
- 仍然需要克隆方法

Microsoft Research
微软亚洲研究院

复习：数据流分析的分配性 Distributivity

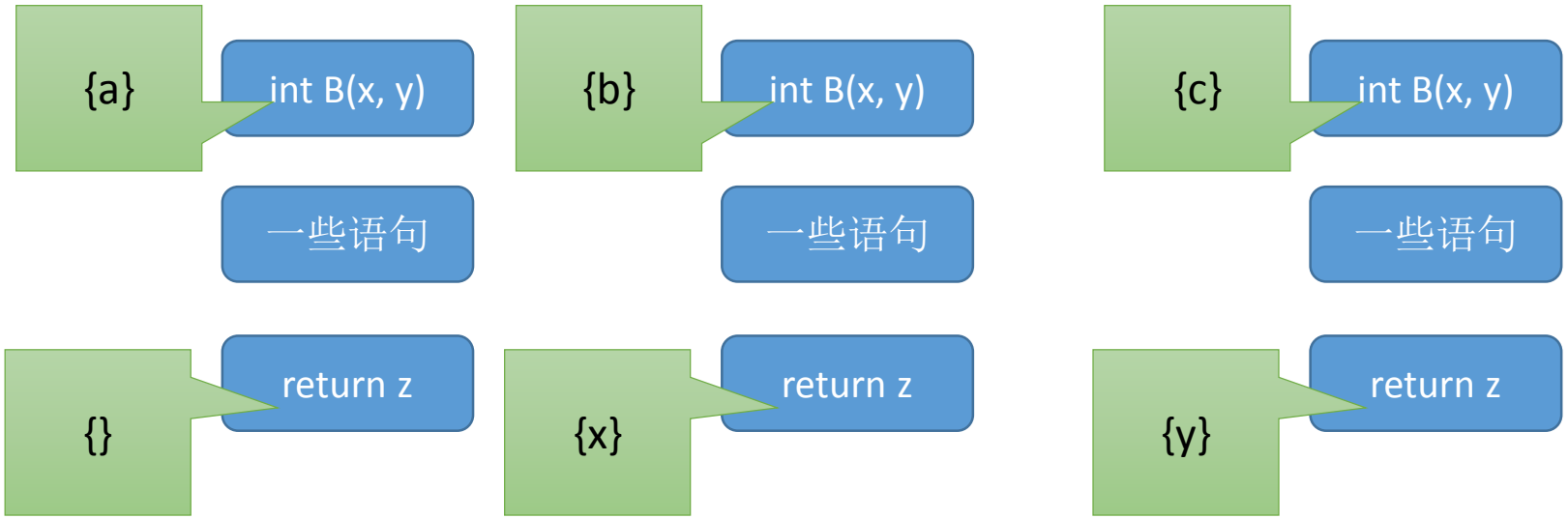
- 什么叫数据流分析的分配性？
- 一个数据流分析满足分配性，如果
  - $\forall v \in V, x, y \in S: f_v(x) \sqcap f_v(y) = f_v(x \sqcap y)$
- 符号分析是否满足分配性？
- GEN/KILL标准型$f(S) = (S - KILL) \cup \text{GEN}$是否满足分配性？

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为仅有初值和X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

