软件分析与测试
数据流分析

严俊

中国科学院大学
University of Chinese Academy of Sciences
ISCAS 中国科學院軟件研究所
Institute of Software, Chinese Academy of Sciences

本次课内容

- 经典程序分析算法
  - 数据流分析：到达定值、活跃变量、非常忙表达式、可用表达式
  - 污点分析
  - 指向分析

数据流分析

- 数据流分析的应用场景
- 基本概念
- 到达定值、活跃变量、非常忙表达式、可用表达式分析

APP中的隐私泄露1

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_field_sensitivity4);

    TelephonyManager telephonyManager = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
    String imei = telephonyManager.getDeviceId(); //source

    Datacontainer data1 = new Datacontainer();

    SmsManager sms = SmsManager.getDefault();
    sms.sendTextMessage("+49 1234", null, data1.value, null, null); //sink, no leak

    data1.value = imei;
}

class Datacontainer{
    String value = "android";
}
```

value是什么?

敏感函数

APP中的隐私泄露2

```java
public class InheritedObjects1 extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_inherited_objects1);

        int a = 45 + 1;
        TelephonyManager telephonyManager = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
        General g;
        if(a == 46){
            g = new VarA();
            g.man = telephonyManager;
        } else{
            g = new VarB();
            g.man = telephonyManager;
        }
        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage("+49 1234", null, g.getInfo(), null, null);  //sink, leak
    }
}
```

```java
public class VarA extends General{
    @Override
    public String getInfo() {
        return man.getDeviceId(); //source
    }
}

public class VarB extends General{
    @Override
    public String getInfo() {
        return "abc";
    }
}
```

g是什么?

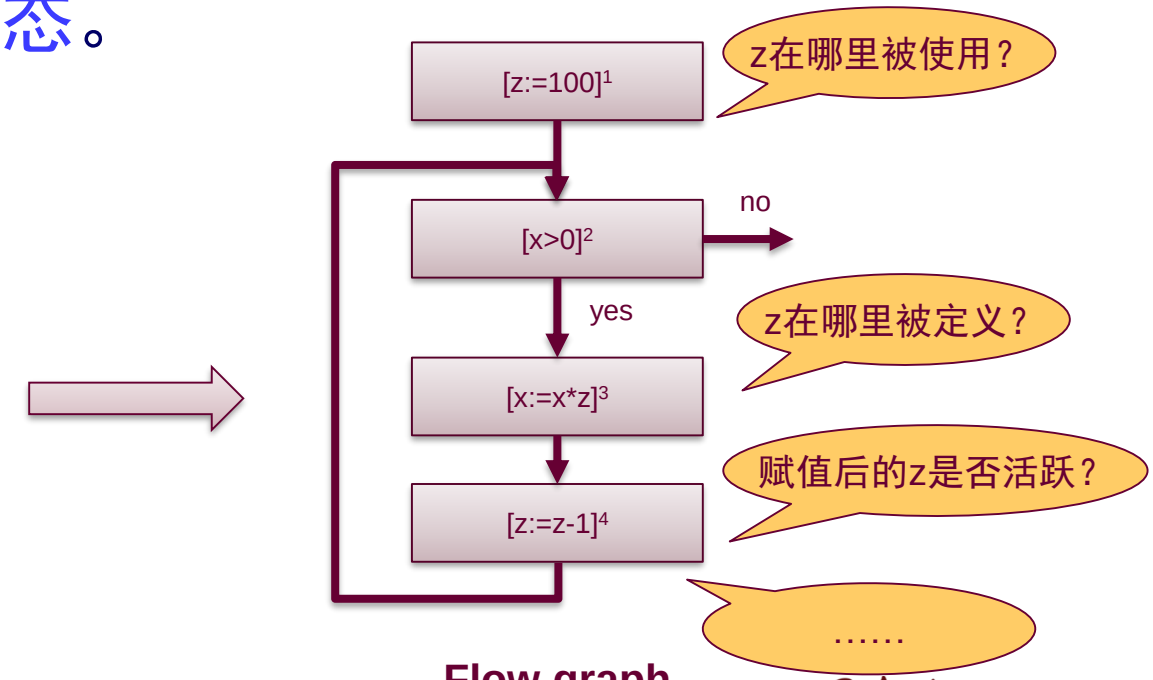
敏感函数

数据流分析

- 数据流分析通过分析程序状态信息在控制流图中的传播来计算每个静态程序点（语句）在运行时可能出现的状态。

