软件分析

基于克隆的过程间分析

熊英飞
北京大学
2014

Microsoft Research
微软亚洲研究院

复习：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？

Microsoft Research 微软亚洲研究院

另一种视角：方程求解

- 数据流分析的传递函数和⊓操作定义了一组方程
  - $\mathrm{D}_{\mathrm{v}_1} = \mathrm{F}_{\mathrm{v}_1}\left(\mathrm{D}_{\mathrm{v}_1}, \mathrm{D}_{\mathrm{v}_2}, \dots, \mathrm{D}_{\mathrm{v}_\mathrm{n}}\right)$
  - $\mathrm{D}_{\mathrm{v}_2} = \mathrm{F}_{\mathrm{v}_2}\left(\mathrm{D}_{\mathrm{v}_1}, \mathrm{D}_{\mathrm{v}_2}, \dots, \mathrm{D}_{\mathrm{v}_\mathrm{n}}\right)$
  - …
  - $\mathrm{D}_{\mathrm{v}_\mathrm{n}} = \mathrm{F}_{\mathrm{v}_\mathrm{n}}\left(\mathrm{D}_{\mathrm{v}_1}, \mathrm{D}_{\mathrm{v}_2}, \dots, \mathrm{D}_{\mathrm{v}_\mathrm{n}}\right)$
- 其中
  - $F_{v_i}\left(\mathrm{D}_{\mathrm{v}_1}, \mathrm{D}_{\mathrm{v}_2}, \dots, \mathrm{D}_{\mathrm{v}_\mathrm{n}}\right) = f_{v_i}(I)$
  - $F_{v_i}\left(\mathrm{D}_{\mathrm{v}_1}, \mathrm{D}_{\mathrm{v}_2}, \dots, \mathrm{D}_{\mathrm{v}_\mathrm{n}}\right) = f_{v_i}\left(\sqcap_{j \in pred(i)} D_{v_j}\right)$
- 数据流分析即为求解该方程的最大解
  - 传递函数和⊓操作表达了该分析的安全性条件，所以该方程的解都是安全的
  - 最大解是最有用的解

Microsoft Research 微软亚洲研究院

方程组求解算法

- 在数理逻辑学中，该类算法称为Unification算法
  - 参考：http://en.wikipedia.org/wiki/Unification_(computer_science)
- 对于单调函数和有限格，标准的Unification算法就是我们学到的最大不动点算法
  - 从$(I, \top, \top, \ldots, \top)$开始反复应用$F_{v_1}$到$F_{v_n}$，直到达到不动点

Microsoft Research
微软亚洲研究院

方程组求解视角的意义

- 如果我们能把分析的安全性表达为数据的约束，我们就把原问题转换成寻找合适的unification算法求解方程组的问题
- 根据分析需要，我们可以用合适的unification算法获得最大解或者最小解
- 有一个有用的解不等式的unification算法
  - 不等式
    - $D_{v_1} \sqsubseteq F_{v_1}(D_{v_1}, D_{v_2}, \ldots, D_{v_n})$
    - $D_{v_2} \sqsubseteq F_{v_2}(D_{v_1}, D_{v_2}, \ldots, D_{v_n})$
    - …
    - $D_{v_n} \sqsubseteq F_{v_n}(D_{v_1}, D_{v_2}, \ldots, D_{v_n})$
  - 可以通过转换成如下方程组求解
    - $D_{v_1} = D_{v_1} \sqcap F_{v_1}(D_{v_1}, D_{v_2}, \ldots, D_{v_n})$
    - $D_{v_2} = D_{v_2} \sqcap F_{v_2}(D_{v_1}, D_{v_2}, \ldots, D_{v_n})$
    - …
    - $D_{v_n} = D_{v_n} \sqcap F_{v_n}(D_{v_1}, D_{v_2}, \ldots, D_{v_n})$
- 很多问题可以转成上述形式的不等式，比如指针分析问题