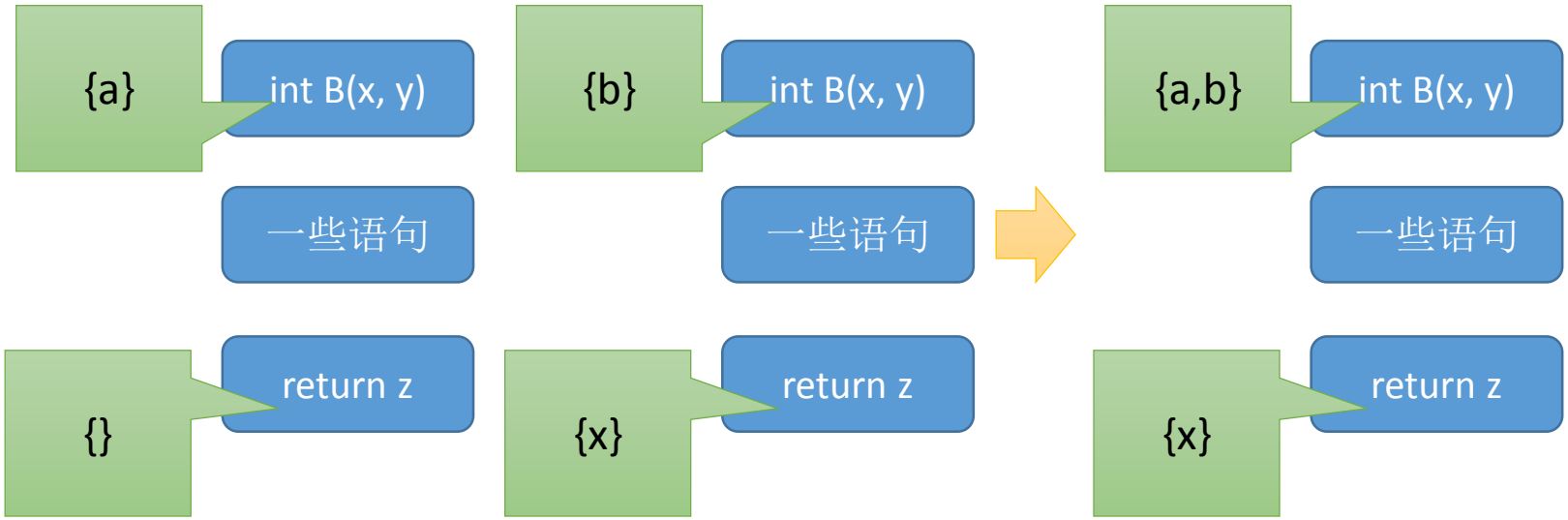
分配性的意义

- 分配性使得我们可以增量计算过程内的数据流分析
- 考虑由集合S的子集和并集操作组成的半格



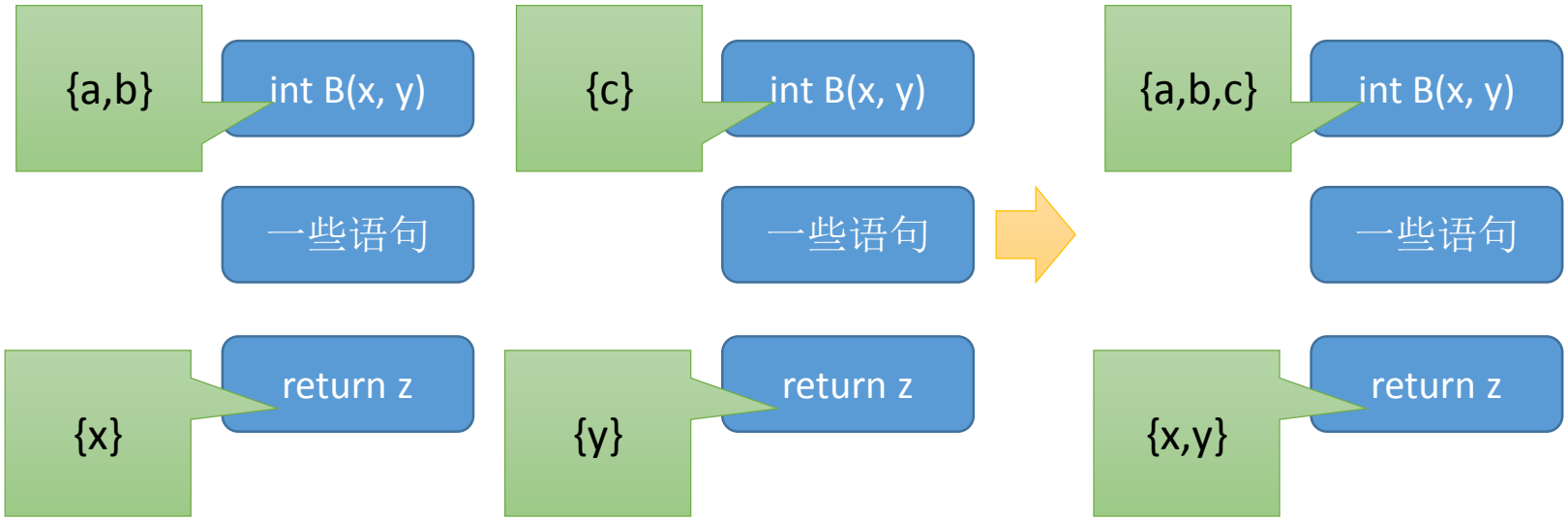
分配性的意义

- 分配性使得我们可以增量计算过程内的数据流分析
- 考虑由集合S的子集和并集操作组成的半格



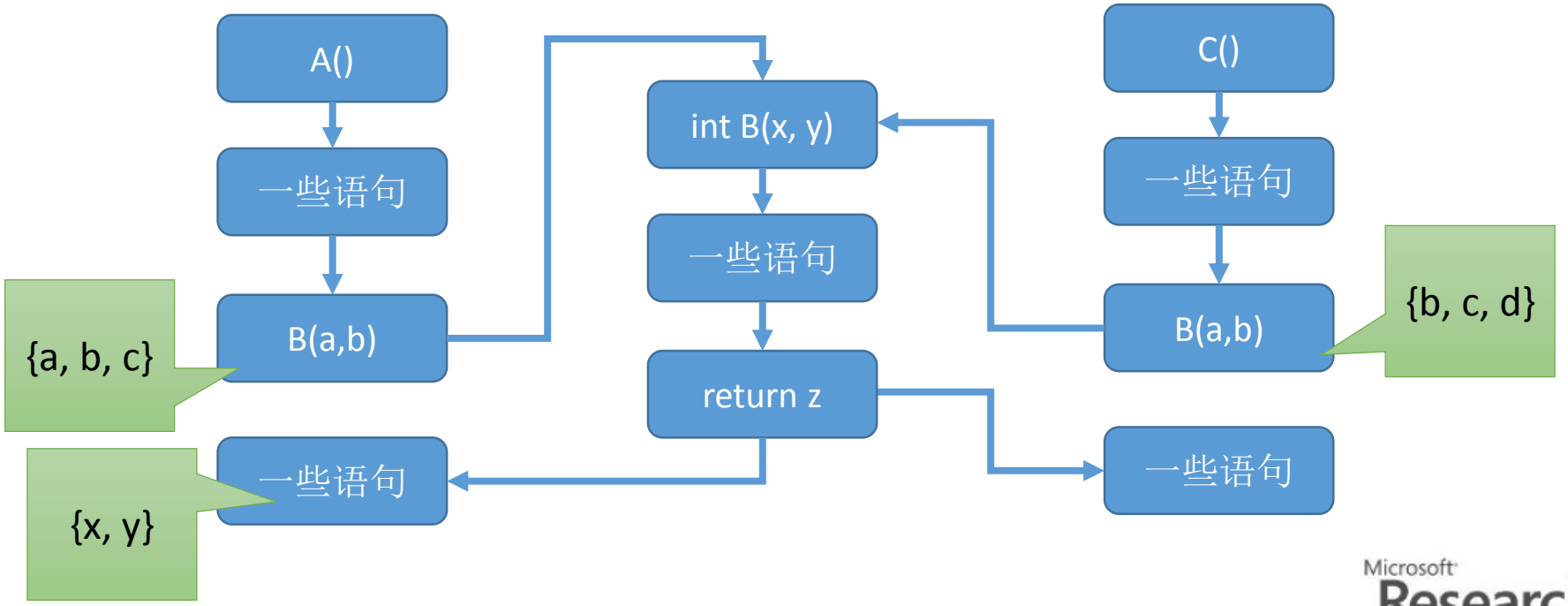
分配性的意义

- 分配性使得我们可以增量计算过程内的数据流分析
- 考虑由集合S的子集和并集操作组成的半格



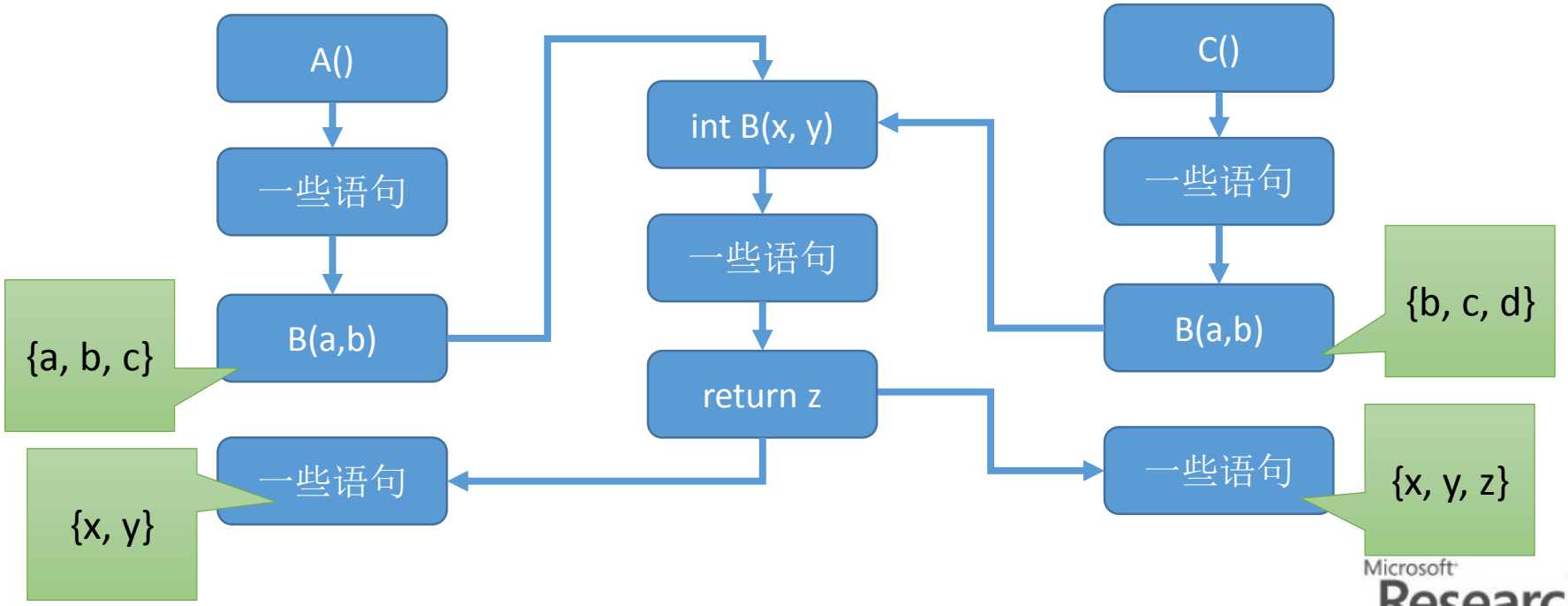
分配性的意义

- 把摘要的粒度从整个集合改为集合中的元素
- B的摘要：
  - a->{}, b->{x, y}, c->{}



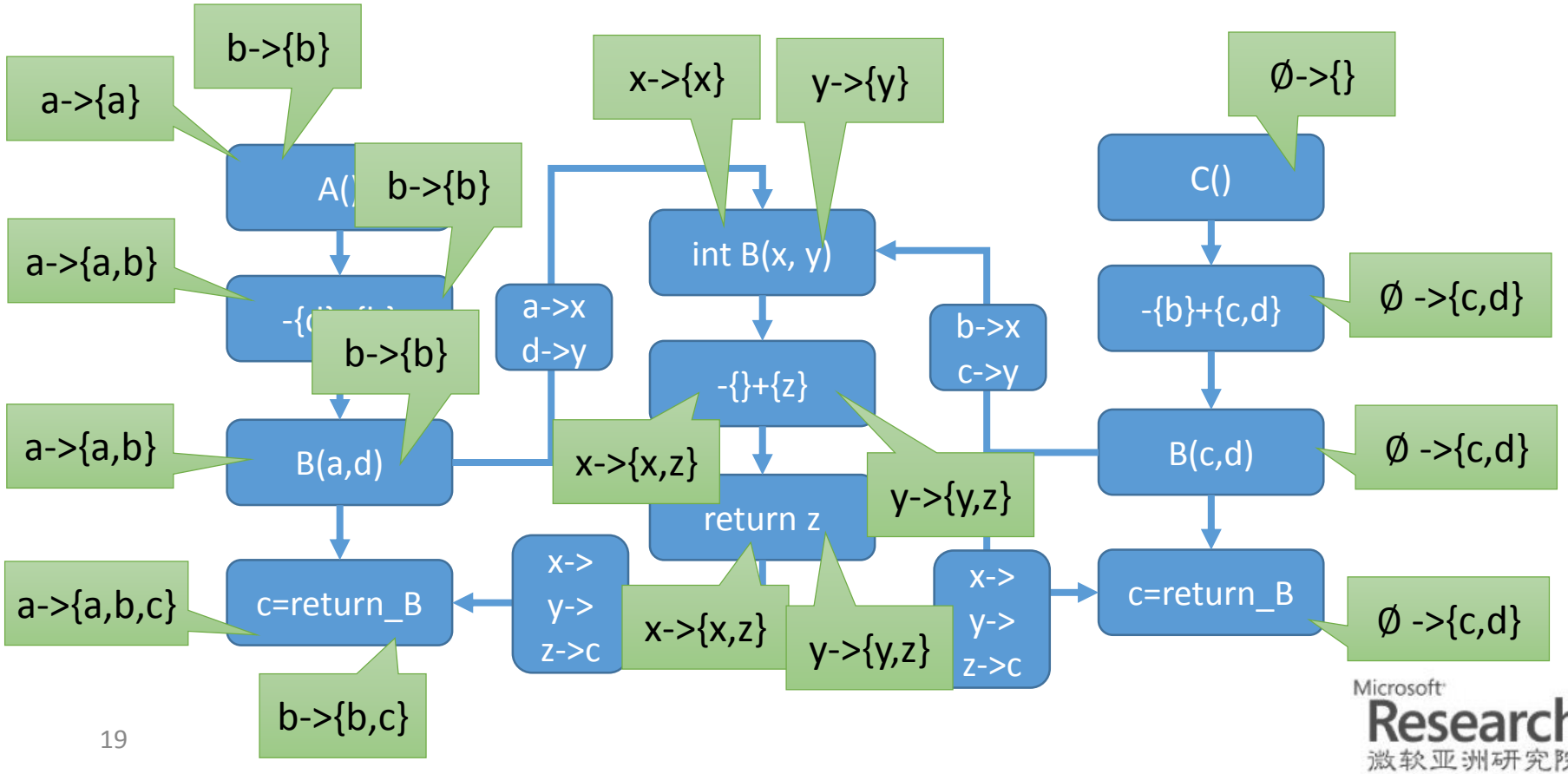
分配性的意义

- 把摘要的粒度从整个集合改为集合中的元素
- B的摘要：
  - a->{}, b->{x, y}, c->{}, d->{z}



计算示例

- 假设A函数entry上的初值是{a, b}，C函数的初值是{}，计算数据流分析结果



Tabulating算法

algorithm Tabulate($G^{\#}_{IP}$)
begin
[1] Let $(N^{\#}, E^{\#}) = G^{\#}_{IP}$
[2] PathEdge := $\{ \langle s_{main}, \mathbf{0} \rangle \rightarrow \langle s_{main}, \mathbf{0} \rangle \}$
[3] WorkList := $\{ \langle s_{main}, \mathbf{0} \rangle \rightarrow \langle s_{main}, \mathbf{0} \rangle \}$
[4] SummaryEdge := $\emptyset$
[5] ForwardTabulateSLRPs()
[6] **for** each $n \in N^{*}$ **do**
[7] $X_n := \{ d_2 \in D \mid \exists d_1 \in (D \cup \{ \mathbf{0} \}) \text{ such that } \langle s_{procOf(n)}, d_1 \rangle \rightarrow \langle n, d_2 \rangle \in \text{PathEdge} \}$
[8] **od**
end

Tabulating算法


```
      procedure Propagate(e)
      begin
[9]     if e ∉ PathEdge then Insert e into PathEdge; Insert e into WorkList fi
      end

      procedure ForwardTabulateSLRPs()
      begin