```
[z:=100]1;
while [x>0]2 do (
      [x:=x*z]3;
      [z:=z-1]4
)
```

Code



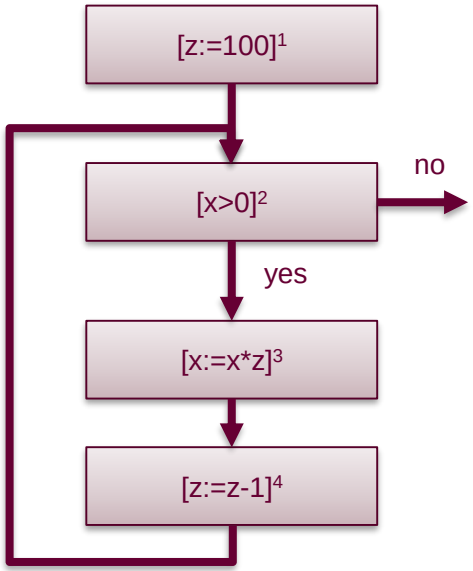
Flow graph

数据流分析的应用场景

- 程序分析
  - 检测特定错误模式
  - 辅助测试过程
- 编译优化
  - 去除冗余的代码
  - 合并同样的计算
- …

基本概念

- 标签 *Labels: Stmt → P(Lab)*
  - *labels([x=a]^l) = {l}*
- 初始标签 Init label: *Stmt → Lab*
  - *init([x=a]^l) = l*
  - *init(S_1 ; S_2) = init(S_1)*
  - *init(if [b]^l then S_1 else S_2) = l*
- 结束标签 Final label: *Stmt → P(Lab)*
  - *final([x=a]^l) = {l}*
  - *final(S_1 ; S_2) = final(S_2)*
  - *final(if [b]^l then S_1 else S_2) = final(S_1)∪final(S_2)*



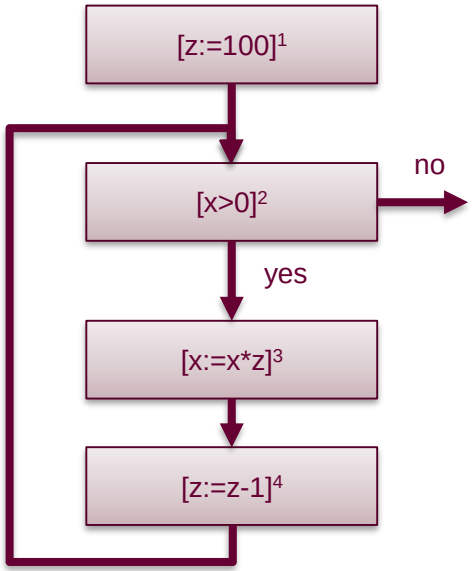
Flow graph

[1] Flemming Nielson , et al. **Principles of Program Analysis.** Springer 1999.

基本概念

- 基本块 *blocks: Stmt → P(Blocks)*
  - $blocks([x=a]^l) = \{[x=a]^l\}$
  - $blocks(S_1; S_2) = blocks(S_1) \cup blocks(S_2)$

- 数据流 *flow*: Stmt → P(Lab × Lab)
  - $flow([x=a]^l)=\emptyset$
  - $flow(S_1; S_2)=flow(S_1) \cup flow(S_2) \cup \{(l,init(S_2)) \mid l \in final(S_1)\}$



Flow graph

[1] Flemming Nielson , et al. **Principles of Program Analysis.** Springer 1999.

基本概念 Example

- **Code:**

  $[z=100]^1$;while $[x>0]^2$ do ($[x=x*z]^3$; $[z=z-1]^4$)

- ***init(code)=1***
- ***final(code)=2***
- ***labels(code)={1,2,3,4}***
- ***flow(code)={(1,2),(2,3),(3,4),(4,2)}***

$[z:=100]^1$;
while $[x>0]^2$ do (
    $[x:=x*z]^3$;
    $[z:=z-1]^4$
)

Code

经典的过程内数据流分析

算法分类：

前向分析 vs. 后向分析，May分析 vs. Must分析

前向Must：可用表达式分析	→	公共子表达式消除
前向May：到达定值分析	→	数据传播分析
后向Must：非常忙表达式分析	→	公共子表达式提取
后向May：活跃变量分析	→	死代码消除

可用表达式 Available Expressions

- 目标：对于每个程序点找到一组表达式，在任意到该程序点的路径上均被计算过且结果未被修改过。
- 用于：在某一点重用之前的表达式，无需重复计算。
- Example:

[x=a+b]^1;
[y=a*b]^2;
while [z>a*b]^3 do
    ([a=a+1]^4;
    [x=a+b]^5)

l =3处，a*b是可用表达式吗？

可用表达式 Available Expressions

- 目标：对于每个程序点找到一组表达式，在任意到该程序点的路径上均被计算过且结果未被修改过。
- 用于：在某一点重用之前的表达式，无需重复计算。
- Example:

[x=a+b]^1;
[y=a*b]^2;
while [z>a+b]^3 do
    ([a=a+1]^4;
    [x=a+b]^5)

l =3处，a+b是可用表达式吗？

可用表达式 Available Expressions

- 目标：对于每个程序点找到一组表达式，在任意到该程序点的路径上均被计算过且结果未被修改过。
- 用于：在某一点重用之前的表达式，无需重复计算。
- Example:

a+b被生成

$[x=a+b]^1$;

a+b被杀死

$[a=a*b]^2$;

l =3处，a+b是可用表达式吗？

while $[z>a+b]^3$ do

a+b被杀死

($[a=a+1]^4$;

$[x=a+b]^5$)

a+b被生成

可用表达式 Available Expressions

- 目标：对于每个程序点找到一组表达式，在任意到该程序点的路径上均被计算过且结果未被修改过。
- 用于：在某一点重用之前的表达式，无需重复计算。
- Example:

[x=a+b]^1;
[a=a*b]^2;
while [z>a+b]^3 do
    ([a=a+1]^4;
    [x=a+b]^5)

l =3处，a+b并非在任意到该程序点的路径上均被计算过且结果未被修改过。

数据流经语句 l，kill/gen表达式

- $kill_{AE}(l)$ produces the set of expressions that any of the variables used in the expression are modified in the block.
  - $kill_{AE}([x=a+b]^l)=\emptyset$; $kill_{AE}([a=a+b]^l)=\{a+b\}$;
- $gen_{AE}(l)$ produces set of expressions that are evaluated in the block and none of the variables used in the expression are later modified in the block.
  - $gen_{AE}([x=a+b]^l)=\{a+b\}$ $gen_{AE}([a=a+b]^l)=\emptyset$

数据流经语句 l，可用表达式集合的变化

- $AE_{entry}(l)$：所有流向l的语句l'的exit的交集
  - $AE_{entry}(l)$ =

$$\begin{cases} \emptyset & if\ l = init(S) \\ \cap \{AE_{exit}(l') | (l', l) \in flow(S)\} & otherwise \end{cases}$$

- $AE_{exit}(l)$：语句l的entry集合进行kill和gen操作
  - $AE_{exit}(l) = (AE_{entry}(l) \backslash kill_{AE}(B^l)) \cup gen_{AE}(B^l)$，
    $B^l \in blocks(S)$

例子

- **Example: $[x=a+b]^1;[y=a*b]^2;$while $[y>a+b]^3$ do $([a=a+1]^4; [x=a+b]^5)$**

l	$kill_{AE}(l)$	$gen_{AE}(l)$
1	∅	{a+b}
2	∅	{a*b}
3	∅	{a+b}
4	{a+b, a*b, a+1}	∅
5	∅	{a+b}

l	$AE_{entry}(l)$	$AE_{exit}(l)$