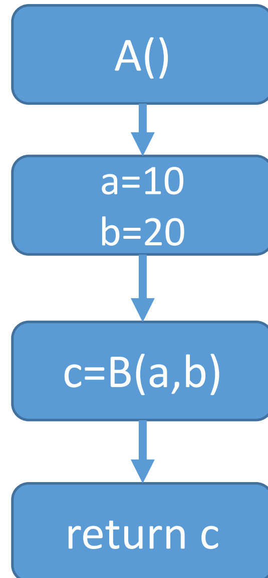
Microsoft Research 微软亚洲研究院

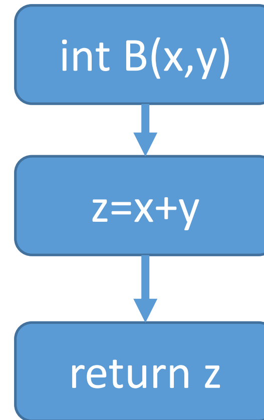
过程间分析

- 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

Microsoft Research 微软亚洲研究院

过程间分析-示例

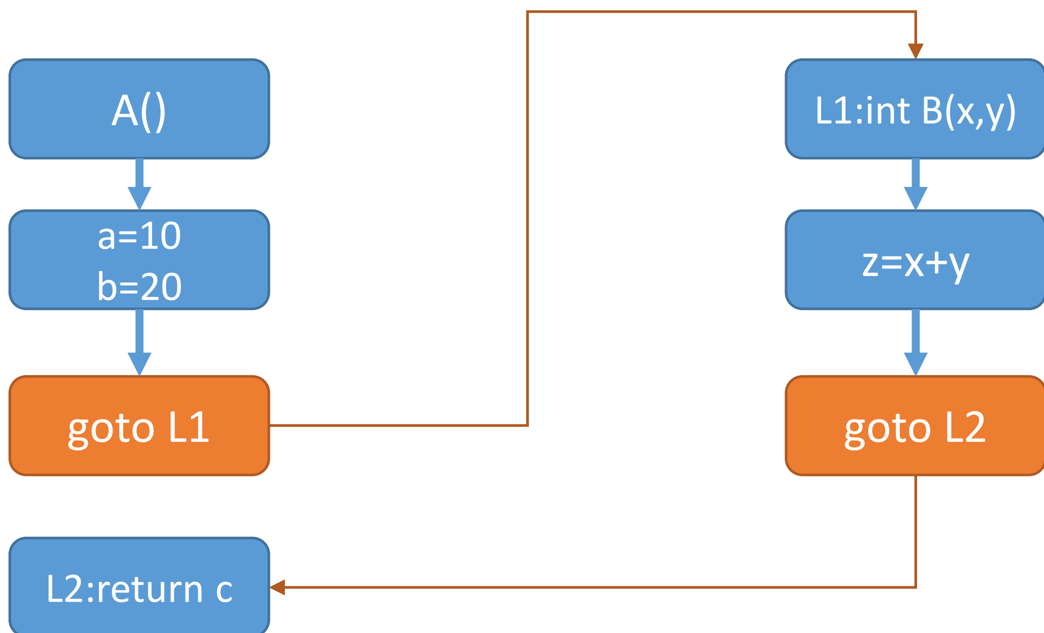




Microsoft Research 微软亚洲研究院

过程间分析-基本思路

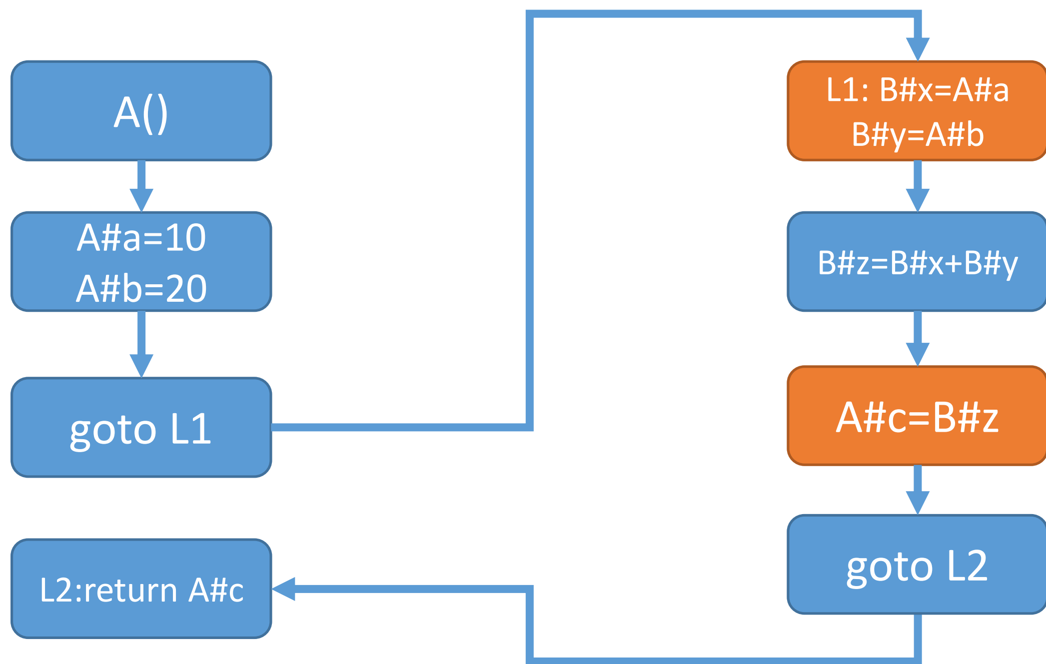
- 把Call/Return语句当成goto语句



Microsoft Research 微软亚洲研究院

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

Microsoft Research 微软亚洲研究院

一种Super CFG的优化方法

- 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

A()

A#a=10
A#b=20

goto L1

L2:return A#c

L1: B#x=A#a
B#y=A#b

B#z=B#x+B#y

A#c=B#z

goto L2

A#a=A#a
A#b=A#b
B#x=B#x
B#y=B#y
B#z=B#x$\hat{+}$B#y
A#c=A#c

符号分析时的转换函数

Microsoft Research 微软亚洲研究院

一种Super CFG的优化方法

- 在本地调用与返回之间添加边来传递本地信息
- 如果有全局变量，添加结点将全局变量重置为T

a=T
b=T
c=T

A()

a=正
b=正
c=c

a=10
b=20

a=a
b=b
c=c

Call B(a,b)

return c

B(x,y)