[10]    while WorkList ≠ ∅ do
[11]      Select and remove an edge ⟨s_p, d_1⟩ → ⟨n, d_2⟩ from WorkList
[12]      switch n
[13]        case n ∈ Call_p :
[14]          for each d_3 such that ⟨n, d_2⟩ → ⟨s_calledProc(n), d_3⟩ ∈ E# do
[15]            Propagate(⟨s_calledProc(n), d_3⟩ → ⟨s_calledProc(n), d_3⟩)
[16]          od
[17]          for each d_3 such that ⟨n, d_2⟩ → ⟨returnSite(n), d_3⟩ ∈ (E# ∪ SummaryEdge) do
[18]            Propagate(⟨s_p, d_1⟩ → ⟨returnSite(n), d_3⟩)
[19]          od
[20]        end case
```

Tabulating算法


```
[21]        case n = e_p :
[22]          for each c ∈ callers(p) do
[23]            for each d_4, d_5 such that ⟨c, d_4⟩ → ⟨s_p, d_1⟩ ∈ E# and ⟨e_p, d_2⟩ → ⟨returnSite(c), d_5⟩ ∈ E# do
[24]              if ⟨c, d_4⟩ → ⟨returnSite(c), d_5⟩ ∉ SummaryEdge then
[25]                Insert ⟨c, d_4⟩ → ⟨returnSite(c), d_5⟩ into SummaryEdge
[26]                for each d_3 such that ⟨s_procOf(c), d_3⟩ → ⟨c, d_4⟩ ∈ PathEdge do
[27]                  Propagate(⟨s_procOf(c), d_3⟩ → ⟨returnSite(c), d_5⟩)
[28]                od
[29]              fi
[30]            od
[31]          od
[32]        end case
[33]        case n ∈ (N_p − Call_p − { e_p }) :
[34]          for each ⟨m, d_3⟩ such that ⟨n, d_2⟩ → ⟨m, d_3⟩ ∈ E# do
[35]            Propagate(⟨s_p, d_1⟩ → ⟨m, d_3⟩)
[36]          od
[37]        end case
[38]      end switch
[39]    od
      end
```