1	∅	$AE_{entry}(1)\{∅\} \cup \{a+b\}$
2	$AE_{exit}(1)$	$AE_{entry}(2) \{∅\} \cup \{a*b\}$
3	$AE_{exit}(2) \cap AE_{exit}(5)$	$AE_{entry}(3) \{∅\} \cup \{a+b\}$
4	$AE_{exit}(3)$	$AE_{entry}(4)\{a+b,a*b,a+1\} \cup \{∅\}$
5	$AE_{exit}(4)$	$AE_{entry}(5)\{∅\} \cup \{a+b\}$

l	$AE_{entry}(l)$	$AE_{exit}(l)$
1	∅	{a+b}
2	{a+b}	{a+b,a*b}
3	{a+b,a∗b} ∩ $AE_{exit}(5)$	{a+b,a∗b}
4	{a+b,a∗b}	∅
5	∅	{a+b}

初始值均为全集

例子

- **Example: [x=a+b]^1;[y=a*b]^2;while [y>a+b]^3 do ([a=a+1]^4; [x=a+b]^5)**

l	$kill_{AE}(l)$	$gen_{AE}(l)$
1	∅	{a+b}
2	∅	{a*b}
3	∅	{a+b}
4	{a+b, a*b, a+1}	∅
5	∅	{a+b}

l	$AE_{entry}(l)$	$AE_{exit}(l)$
1	∅	$AE_{entry}(1)\backslash\{\emptyset\} \cup \{a+b\}$
2	$AE_{exit}(1)$	$AE_{entry}(2)\backslash\{\emptyset\} \cup \{a*b\}$
3	$AE_{exit}(2) \cap AE_{exit}(5)$	$AE_{entry}(3)\backslash\{\emptyset\} \cup \{a+b\}$
4	$AE_{exit}(3)$	$AE_{entry}(4)\backslash\{a+b, a*b, a+1\} \cup \{\emptyset\}$
5	$AE_{exit}(4)$	$AE_{entry}(5)\backslash\{\emptyset\} \cup \{a+b\}$

l	$AE_{entry}(l)$	$AE_{exit}(l)$
1	∅	{a+b}
2	{a+b}	{a+b,a*b}
3	{a+b}a∗b} ∩ $AE_{exit}(5)$	{a+b}a∗b}
4	{a+b}a∗b}	∅
5	∅	{a+b}

迭代更新

迭代到不动点

例子

- **Example: [x=a+b]^1;[y=a*b]^2;while [y>a+b]^3 do ([a=a+1]^4; [x=a+b]^5)**

l	$kill_{AE}(l)$	$gen_{AE}(l)$
1	∅	{a+b}
2	∅	{a*b}
3	∅	{a+b}
4	{a+b, a*b, a+1}	
5	∅	

l	$AE_{entry}(l)$	$AE_{exit}(l)$
1	∅	$AE_{entry}(1)\backslash\{\emptyset\} \cup \{a+b\}$
2	$AE_{exit}(1)$	$AE_{entry}(2) \backslash\{\emptyset\} \cup \{a*b\}$
3	$AE_{exit}(2) \cap AE_{exit}(5)$	$AE_{entry}(3) \backslash\{\emptyset\} \cup \{a+b\}$
4		$AE_{entry}(4)\backslash\{a+b,a*b,a+1\} \cup \{\emptyset\}$
5		$AE_{entry}(5)\backslash\{\emptyset\} \cup \{a+b\}$

特征1：所有路径满足条件（Must）
特征2：表达式传播流向前，如1→2
具有这类特征的：前向Must分析

1	∅	{a+b}
2	{a+b}	{a+b,a*b}
3	{a+b}	{a+b}
4	{a+b}	∅
5	∅	{a+b}

迭代更新

迭代到不动点

到达定值 Reaching Definition

- 目标：对于每个程序点找到一组定义点，在某一条到该程序点的路径上被赋值且赋值未被更改。
- 用于：追踪数据的定义和使用。
- Example:

$[x=5]^1$;
$[y=1]^2$;
while $[x>1]^3$ **do**
  ($[y=x*y]^4$;
  $[x=x-1]^5$)

l=3处，x的值来自哪个定义点？

数据流经语句 l，kill/gen表达式

- $kill_{RD}(l)$ produces the set of pairs of variables and labels of assignments that are destroyed (if the block assigns anew value to the variable) by the block.
  - $([x=3]^1;[x=a+b]^2;)$, $kill_{RD}([x=a+b]^2)=\{(x,?), (x,1),(x,2)\}$
  - "?" denote uninitialized variables
- $gen_{RD}(l)$ produces the set of pairs of variables and labels of assignments generated by the block; only assignments generate definitions.
  - For $([x=3]^1;[x=a+b]^2;)$, $gen_{RD}([x=a+b]^2)=\{(x,2)\}$

数据流经语句 l，到达定值集合的变化

- **$RD_{entry}(l)$**：所有流向l的语句l'的exit的并集
  - **$RD_{entry}(l)$ =**

$$\begin{cases} \{(x,?)|x \in Var(S)\} & if\ l = init(S) \\ \cup \{RD_{exit}(l')|(l',l) \in flow(S)\} & otherwise \end{cases}$$

- **$RD_{exit}(l)$**：语句l的entry集合进行kill和gen操作
  - **$RD_{exit}(l)$ =** $(RD_{entry}(l) \backslash kill_{RD}(B^l)) \cup gen_{RD}(B^l)$， $B^l \in blocks(S)$

到达定值 Reaching Definition

- **Example: [x=5]^1; [y=1]^2; while [x>1]^3 do ([y=x*y]^4; [x=x-1]^5)**

l	$kill_{RD}(l)$	$gen_{RD}(l)$
1	{(x,?),(x,1),(x,5)}	{(x,1)}
2	{(y,?),(y,2),(y,4)}	{(y,2)}
3	∅	∅
4	{(y,?),(y,2),(y,4)}	{(y,4)}
5	{(x,?),(x,1),(x,5)}	{(x,5)}

l	$RD_{entry}(l)$	$RD_{exit}(l)$
1	{(x,?),(y,?)}	$(RD_{entry}(1)\setminus\{(x,?),(x,1),(x,5)\})\cup\{(x,1)\}$
2	$RD_{exit}(1)$	$(RD_{entry}(2)\setminus\{(y,?),(y,2),(y,4)\})\cup\{(y,2)\}$
3	$RD_{exit}(2)\cup RD_{exit}(5)$	$RD_{entry}(3)$
4	$RD_{exit}(3)$	$(RD_{entry}(4)\setminus\{(y,?),(y,2),(y,4)\})\cup\{(y,4)\}$