x=a
y=b
z=T

z=x+y

x=x
y=y
z=x$\hat{+}y$

Return Z

x=x
y=y
z=z

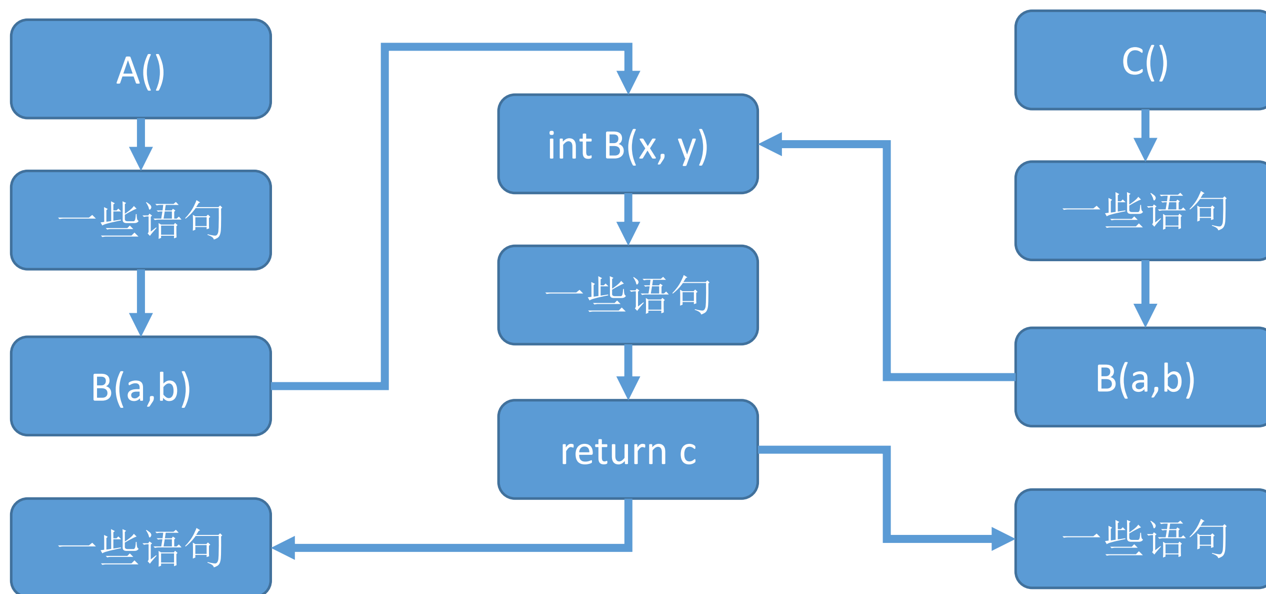
C=B_return

a=T
b=T
c=z

Microsoft Research
微软亚洲研究院

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

Microsoft Research 微软亚洲研究院

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

Microsoft Research 微软亚洲研究院

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

Microsoft Research
微软亚洲研究院

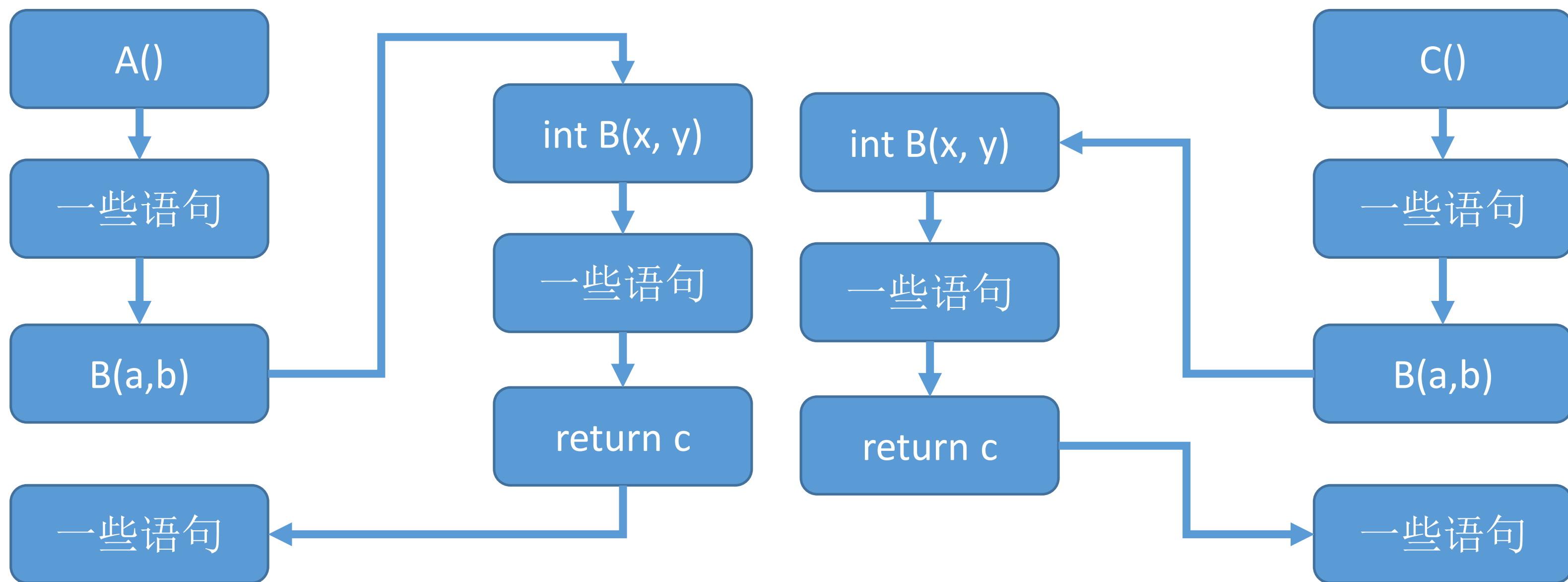
上下文敏感性
Context-sensitivity

- 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

Microsoft
Research
微软亚洲研究院

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

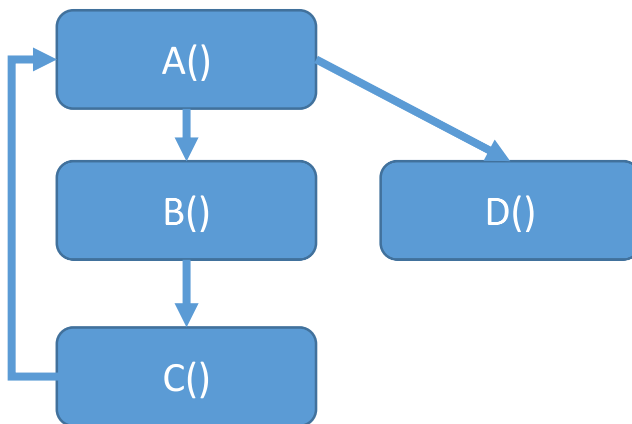
- 给每一处调用创建原函数的一份复制



Microsoft Research
微软亚洲研究院

Call Graph调用图

- 结点表示过程，边表示调用关系



Microsoft Research 微软亚洲研究院

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;}

Microsoft Research 微软亚洲研究院

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);}