Microsoft Research
微软亚洲研究院

自顶向下的摘要小结

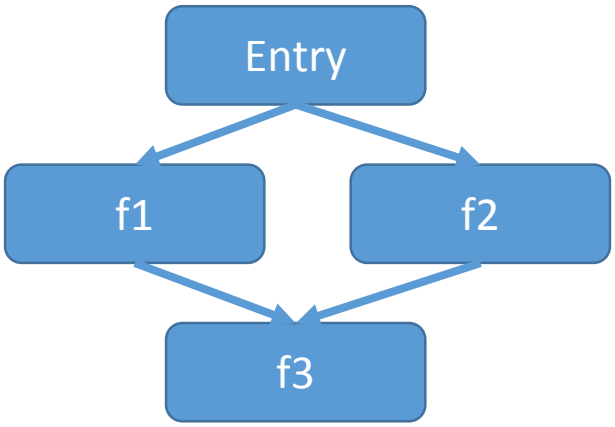
- 要求数据流分析为可分配的
- 利用分配性将集合上的计算分解为元素和空集上的计算
- 摘要用于保存元素上的计算结果，避免相同值上的重复计算
- 采用Worklist算法逐步增量计算出所有的值

计算过程为交集的情况

- 只需要利用数学上的对偶性将其转换成并集的问题
- 课后作业

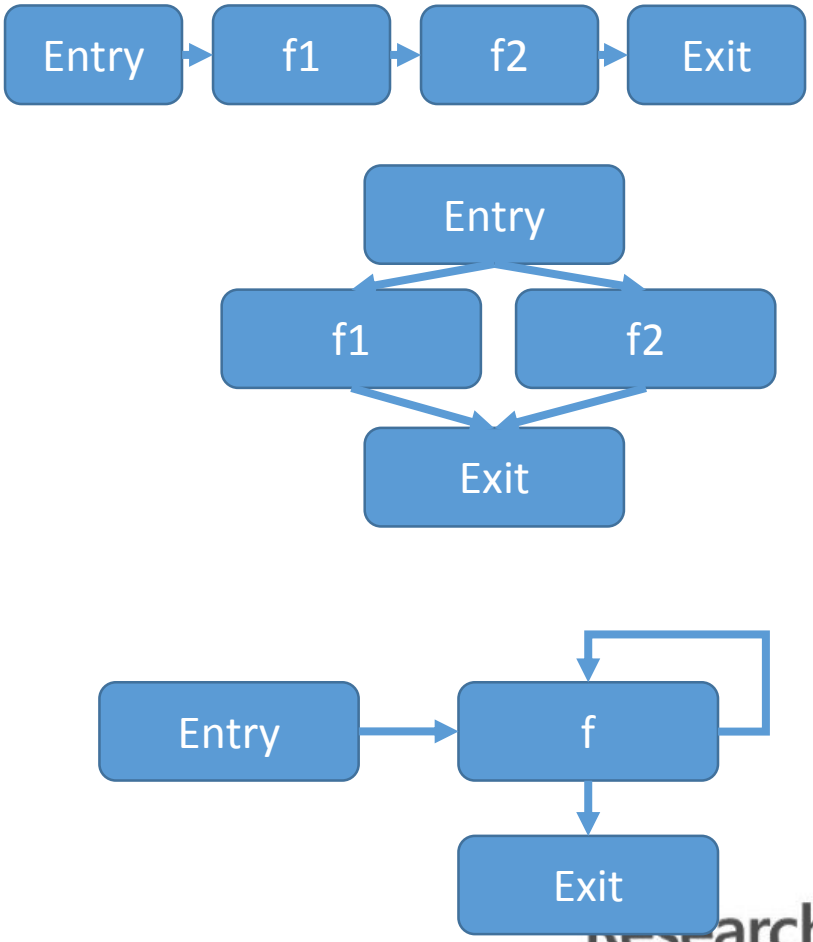
自底向上的分析

- 给定CFG，数据流分析算法实际上定义了从entry到exit的一个转换函数
  - f(S)=f3(f1(S)⊓f2(S))
- 该函数可以作为过程的摘要。但是，这种摘要的执行效率和克隆完全等价
- 如果对摘要函数进行化简，则摘要能提高分析效率



合并操作符

- 顺序：函数组合操作
  - $f_2 \circ f_1(x) = f_2(f_1(x))$
- 选择：函数交汇操作
  - $f_1 \sqcap f_2(x) = f_1(x) \sqcap f_2(x)$
- 循环：函数闭包操作
  - $f^* = \sqcap_{n \geq 0} f^n$



合并操作符-gen/kill标准型

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

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

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

合并操作符-gen/kill标准型

$$
\begin{aligned}
f^2(x) &= f\big(f(x)\big) \\
&= \Big(gen \cup \big((gen \cup (x - kill)\big) - kill\Big) \\
&= gen \cup (x - kill) \\
f^3(x) &= f\big(f^2(x)\big) \\
&= gen \cup (x - kill)
\end{aligned}
$$

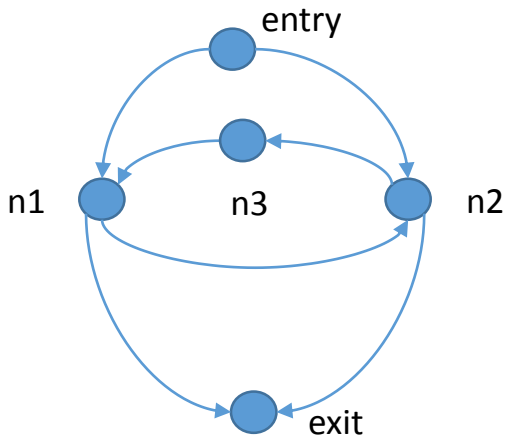
$$
\begin{aligned}
f^*(x) &= I \sqcap f^1(x) \sqcap f^2(x) \sqcap \ldots \\
&= x \cup \big(gen \cup (x - kill)\big) \\
&= gen \cup x
\end{aligned}
$$

控制流图分解

- 如何把控制流图分解成顺序、选择、循环等基本结构？

- 常被称为控制流分析（Control flow analysis），基于区域的分析（Region-based analysis）
- 控制流分析也指构建调用图的过程