5	$RD_{exit}(4)$	$(RDentry(5)\setminus\{(x,?),(x,1),(x,5)\}) \cup \{(x,5)\}$

?为未定义

初始值均为空集

l	$RD_{entry}(l)$	$RD_{exit}(l)$
1	{(x,?),(y,?)}	{(y,?),(x,1)}
2	{(y,?),(x,1)}	{(x,1),(y,2)}
3	{(x,1),(y,2)} ∪ $RD_{exit}(5)$	{(x,1),(y,2)} ∪ $RD_{exit}(5)$
4	{(x,1),(y,2),(y,4),(x,5)}	{(x,1),(y,4),(x,5)}
5	{(x,1),(y,4),(x,5)}	{(y,4),(x,5)}

到达定值 Reaching Definition

- **Example: [x=5]^1; [y=1]^2; while [x>1]^3 do ([y=x*y]^4; [x=x-1]^5)**

l	$kill_{RD}(l)$	$gen_{RD}(l)$
1	{(x,?),(x,1),(x,5)}	{(x,1)}
2	{(y,?),(y,2),(y,4)}	{(y,2)}
3	∅	∅
4	{(y,?),(y,2),(y,4)}	{(y,4)}
5	{(x,?),(x,1),(x,5)}	{(x,5)}

l	$RD_{entry}(l)$	$RD_{exit}(l)$
1	{(x,?),(y,?)}	$(RD_{entry}(1)\setminus\{(x,?),(x,1),(x,5)\})\cup\{(x,1)\}$
2	$RD_{exit}(1)$	$(RD_{entry}(2)\setminus\{(y,?),(y,2),(y,4)\})\cup\{(y,2)\}$
3	$RD_{exit}(2)\cup RD_{exit}(5)$	$RD_{entry}(3)$
4	$RD_{exit}(3)$	$(RD_{entry}(4)\setminus\{(y,?),(y,2),(y,4)\})\cup\{(y,4)\}$
5	$RD_{exit}(4)$	$(RDentry(5)\setminus\{(x,?),(x,1),(x,5)\}) \cup \{(x,5)\}$

?为未定义

迭代更新

l	$RD_{entry}(l)$	$RD_{exit}(l)$
1	{(x,?),(y,?)}	{(y,?),(x,1)}
2	{(y,?),(x,1)}	{(x,1),(y,2)}
3	{(x,1),(y,2)} ∪ $RD_{exit}(5)$	{(x,1),(y,2)} ∪ $RD_{exit}(5)$
4	{(x,1),(y,2),(y,4),(x,5)}	{(x,1),(y,4),(x,5)}
5	{(x,1),(y,4),(x,5)}	{(y,4),(x,5)}

到达定值 Reaching Definition

- **Example: [x=5]^1; [y=1]^2; while [x>1]^3 do ([y=x*y]^4; [x=x-1]^5)**

l	$kill_{RD}(l)$	$gen_{RD}(l)$
1	{(x,?),(x,1),(x,5)}	{(x,1)}
2	{(y,?),(y,2),(y,4)}	{(y,2)}
3	∅	∅
4	{(y,?),(y,2),(y,4)}	{(y,4)}
5	{(x,?),(x,1),(x,5)}	{(x,5)}

l	$RD_{entry}(l)$	$RD_{exit}(l)$
1	{(x,?),(y,?)}	$(RD_{entry}(1)\setminus\{(x,?),(x,1),(x,5)\})\cup\{(x,1)\}$
2	$RD_{exit}(1)$	$(RD_{entry}(2)\setminus\{(y,?),(y,2),(y,4)\})\cup\{(y,2)\}$
3	$RD_{exit}(2)\cup RD_{exit}(5)$	$RD_{entry}(3)$
4	$RD_{exit}(3)$	$(RD_{entry}(4)\setminus\{(y,?),(y,2),(y,4)\})\cup\{(y,4)\}$
5	$RD_{exit}(4)$	$(RDentry(5)\setminus\{(x,?),(x,1),(x,5)\})\cup\{(x,5)\}$

?为未定义

l	$RD_{entry}(l)$	$RD_{exit}(l)$
1	{(x,?),(y,?)}	{(y,?),(x,1)}
2	{(y,?),(x,1)}	{(x,1),(y,2)}
3	{(x,1),(y,2),(y,4),(x,5)}	{(x,1),(y,2),(y,4),(x,5)}
4	{(x,1),(y,2),(y,4),(x,5)}	{(x,1),(y,4),(x,5)}
5	{(x,1),(y,4),(x,5)}	{(y,4),(x,5)}

迭代更新

迭代到不动点

到达定值 Reaching Definition

- **Example: [x=5]^1; [y=1]^2; while [x>1]^3 do ([y=x*y]^4; [x=x-1]^5)**

l	$kill_{RD}(l)$	$gen_{RD}(l)$
1	{(x,?),(x,1),(x,5)}	{(x,1)}
2	{(y,?),(y,2),(y,4)}	{(y,2)}
3	∅	∅
4	{(y,?),(y,2),(y,4)}	{(y,4)}
5	{(x,?),(x,1),(x,5)}	{}

l	$RD_{entry}(l)$	$RD_{exit}(l)$
1	{(x,?),(y,?)}	$(RD_{entry}(1)\setminus\{(x,?),(x,1),(x,5)\})\cup\{(x,1)\}$
2	$RD_{exit}(1)$	$(RD_{entry}(2)\setminus\{(y,?),(y,2),(y,4)\})\cup\{(y,2)\}$
3	$RD_{exit}(2)\cup RD_{exit}(5)$	$RD_{entry}(3)$
4	$RD_{exit}(3)$	$(RD_{entry}(4)\setminus\{(y,?),(y,2),(y,4)\})\cup\{(y,4)\}$
5	$RD_{exit}(4)$	$RD_{entry}(5)\setminus\{(x,?),(x,1),(x,5)\}) \cup \{(x,5)\}$

?为未定义

特征1：存在路径满足条件（May）
特征2：表达式传播流向前，如1→2
具有这类特征的：前向May分析

1	{(x,?),(y,?)}	{(y,?),(x,1)}
2	{(y,?),(x,1)}	{(x,1),(y,2)}
3	{(x,1),(y,2),(y,4),(x,5)}	{(x,1),(y,2),(y,4),(x,5)}
4	{(x,1),(y,2),(y,4),(x,5)}	{(x,1),(y,4),(x,5)}
5	{(x,1),(y,4),(x,5)}	{(y,4),(x,5)}

迭代更新

迭代到不动点

非常忙表达式 Very Busy Expressions

- 目标：对于每个程序点找到一组表达式，无论向后走哪条路径，在表达式中任意一个变量被重新赋值前，这个表达式都会被用到。
- 用于：提前计算这个表达式的值并存储其值，节省空间。
- Example:

If $[a>b]^1$ then

$([x=b-a]^2; [y=a-b]^3)$

else

$([y=b-a]^4; [x=a*b]^5)$

l =1处，哪个表达式可被提前计算？

后向Must分析

活跃变量 Live Variables

- 目标：对于每个程序点找到一组变量，存在一条向后的路径表明该变量在被重新赋值前被使用。
- 用于：消除无用代码。
- Example:

```
[x=2]1; [y=4]2; [x=1]3;
if [y>x]4
  then [z=y]5
  else [x=y*y]6);
print(z)
```

哪些语句是无用代码？

后向May分析

基于迭代的数据流分析

- 基于迭代的数据流分析原理是什么？
- 算法能够保证一定会终止/达到不动点吗？

数据流分析单调框架—偏序

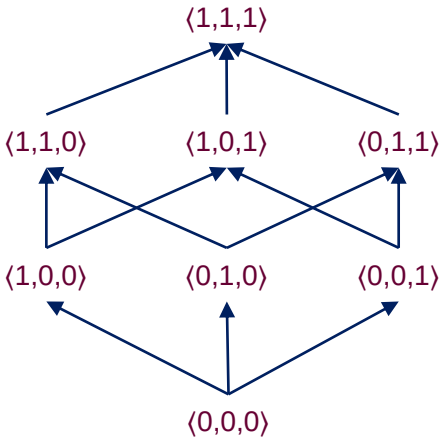
- 对于非空集合S上的关系，如果是自反的、反对称的和传递的，则称其为S上的偏序关系$\sqsubseteq$
  - 自反性：　$\forall x \in S: x \sqsubseteq x$
  - 传递性：　$\forall x,y,z \in S: x \sqsubseteq y \wedge y \sqsubseteq z \Rightarrow x \sqsubseteq z$
  - 非对称性：$\forall x,y \in S: x \sqsubseteq y \wedge y \sqsubseteq x \Rightarrow x = y$
- 集合S和其偏序关系的二元组 (S, $\sqsubseteq$)叫做偏序集
  - 当 P 为整数集合，$\sqsubseteq$ 为小于等于关系时，(P, $\sqsubseteq$) 是一个偏序集。$\sqsubseteq$ 为小于关系时，(P, $\sqsubseteq$) 不是一个偏序集（不满足自反性）。

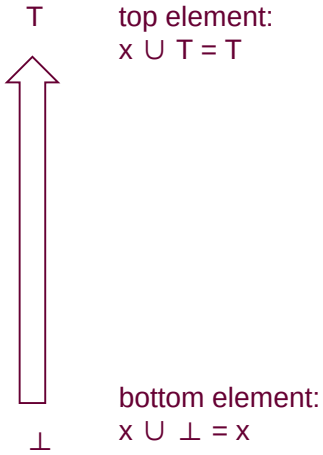
数据流分析单调框架—半格

- 格：
  - 设 (S, ⊑)是偏序集，如果任意两个元素对存在最小上界和最大下界， (S, ⊑)为一个格。
- 半格：
  - 并半格：设 (S, ⊑)是偏序集，如果任意两个元素对存在最小上界。**任意集合和并操作∪组成了一个半格。**
  - 交半格：设 (S, ⊑)是偏序集，如果任意两个元素对存在最大下界。**任意集合和交操作∩组成了一个半格。**

数据流分析单调框架—半格

- **并半格 Semi-lattice**
  - $S = \{x \mid x \subseteq \{ d_1, d_2, d_3\}\}$





数据流分析单调框架—半格

- 半格的高度 H()：
  - **the largest number of ⊑ relations that will fit in a descending chain (最长链的长度+1）**
- H(可用表达式分析) = 表达式总数+1
- H(到达定值变量分析) =变量赋值次数+1
- H(活跃变量分析) =变量总数+1
- H(非常忙表达式分析) =表达式总数+1

数据流分析单调框架—单调函数

- **单调函数 Monotone Function**
  - 给定一个偏序关系 $(S,\sqsubseteq)$，称一个定义在 S上的 函数 f为单调函数，当且仅对任意 $a,b\in S$满足
  - $a\sqsubseteq b \Rightarrow f(a)\sqsubseteq f(b)$
  - 输入的偏序关系决定输出的偏序关系
- RD_{exit}转移方程对应着$f(S)$=(S-KILL)∪GEN操作，其中对于特定的程序语句，GEN, KILL两个集合是固定不变的， $f(S)$ 是一个单调函数
- RD_{entry}转移方程对应着集合上的交/并操作,也是单调的

迭代是否可终止？

- **数据流分析算法是否存在不动点？ (以RD为例)**
  - 定义向量$\overrightarrow{RD}$=($RD_{entry}(1),\ldots, RD_{entry}(n), RD_{exit}(1),\ldots, RD_{exit}(n)$)
  - 定义函数F的$\overrightarrow{RD}$上的数据流操作，$F(\overrightarrow{RD}) = (F_{entry\text{-}1}(\overrightarrow{RD}),\ldots, F_{entry\text{-}m}(\overrightarrow{RD}), \quad F_{exit\text{-}1}(\overrightarrow{RD}),\ldots, F_{exit\text{-}m}(\overrightarrow{RD}))$
  - 因为对entry和对exit的操作函数均为单调函数→ F是单调函数
  - F初始取值为$\vec{\phi}=(\phi,\ldots\phi)$，其中$\phi$对应一个有限集合，F的取值是有限的
    - 根据单调性：$\vec{\phi} \sqsubseteq f(\vec{\phi}) \Rightarrow F(\vec{\phi}) \sqsubseteq F^2(\vec{\phi})$