Microsoft Research 微软亚洲研究院

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

Microsoft Research 微软亚洲研究院

基于克隆的上下文敏感分析

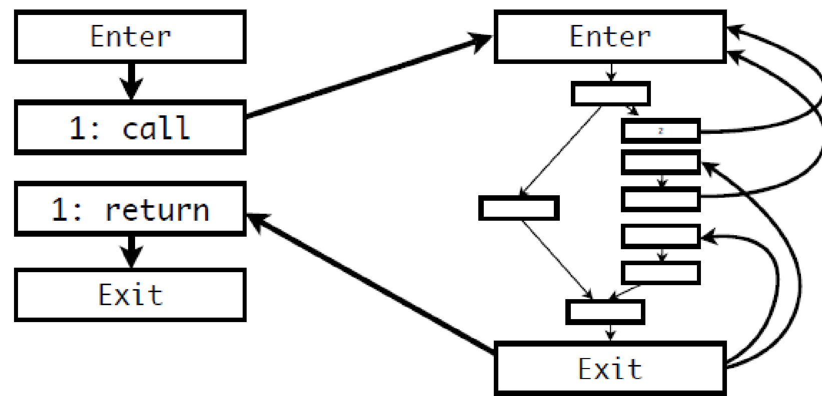
- 解决方案1：只复制没有递归调用的函数
  - 仍然没有解决指数爆炸的问题
  - 如果有一个比较长的递归调用链，则该链上的函数都不能较精确的分析
    - p() { if(…) q(); else r();}
    - q() {if (…) r(); else s();}
    - r() {if (…) p(); else s();}
    - s() {…;p();…}
- 解决方案2：
  - 只使用最近k次调用区分上下文
  - 如果最近k次调用的位置都相同，则不复制，否则复制

Microsoft Research 微软亚洲研究院

Fibonacci函数示例 --Context-Insensitive

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

fib(int n) {
  if n <= 1
       x := 0
  else
     2: y := fib(n-1);
     3: z := fib(n-2);
        x:= y+z;
  return x;
}
```



Microsoft Research 微软亚洲研究院

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

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

fib(int n) {
  if n <= 1
       x := 0
  else
    2: y := fib(n-1);
    3: z := fib(n-2);
       x:= y+z;
  return x;
}
```