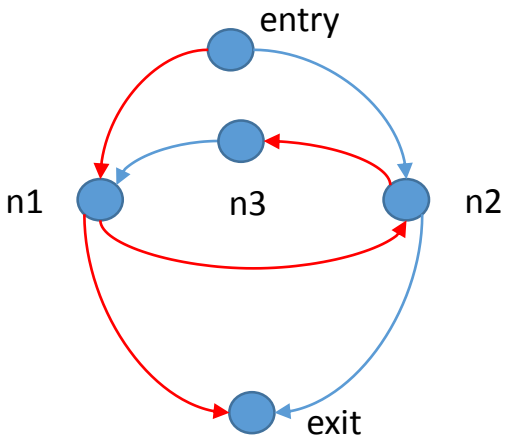
控制流图分解算法



Microsoft Research
微软亚洲研究院

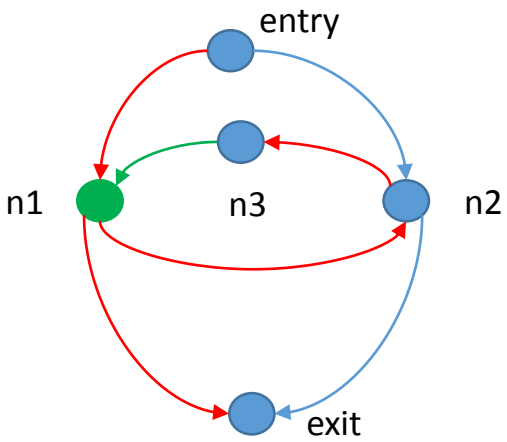
控制流图分解算法

- 从entry出发做一个深度优先遍历，得到生成树



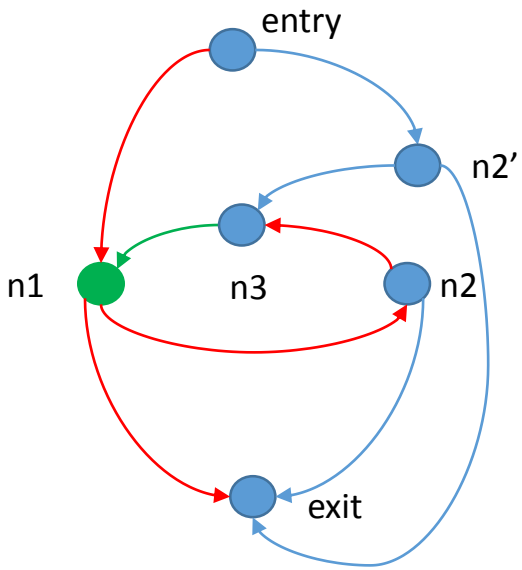
控制流图分解算法

- 检查剩下的边，如果有一条边在生成树中从较深层次的结点指向较浅层次的结点，则该边标示了一个循环，称为回边，该边的尾结点称为循环入口



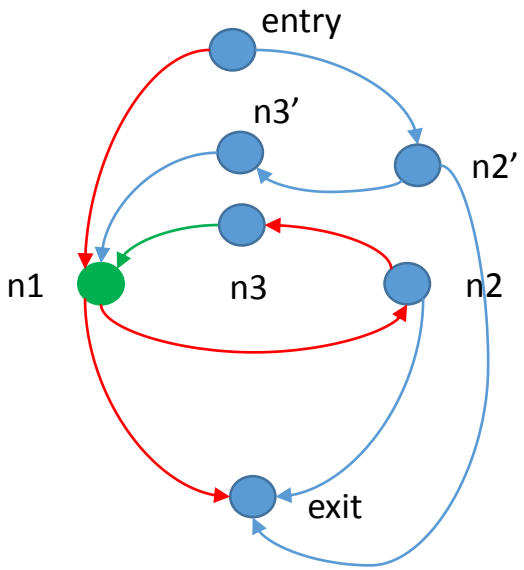
控制流图分解算法

- 对所有从循环外部指向循环内非入口结点n的边，复制n使得n的副本数等于入边数，每个n的副本连接一个入边，同时副本都有原来所有的出边。



控制流图分解算法

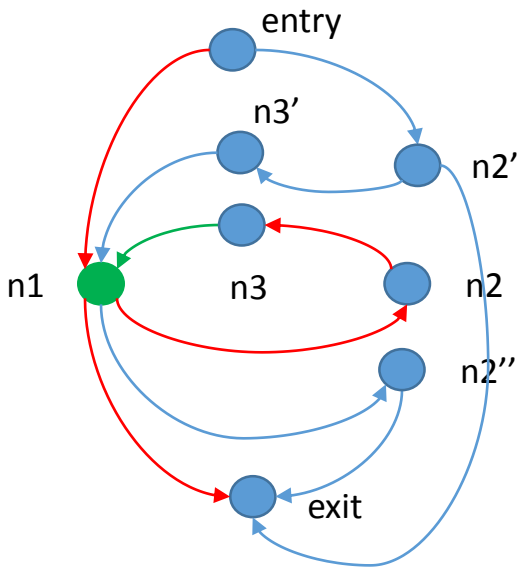
- 对所有从循环外部指向循环内非入口结点n的边，复制n使得n的副本数等于入边数，每个n的副本连接一个入边，同时副本都有原来所有的出边。



Microsoft Research 微软亚洲研究院

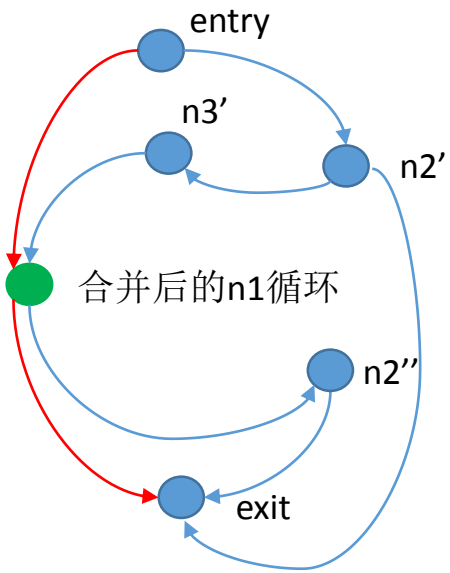
控制流图分解算法

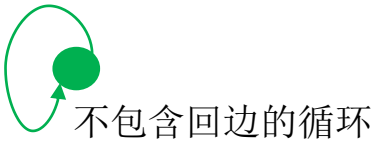
- 对于从循环非入口结点n连向循环外的边，复制n使得n的副本数等于出边数，每个n的副本连接一个出边，同时副本都有原来所有的入边。

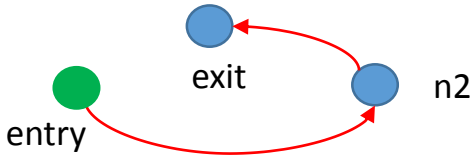


控制流图分解算法

- 分解后，流图由以下两种基本结构递归组成：
  - 包含entry和exit的有向无环图（内部循环看做循环入口一个结点）
  - 由一个结点组成的环

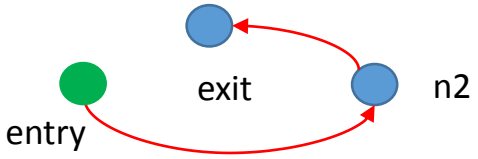


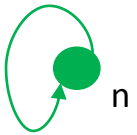




摘要构造算法

- 有向无环图: 通过下式递归计算从entry结点到某个结点v的传递函数g_v，最后取g_{exit}作为该图的摘要
  - $g_v = f_v \circ \sqcap_{w \in pred(v)} g_w$
  - $g_{entry} = ID$
- 由一个结点组成的环：
  - $g = g_n^*$





自底向上的摘要小结

- 需要提供三种转换函数合并操作
  - 组合
  - 交汇
  - 传递闭包
- 将过程中的转换函数合并成一个大的转换函数，减少过程内部的重复计算

对比自顶向下和自底向上的分析
一个较复杂的例子

摘自Xin Zhang的胶片

Microsoft Research
微软亚洲研究院

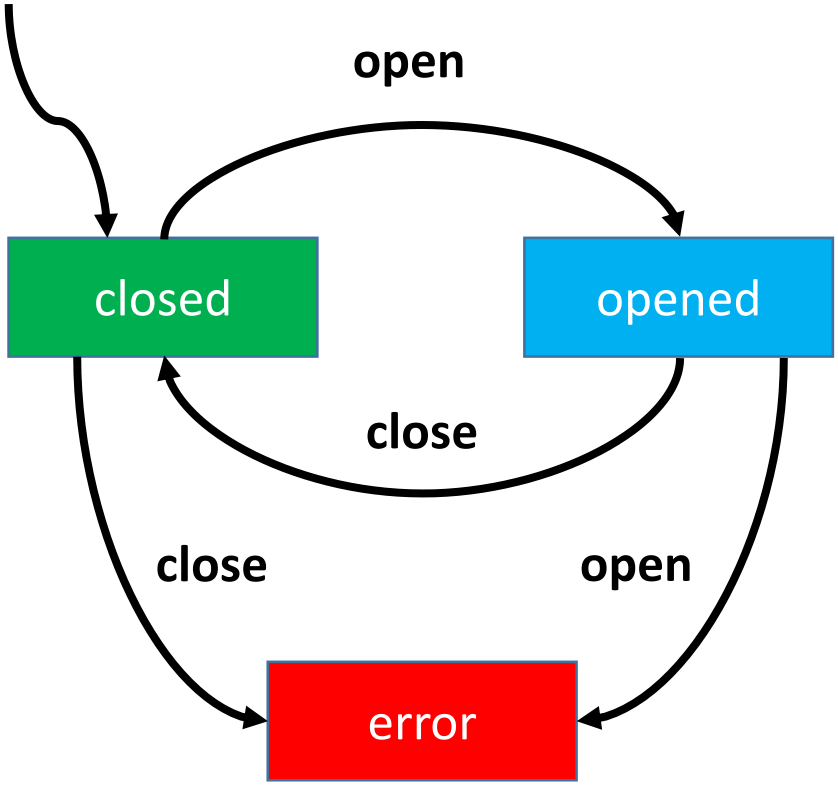
Typestate analysis example [Fink et al. ISSTA'06]

```
main() {
    v1 = new File(); // h1
p1: foo(v1);




}

foo(File f) {
    f.open();
    f.close();
}
```