    - 可得：$\vec{\phi} \sqsubseteq F(\vec{\phi}) \sqsubseteq F^2(\vec{\phi}) \sqsubseteq F^3(\vec{\phi}) \sqsubseteq \ldots \sqsubseteq F^n(\vec{\phi}) = F^{n+1}(\vec{\phi})$
  - 因此，存在不动点 $F^n(\emptyset)$

迭代是否可终止？

- **数据流分析算法是否存在不动点？**
  - May分析，自底⊥向上，$F(\perp), F(F(\perp)), \ldots, F_n(\perp)$直到最大不动点
  - Must分析，自顶 T 向下，$F(T), F(F(T)), \ldots, F_n(T)$ 直到最小不动点
  - 对于**数据流分析算法**，均存在不动点

重新走一遍到达定值分析

Example:

[x=5]^1;

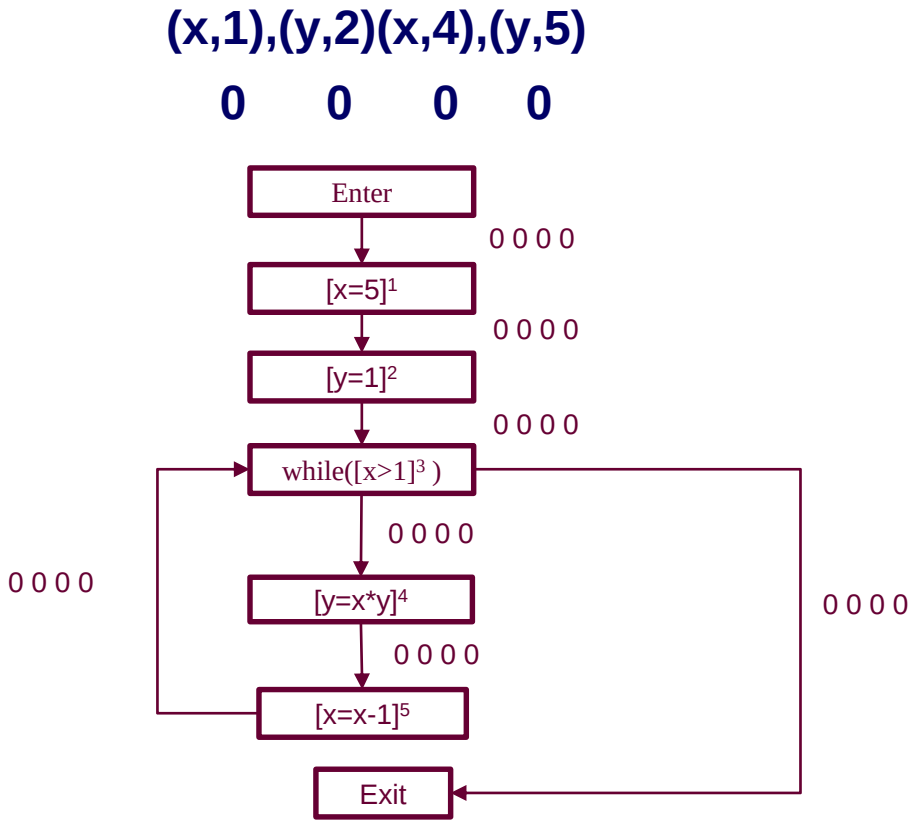
[y=1]^2;

while [x>1]^3 do (

[y=x*y]^4;

[x=x-1]^5

)



重新走一遍到达定值分析

Example:

[x=5]^1;

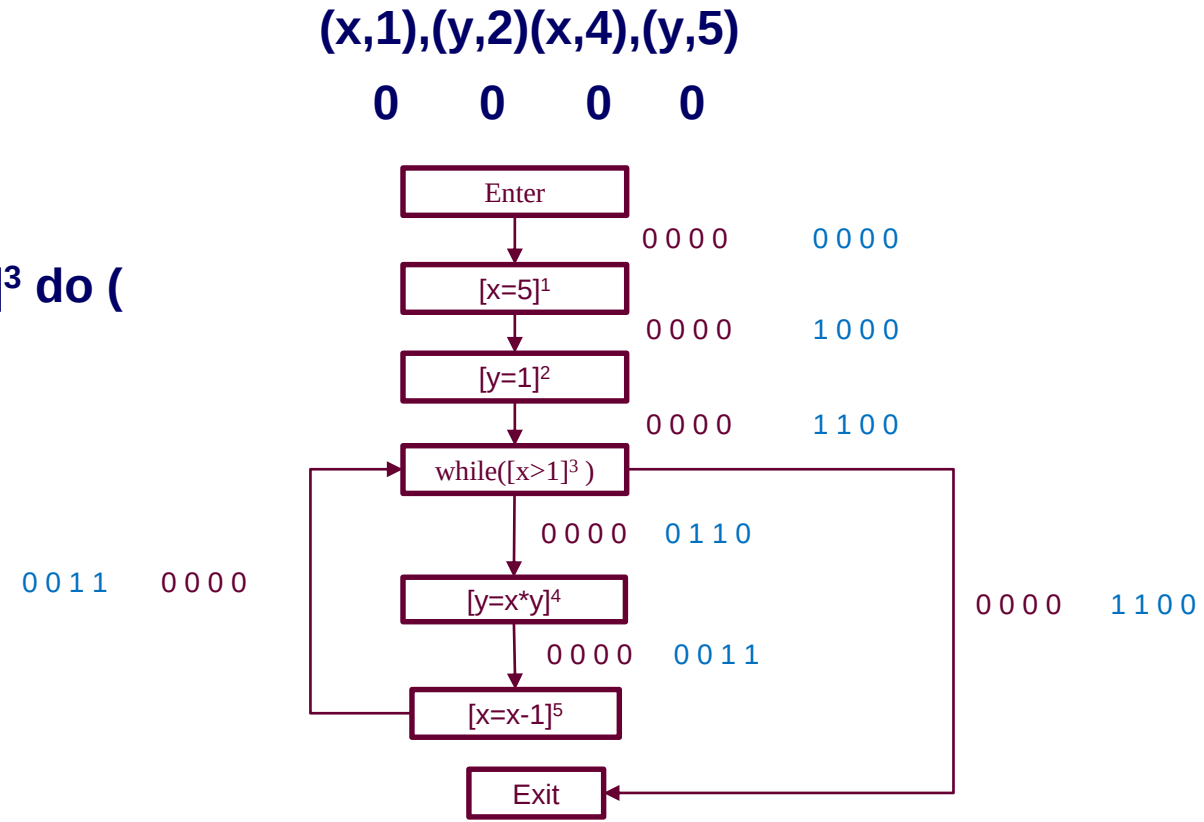
[y=1]^2;

while [x>1]^3 do (

[y=x*y]^4;

[x=x-1]^5

)



重新走一遍到达定值分析

Example:

[x=5]^1;

[y=1]^2;

while [x>1]^3 do (

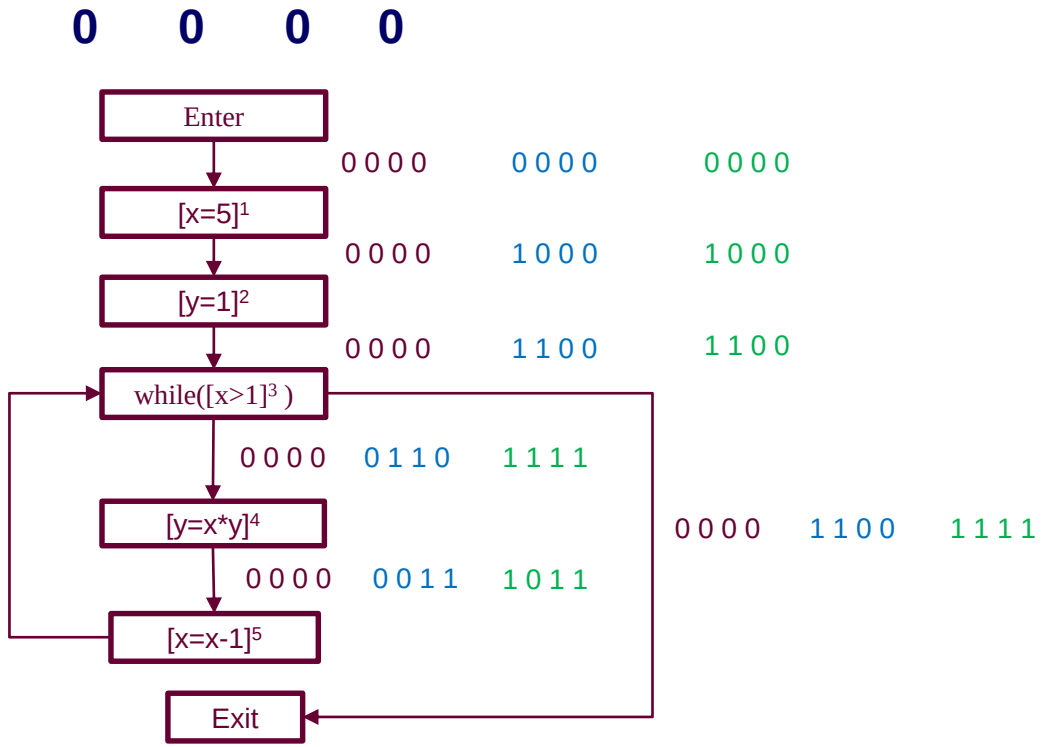
[y=x*y]^4;

[x=x-1]^5

)

(x,1),(y,2)(x,4),(y,5)

Round3



重新走一遍到达定值分析

Example:

[x=5]^1;

[y=1]^2;

while [x>1]^3 do (

[y=x*y]^4;

[x=x-1]^5

)

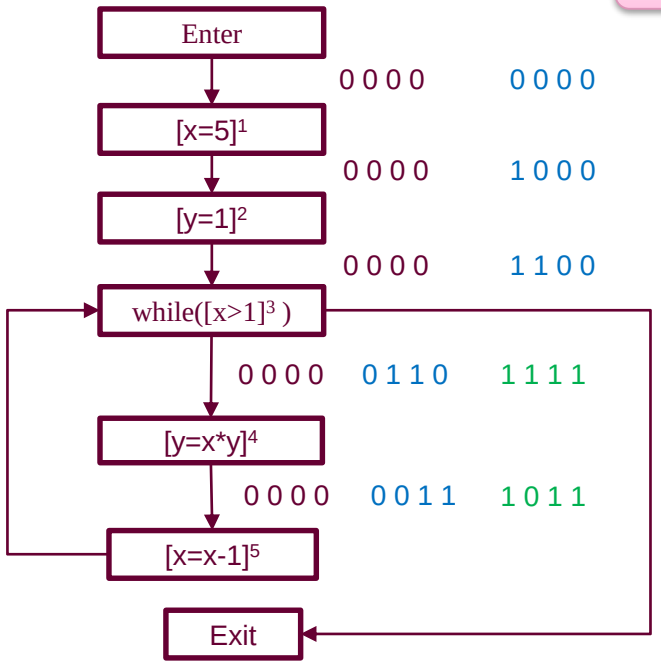
0011 0011 0011 0000

(x,1),(y,2)(x,4),(y,5)

Round4

0 0 0 0

1不会变成0 → 迭代一定会终止



0000 0000 0000 0000

0000 1000 1000 1000

0000 1100 1100 1100

0000 0110 1111 1111

0000 1100 1111 1111

0000 0011 1011 1111

基于格的数据流分析框架

- 数据流分析框架（G,D,L,F）
  - G: 控制流图 (V,E)
  - D: 数据流分析方向，前向&后向
  - L: 半格(S, ⊑)，包括值域集合S和一个交/并操作
  - F: 一组f(S)→ f(S)的转移方程（单调函数）
- 涵盖了四类经典的数据流分析：
  - 前向Must、前向May、后向Must、后向May分析

本次课内容

- 经典程序分析算法
  - 数据流分析：到达定值、活跃变量、非常忙表达式、可用表达式
  - 污点分析
  - 指向分析

信息流分析理论模型

- 为程序中的变量赋予安全类别
  - 高级别(high)和低级别(low)
- 一个格被定义为一个二元组⟨SC, ⊑⟩，其中SC 是一个安全类别的集合; ⊑是建立在 SC 上的偏序关系，代表带有安全类型的变量之间是否可以进行赋值传递。
  - 在SC:{Public,Secret}集合上, Public ⊑ Secret 的偏序关系表示Public级别的变量可以赋值给Secret级别的变量.

污点分析理论模型

- 污点分析Taint Analysis是一类典型的信息流分析，又被称作信息流跟踪技术
- 格⟨SC, ⊑⟩
  - SC:{Tainted;Untainted}
  - ⊑:变量之间是否可以进行赋值传递
  - 如果信息从 Tainted类型变量传播给 Untainted类型的变量,那么需将Untainted类型的数据标成Tainted类型;
  - 如果 Tainted 类型的变量传递到重要数据区域或者信息泄露点,那就意味着信息流策略被违反

污点分析概念

- 污点分析可以抽象成一个三元组⟨sources,sinks,sanitizers⟩
  - source 即污点源,代表直接引入不受信任的数据或者机密数据到系统中;
  - sink即污点汇聚点,代表泄露隐私数据到外界;
  - sanitizer 即无害处理,代表通过数据加密或者移除危害操作等手段使数据传播不再产生危害。
  - 污点分析就是分析程序中由污点源引入的数据是否能够不经无害处理,而直接传播到污点汇聚点

污点传播分析

- 污点传播与程序的控制流、数据流相关。
- 显式流分析
  - 分析污点标记如何随程序中变量间的数据依赖关系传播