Context: -

Enter

1: call

1: return

Exit

Context: 1

Enter

Exit

Context: 2

Enter

Exit

Context: 3

Enter

Exit

Microsoft Research 微软亚洲研究院

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

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

fib(int n) {
  if n <= 1
       x := 0
  else
    2: y := fib(n-1);
    3: z := fib(n-2);
       x:= y+z;
  return x;
}
```

Context: -

Enter

1: call

1: return

Exit

Context: 1::-

Context: 2::1

Context: 3::1

Context: 2::2

Context: 3::2

Context: 2::3

Context: 3::3

Microsoft Research 微软亚洲研究院

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

Microsoft Research 微软亚洲研究院

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

Microsoft Research 微软亚洲研究院

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？
    - 把代码变换成if(c1){x();z();n();} else {y();z();m();}
- 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

Microsoft Research 微软亚洲研究院

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

Microsoft Research 微软亚洲研究院

动态过程调用

- 实际程序中被调用过程往往是动态确定的
  - 命令式语言的过程指针
  - 面向对象语言的虚函数/抽象方法
  - 函数式语言中的高阶函数
- 无法静态确定具体的调用过程
- 解决方案
  - 考虑所有可能的过程
  - 根据上下文复制所有可能的过程并添加跳转路径
  - 根据指针分析的结果往往可以把可能的过程限制在一个比较小的范围

Microsoft Research
微软亚洲研究院

指针分析

- 程序静态分析中研究最广的问题
- 几乎涉及到程序静态分析的所有方面
- 因为内容太深太广，建议直接使用工具，有必要时再深入学习

Microsoft Research 微软亚洲研究院

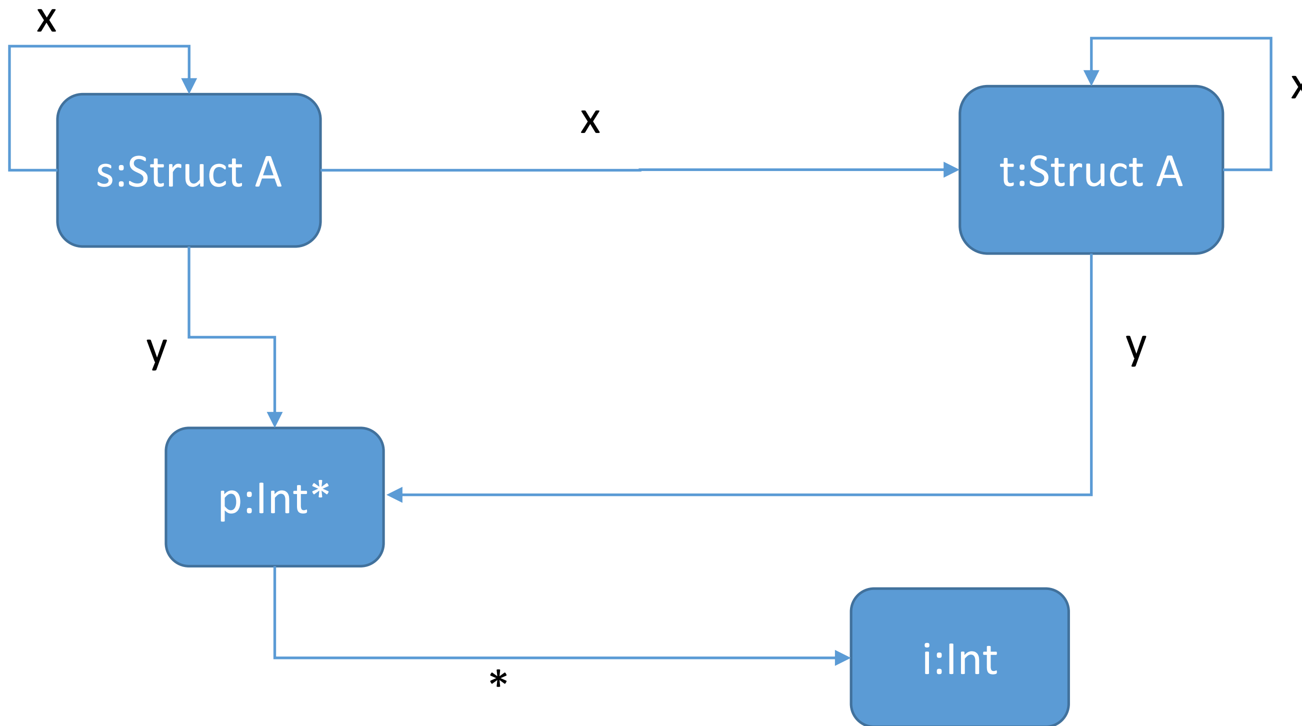
指针分析-术语

- 指针分析Pointer Analysis：通用指以下所有类别的和指针有关的分析
- 指向分析Points-to Analysis：确定每个指针变量所指向的可能内存位置的集合
- 别名分析Alias Analysis：给定两个表达式，判断他们是must alias, may alias, 还是must-not alias
- 形状分析Shape Analysis：分析内存中对象互相指向所形成的结构

- 传统上，以上各种分析的算法独立发展。在现代指针分析中，以上分析都合并成同一种基于指向图的分析。

Microsoft Research 微软亚洲研究院

指向图Points-to Graph



指向图中的一个结点通常对应：1）一个过程局部变量；2)一个全局变量；3)一个malloc/new语句，抽象表示所有该语句分配的内存

从指向图中可以得到以上三种分析所需的信息。

Microsoft Research
微软亚洲研究院

指针分析-其他常见术语

- flow-sensitive: 在每一个CFG node上产生一个指向图
- context-sensitive:
  - heap-cloing：每个过程产生一个抽象的指向图，即指向图中的结点不与具体的程序信息关联。该指向图和传入参数信息结合的时候产生具体的指向图
- Field-sensitivity: 指向图的边上是否带标签