Top-down approach

Allocation site

```
m
     v   = new File(); // h1
     {(h1, closed, {v1}, ∅)}
p1:  foo(v1);

     v2 = new File(); // h2

p2:  foo(v2);

     v3 = new File(); // h3

p3:  foo(v3);

}
```

```
foo(File f) {

     f.open();

     f.close();

}
```

Microsoft Research 微软亚洲研究院

Top-down approach

Type-state

```
main
    v1 = new File(); // h1
    {(h1, closed, {v1}, ∅)}
p1: foo(v1);

    v2 = new File(); // h2

p2: foo(v2);

    v3 = new File(); // h3

p3: foo(v3);

}
```

```
foo(File f) {

    f.open();

    f.close();

}
```


Microsoft Research 微软亚洲研究院

Top-down approach

Must-alias accesspath set

```
main(
    v1 = new File(); // h1
    {(h1, closed, {v1}, ∅)}
p1: foo(v1);

    v2 = new File(); // h2

p2: foo(v2);

    v3 = new File(); // h3

p3: foo(v3);

}
```

```
foo(File f) {

    f.open();

    f.close();

}
```


Microsoft Research 微软亚洲研究院

Top-down approach

Must-not-alias accesspath set

```
main()
    v1 = new File(); // h1
    {(h1, closed, {v1}, ∅)}
p1: foo(v1);

    v2 = new File(); // h2

p2: foo(v2);

    v3 = new File(); // h3

p3: foo(v3);

}
```

```
foo(File f) {

    f.open();

    f.close();

}
```


Microsoft Research 微软亚洲研究院

Top-down approach

```
main() {
    v1 = new File(); // h1
    {(h1, closed, {v1}, ∅)}
p1: foo(v1);

    v2 = new File(); // h2

p2: foo(v2);

    v3 = new File(); // h3

p3: foo(v3);

}
```

```
foo(File f) {

    f.open();

    f.close();

}
```

Top-down approach


```
main() {
    v1 = new File(); // h1
    {(h1, closed, {v1}, ∅)}
p1: foo(v1);

    v2 = new File(); // h2

p2: foo(v2);

    v3 = new File(); // h3

p3: foo(v3);

}
```

Top-down summaries

$(h_1, closed, \{f\}, \emptyset) \rightarrow \{(h_1, closed, \{f\}, \emptyset)\}\ [T_1]$

T_1

```
foo(File f) {
    {(h1, closed, {f}, ∅)}
    f.open();
    {(h1, opened, {f}, ∅)}
    f.close();
    {(h1, closed, {f}, ∅)}
}
```

Top-down approach

Low Reusability

Top-down summaries

$(h_1, closed, \{f\}, \emptyset) \rightarrow \{(h_1, closed, \{f\}, \emptyset)\}$ $[T_1]$

$(h_1, closed, \emptyset, \{f\}) \rightarrow \{(h_1, closed, \emptyset, \{f\})\}$ $[T_2]$

$(h_2, closed, \{f\}, \emptyset) \rightarrow \{(h_2, closed, \{f\}, \emptyset)\}$ $[T_3]$

$(h_2, closed, \emptyset, \{f\}) \rightarrow \{(h_2, closed, \emptyset, \{f\})\}$ $[T_4]$

$(h_3, closed, \{f\}, \emptyset) \rightarrow \{(h_3, closed, \{f\}, \emptyset)\}$ $[T_5]$

```
main() {
    v1 = new File(); // h1
```
$\{(h_1, closed, \{v_1\}, \emptyset)\}$
```
p1: foo(v1);
```
$\{(h_1, closed, \{v_1\}, \emptyset)\}$
```
    v2 = new File(); // h2
```
$\{(h_1, closed, \{v_1\}, \emptyset), (h_2, closed, \{v_2\}, \emptyset)\}$
```
p2: foo(v2);
```
$\{(h_1, closed, \{v_1\}, \emptyset), (h_2, closed, \{v_2\}, \emptyset)\}$
```
    v3 = new File(); // h3
```
T_2 $\{(h_1, closed, \{v_1\}, \emptyset), (h_2, closed, \{v_2\}, \emptyset), (h_3, closed, \{v_3\}, \emptyset)\}$
```
p3: foo(v3);
```
$\{(h_1, closed, \{v_1\}, \emptyset), (h_2, closed, \{v_2\}, \emptyset), (h_3, closed, \{v_3\}, \emptyset)\}$
```
}
```

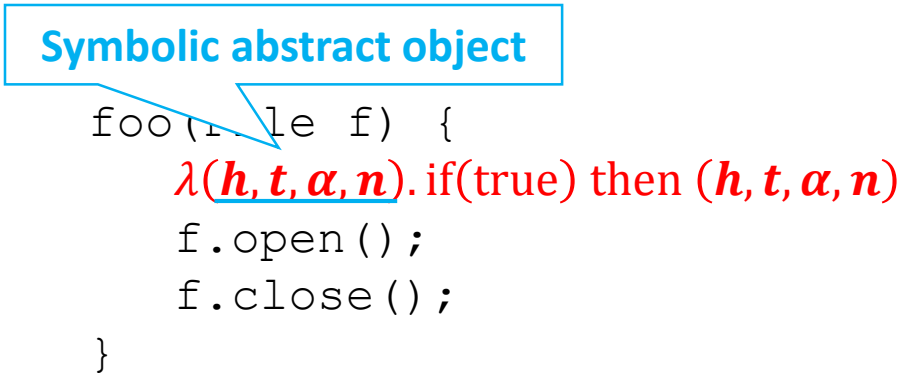
```
foo(File f) {
    f.open();f.close();
}
```

Bottom-up approach


```
foo(File f) {
    λ(h, t, α, n). if(true) then (h, t, α, n)
    f.open();
    f.close();
}
```