```
1  void foo () {
2    int a = source() ,
3    int b = source() ;
4    int x, y;
5    x = a * 2 ;
6    y = b + 4 ;
7    sink(x);
8    sink(y);
9  }
```

显式污点传播

[1] 王蕾,李丰,李炼,冯晓兵.污点分析技术的原理和实践应用.软件学报,2017,28(4):860-882.

污点传播分析

隐式流分析

- 分析污点标记如何随程序中变量间的控制依赖关系传播

```
1  void foo () {
2    String X = source();
3    String Y = new String();
4    for (int i = 0; i < X.length(); i++) {
5      int x = (int)X.charAt(i);
6      int y = 0;
7      for (int j = 0; j < x; j++)  {
8        y = y + 1;
9      }
10     Y = Y + (char)y;
11   }
12   sink(Y);
13 }
```

显式污点传播　　隐式污点传播

[1] 王蕾,李丰,李炼,冯晓兵.污点分析技术的原理和实践应用.软件学报,2017,28(4):860−882.

污点分析方法

- **基于动态分析的技术**
  - 在程序运行过程中实时监控程序。
  - 硬件支持、程序插桩，重点在于降低监控的开销。
- **基于静态分析技术**
  - 函数内：直接赋值传播、别名（指针）传播
  - 函数间：函数调用传播
  - 需要综合运用过程内数据流分析、指向分析、跨过程分析等

应用污点分析检测安卓隐私泄露

- 源点：
  - getDeviceUID()/getPasswordHistoryLength()/…
- 汇点：
  - write()/send()/writeToParcel() /…
- 典型工具：
  - FlowDroid、Amandroid、IccTa、DidFail等

应用污点分析检测安卓隐私泄露

- **研究难点：**
  - 安卓框架的语义分析，如跨组件数据传播
  - 安卓场景中污点和汇点的自动识别
  - 安卓库函数分析，自动摘要构建
  - 精确高效的别名分析
  - ……

代表性工作 FlowDroid

- 基于IFDS数据流分析框架分析污点传播行为
- 通过按需的后向别名分析方法提高分析效率
- 对 Android 生命周期函数进行模拟,构建dummyMain和虚拟节点连接成完整的调用图

Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps CCF A

S Arzt, S Rasthofer, C Fritz, E Bodden, A Bartel... - Acm Sigplan ..., 2014 - dl.acm.org

... In this work we thus present **FLOWDROID**, a novel and ... Novel on-demand algorithms help **FLOWDROID** maintain high ... Android test applications, **FLOWDROID** finds a very high fraction ...

★ Save 99 Cite Cited by 2512 Related articles All 31 versions no code implementation

[1] Steven Arzt et al., FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. PLDI 14.

IFDS数据流分析框架

- IFDS框架是一个上下文敏感和流敏感的数据流分析框架
  - I =过程间(interprocedural)
  - F=有限的(finite)
  - D=满足分配率的(distributive)
  - S=子集合(subset)
- 用于：在有限的值域中、data flow fact需要通过并/交集合操作、满足分配率的问题的求解

IFDS数据流分析框架

- 核心思想：将程序分析问题转化为图可达问题
- 四类图转移边：
  - (1) Call-Flow,即求解函数调用(参数映射)的转移函数.
  - (2) Return-Flow,即求解函数返回语句返回值到调用点的转移函数.
  - (3) CallToReturn-Flow,即函数调用到函数返回的转移函数.
  - (4) Normal-Flow,是指除了上述 3 种函数处理范围之外的语句的转移函数.
- 如果source→ sink可达，则发生隐私泄露

污点分析中的别名

- **FlowDroid的例子**
  - **source到sink可达吗？**
  - **前向污点分析+按需后向别名分析**

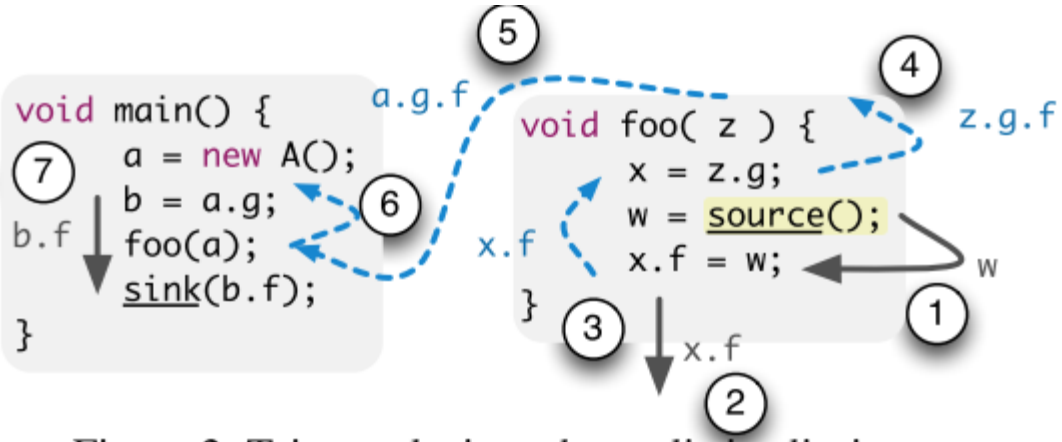


Figure 2: Taint analysis under realistic aliasing

可达性分析：
① w指向敏感数据
② x.f指向敏感数据
③ 寻找x的定义
④ z.g.f指向敏感数据
⑤ a.g.f指向敏感数据
⑥ b.f指向敏感数据
⑦ b.f → sink

本次课内容

- 经典程序分析算法
  - 数据流分析：到达定值、活跃变量、非常忙表达式、可用表达式
  - 污点分析
  - 指向分析

指针分析/别名分析

- **pointer analysis / points-to analysis**
  - 指针可能指向哪块存储单元/空间
  - 变量或域（字段）可能指向哪个对象

- **alias analysis**
  - 某存储单元/空间可能被哪些变量访问

code	pointer analysis	alias analysis
p = new A();	p → new A	
q = p;	q → new A	p -- q

符号执行难以处理的例子