- inclusion-based/Andersen-style:
  - 两种基本的flow-insensitive指针分析算法之一
  - 产生较精确的分析结果，但速度较慢
- unification-based/Steensgaard-style:
  - 当指向图中同一个结点的两条出边标签相同时，合并这两条边和对应的结点
  - 精度不如Andersen算法，但速度较快

Microsoft Research 微软亚洲研究院

程序分析框架

- 作用
  - 词法分析、语法分析
  - 产生控制流图
  - 提供指针分析、产生调用图
  - 将程序转成易于分析的中间语言
    - 三地址码
    - SSA形式（每个变量只被赋值一遍）
  - 提供转换函数和合并操作，自动产生数据流分析
  - 提供摘要构建函数，自动产生过程间分析

- Java
  - Soot
  - WALA
  - Chord
- C
  - LLVM
  - ~~Saturn~~
  - ~~Crystal~~

Microsoft Research 微软亚洲研究院

Java 分析框架Soot

- 加拿大McGill大学Laurie J. Hendren教授的团队开发
- 当前维护者
  - Patrick Lam（加拿大Waterloo大学教授，也是2.0的主要开发者）
  - Feng Qian（谷歌工程师，也是2.0的主要开发者）
  - Ondrej Lhotak （加拿大Waterloo大学教授，也是2.0的主要开发者）
  - Eric Bodden（德国TU Darmstadt教授）
- 在Java的分析框架中，相对用户较多文档较全

Microsoft Research 微软亚洲研究院

Soot

- Java分析框架
- 包括命令行和Eclipse插件两种工具接口
- 包含一系列中间语言
- 提供构建分析、转换工具的一系列接口

Microsoft Research 微软亚洲研究院

Soot中间语言

- Baf: Java字节码的Soot版本
- Jimple： 三地址中间码
- Shimple: SSA版本的Jimple
- Grimp： 合并表达式后的Jimple

Microsoft Research 微软亚洲研究院

Soot Jimple语句示例

- Core statements:
  - NopStmt
  - DefinitionStmt: IdentityStmt, AssignStmt
- Intraprocedural control-flow:
  - IfStmt
  - GotoStmt
  - TableSwitchStmt,LookupSwitchStmt
- Interprocedural control-flow:
  - InvokeStmt
  - ReturnStmt, ReturnVoidStmt
  - …
- Jimple总共只包括15条语句，但Java字节码包括超过200条语句

Microsoft Research 微软亚洲研究院

Soot程序结构获取方法

Scene.v()

Scene (singleton)

getSootClass()

SootClass getField() SootField getSignature()

getMethod()

SootMethod getSignature()

getActiveBody()

JimpleBody

同时提供一系列方法获取Jimple语句，也就对应了控制流图。

Microsoft Research
微软亚洲研究院

在Soot中实现过程内数据流分析

1. Subclass *FlowAnalysis
2. Implement abstraction: merge(), copy()
3. Implement flow function flowThrough()
4. Implement initial values:
   - newInitialFlow() and
   - entryInitialFlow()
5. Implement constructor
   - (it must call doAnalysis())

Microsoft Research 微软亚洲研究院

Soot对过程间分析的支持

- Soot不提供过程间分析框架
- 提供Call Graph的构建和查询
- 提供points-to分析

Microsoft Research
微软亚洲研究院

LLVM

- 请高庆介绍

Microsoft
Research
微软亚洲研究院