Microsoft Research
微软亚洲研究院

Bottom-up approach



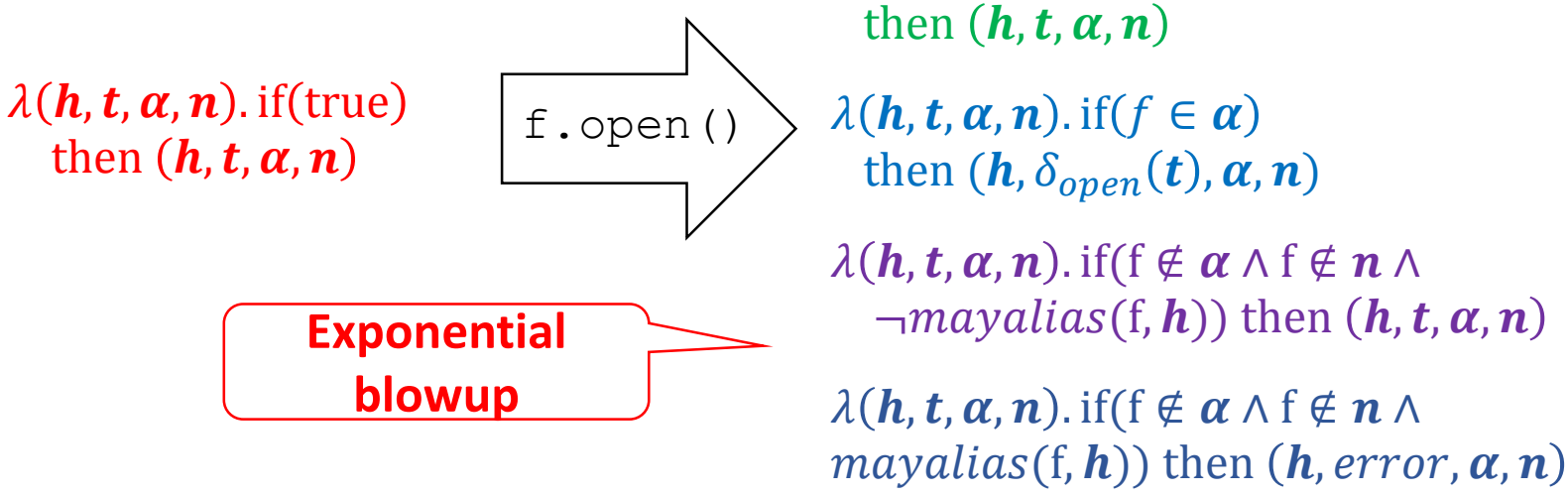
Bottom-up approach

Case condition

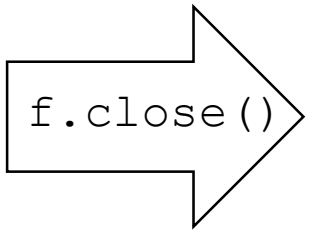
```
foo(File f) {
    λ(h, t, α, n). if(true) then (h, t, α, n)
    f.open();
    f.close();
}
```


Microsoft Research
微软亚洲研究院

Bottom-up approach



Bottom-up approach



$\lambda(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{n})$. if($f \in \boldsymbol{n}$) then $(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{n})$

$\lambda(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{n})$. if($f \in \boldsymbol{\alpha}$) then $(\boldsymbol{h}, \delta_{open}(\boldsymbol{t}), \boldsymbol{\alpha}, \boldsymbol{n})$

$\lambda(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{n})$. if(f $\notin \boldsymbol{\alpha} \wedge$ f $\notin \boldsymbol{n} \wedge$ $\neg mayalias(\text{f}, \boldsymbol{h})$ then $(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{n})$

$\lambda(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{n})$. if(f $\notin \boldsymbol{\alpha} \wedge$ f $\notin \boldsymbol{n} \wedge$ $mayalias(\text{f}, \boldsymbol{h})$) then $(\boldsymbol{h}, error, \boldsymbol{\alpha}, \boldsymbol{n})$

$\lambda(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{n})$. if($f \in \boldsymbol{n}$) then $(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{n})$

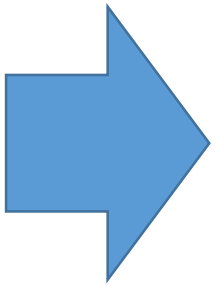
$\lambda(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{n})$. if($f \in \boldsymbol{\alpha}$) then $(\boldsymbol{h}, \delta_{open \circ close}(\boldsymbol{t}), \boldsymbol{\alpha}, \boldsymbol{n})$

$\lambda(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{n})$. if(f $\notin \boldsymbol{\alpha} \wedge$ f $\notin \boldsymbol{n} \wedge$ $\neg mayalias(\text{f}, \boldsymbol{h})$) then $(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{n})$

$\lambda(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{n})$. if(f $\notin \boldsymbol{\alpha} \wedge$ f $\notin \boldsymbol{n} \wedge$ $mayalias(\text{f}, \boldsymbol{h})$) then $(\boldsymbol{h}, error, \boldsymbol{\alpha}, \boldsymbol{n})$

Bottom-up approach

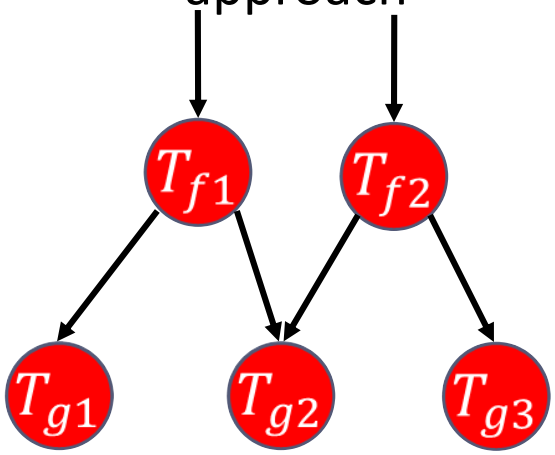
```
foo(File f) {
    f.open();
    f.close();
}
```



Bottom-up summaries
$\lambda(\boldsymbol{h},\boldsymbol{t},\boldsymbol{\alpha},\boldsymbol{n})$. if$(f \in \boldsymbol{n})$ then $(\boldsymbol{h},\boldsymbol{t},\boldsymbol{\alpha},\boldsymbol{n})$ $[\boldsymbol{B_1}]$
$\lambda(\boldsymbol{h},\boldsymbol{t},\boldsymbol{\alpha},\boldsymbol{n})$. if$(f \in \boldsymbol{\alpha})$ then $(\boldsymbol{h}, \delta_{open \circ close}(\boldsymbol{t}),\boldsymbol{\alpha},\boldsymbol{n})$ $[\boldsymbol{B_2}]$
$\lambda(\boldsymbol{h},\boldsymbol{t},\boldsymbol{\alpha},\boldsymbol{n})$. if(f $\notin \boldsymbol{\alpha} \wedge$ f $\notin \boldsymbol{n} \wedge \neg mayalias(\text{f},\boldsymbol{h})$) then $(\boldsymbol{h},\boldsymbol{t},\boldsymbol{\alpha},\boldsymbol{n})$ $[\boldsymbol{B_3}]$
$\lambda(\boldsymbol{h},\boldsymbol{t},\boldsymbol{\alpha},\boldsymbol{n})$. if(f $\notin \boldsymbol{\alpha} \wedge$ f $\notin \boldsymbol{n} \wedge mayalias(\text{f},\boldsymbol{h})$) then $(\boldsymbol{h}, error,\boldsymbol{\alpha},\boldsymbol{n})$ $[\boldsymbol{B_4}]$

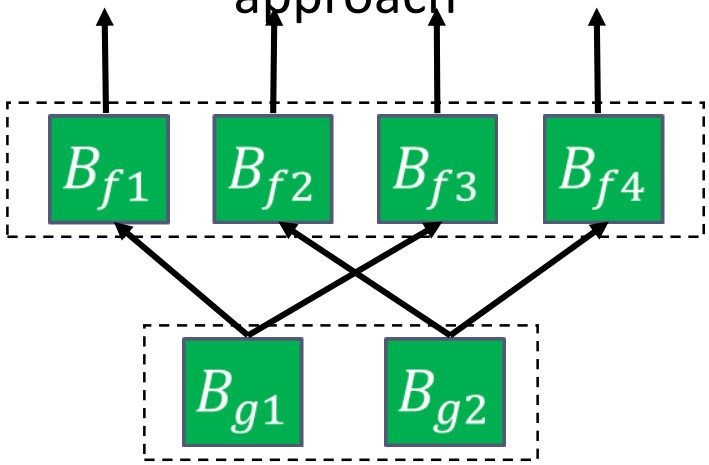
Two approaches to interprocedural analysis

Top-down approach



- Consider only contexts in program.
- Monomorphic summaries.
- Low reusability.
- Blow-up with number of contexts.
- Cheap to compute.
- Cheap to instantiate.
- Easy to implement.

Bottom-up approach



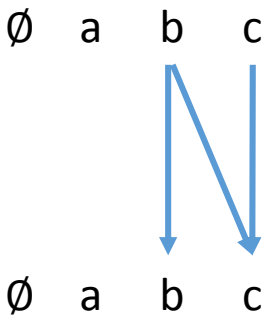
- Consider all possible contexts.
- Polymorphic summaries.
- High reusability.
- Blow-up with number of cases.
- Expensive to compute.
- Expensive to instantiate.
- Hard to implement.

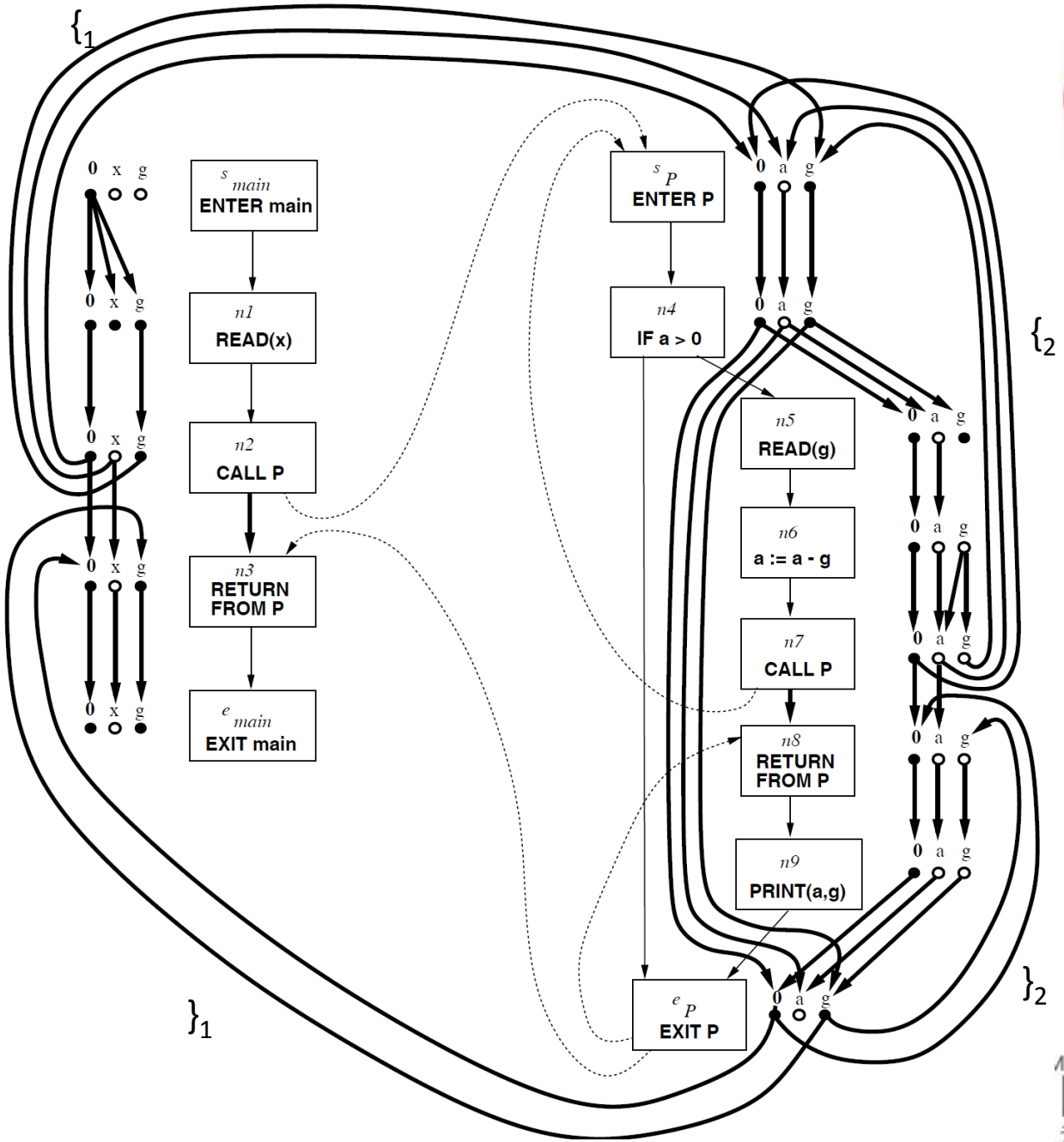
上下文无关语言可达性 -问题定义

- 以上自顶向下的分析常常被描述为上下文无关语言可达性问题（Context-free language Reachability）
- 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。

上下文无关文法可达性-抽象方法

- 对于摘要Ø->{}, a->{}, b->{b, c}, c->{c}
- 可以表示成图




{1
0 x g
0 x g
0 x g
0 x g
0 x g
s main
ENTER main
n1
READ(x)
n2
CALL P
n3
RETURN
FROM P
e main
EXIT main
}1
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 a g
{2
}2

限定文法

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

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

$\quad | \ \ldots$

$\quad | \ aS$

$\quad | \ \epsilon$

Microsoft Research
微软亚洲研究院

计算方法

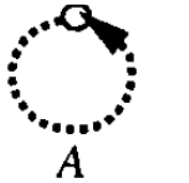
- 把原文法改写为右边只有最多两个非终结符的形式

$S \rightarrow L_1E_1 \mid L_2E_2 \mid \ldots \mid AS \mid \epsilon$

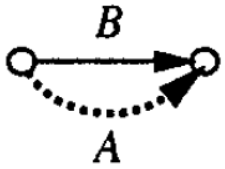
$E_1 \rightarrow SR_1$
$E_2 \rightarrow SR_2$
$L_1 \rightarrow \{_1$
$L_2 \rightarrow \{_2$
$R_1 \rightarrow \}_1$
$R_2 \rightarrow \}_2$

计算方法

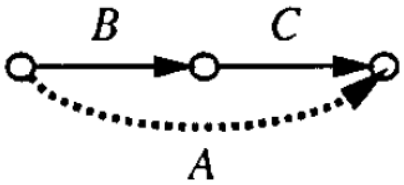
- 按如下三种模式不断添加边，直到没有边需要添加



(a) $A \rightarrow \varepsilon$



(b) $A \rightarrow B$



(c) $A \rightarrow B\ C$

练习

- CFL可达性问题的求解算法和Tabulating算法在效率上有什么不同？

- Tabulating只计算从entry可达的边，而CFL可达性问题的求解算法会计算出任意两点之间的可达性

该领域的最新论文

- **On Abstraction Refinement for Program Analyses in Datalog**.
  - Xin Zhang, Georgia Institute of Technology;
  - Ravi Mangal, Georgia Institute of Technology;
  - Radu Grigore, University of Oxford;
  - Mayur Naik, Georgia Institute of Technology;
  - Hongseok Yang, University of Oxford.
  - PLDI'14
- **Summary-Based Context-Sensitive Data-Dependence Analysis in Presence of Callbacks**
  - Hao Tang (Peking University, China)
  - Xiaoyin Wang (University of Texas at San Antonio)
  - Lingming Zhang (University of Texas at Dallas)
  - Bing Xie (Peking University, China)
  - Lu Zhang (Peking University, China)
  - Hong Mei (Peking University, China)
  - POPL'15

作业

1. 给定由S的子集和交集操作构成的半格，描述如何转成并集操作构成的半格，以便应用自顶向下的过程间分析
2. 将并集操作换成交集操作，给出Gen/Kill标准型在自底向上分析中三种函数合并操作的计算公式