```
void f(int *x, int *y)
{
   *x = 2;
   *y = 1;
   int z = 0;
   if (*x + *y <= 2) {
      *x = *y = 1 / z;
   }
}
```

x和y的指向关系？

```
void g() {
   int x = 0;
   f(&x, &x);
}
```

经典的指向分析算法

- **Anderson 算法**
  - 基于子集约束（subset constraint）求解的流不敏感的指针分析方法，复杂度$O(n^3)$
  - **Step1:** 构造Pointer Flow Graph (PFG)
    - 有向图，标明了对象如何沿着指针流动
    - 节点：变量或对象域
    - 边：指针x→指针y，标明了对象与指针的传递关系
  - **Step2: 将指向关系沿着PFG的边传播**
    - 传播至不动点

指针分析规则

构造PFG

语句种类	语句	指针分析规则	PFG 边
对象创建	$i: x = \text{new } T()$	$\dfrac{o_i = Heap(i)}{o_i \in pt(x)}$	
复制	$x = y$	$\dfrac{o_i \in pt(y)}{o_i \in pt(x)}$	$x \leftarrow y$
字段存储	$x.f = y$	$\dfrac{o_i \in pt(x),\ o_j \in pt(y)}{o_j \in pt(o_i.f)}$	$o_i.f \leftarrow y$
字段读取	$y = x.f$	$\dfrac{o_i \in pt(x),\ o_j \in pt(o_i.f)}{o_j \in pt(y)}$	$y \leftarrow o_i.f$
方法调用	$l: r = x.k(a_1,\cdots,a_n)$	$\dfrac{\begin{array}{c} o_i \in pt(x),\ m = Dispatch(o_i, k) \\ o_u \in pt(a_j),\ 1 \leqslant j \leqslant n \\ o_v \in pt(m_{\text{ret}}) \end{array}}{\begin{array}{c} o_i \in pt(m_{this}) \\ o_u \in pt(m_{p_j}),\ 1 \leqslant j \leqslant n \\ o_v \in pt(r) \end{array}}$	$a_1 \rightarrow m_{p_1}$ $\vdots$ $a_n \rightarrow m_{p_n}$ $r \leftarrow m_{\text{ret}}$

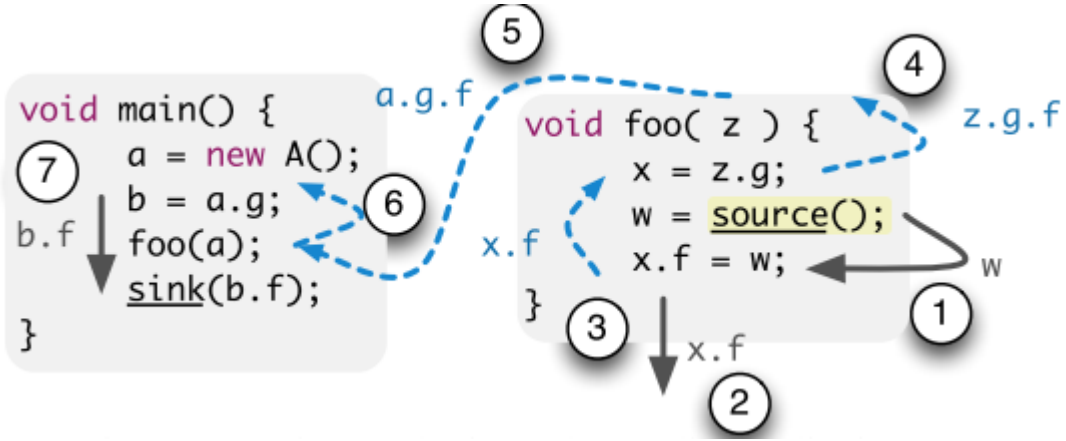
[1]谭 添, 马晓星,许 畅,马春燕,李樾. Java 指针分析综述 谭.软件学报,2023, 60(2):274−293.

简化的指向分析算法

- **Steensgaard算法**
  - 对Anderson算法进行简化
  - 将集合关系简化为相等关系
  - 效率更高，但损失精度

	Anderson算法	Steensgaard算法
x = y	pt(y) ⊆ pt(x)	pt(y) = pt(x)

Anderson 算法实例

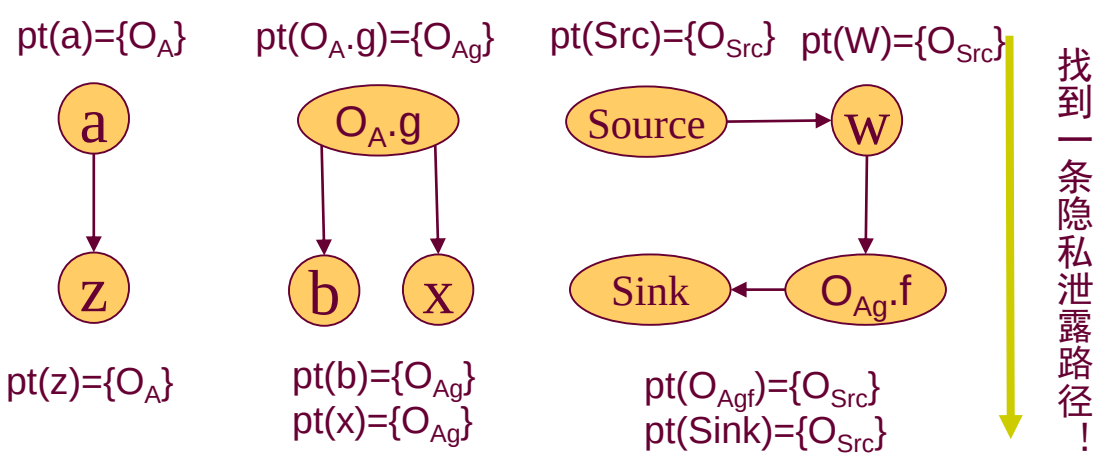


New: a = new A();

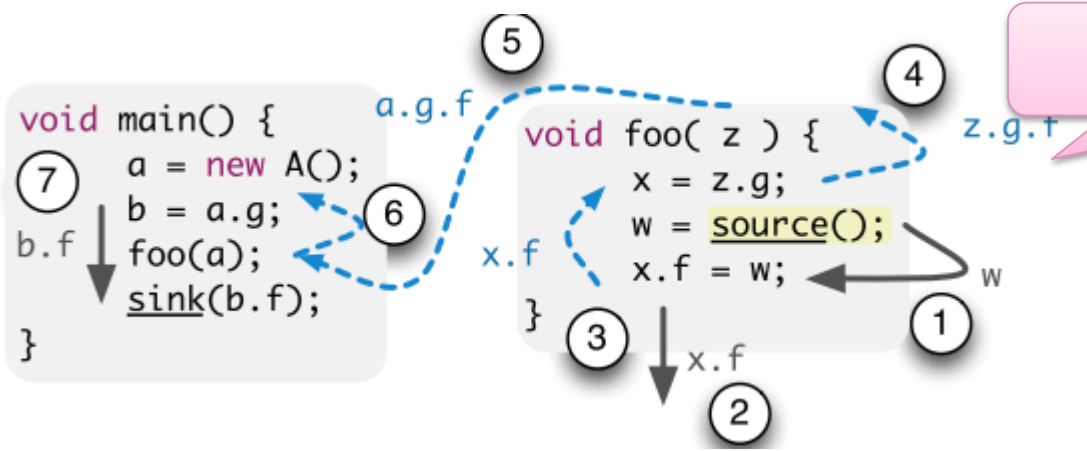
Assign: z=a;
Assign: w= source();

Load: b=a.g;
Load: x=z.g;
Load : sink=b.f;

Store: x.f=w;



Anderson 算法实例



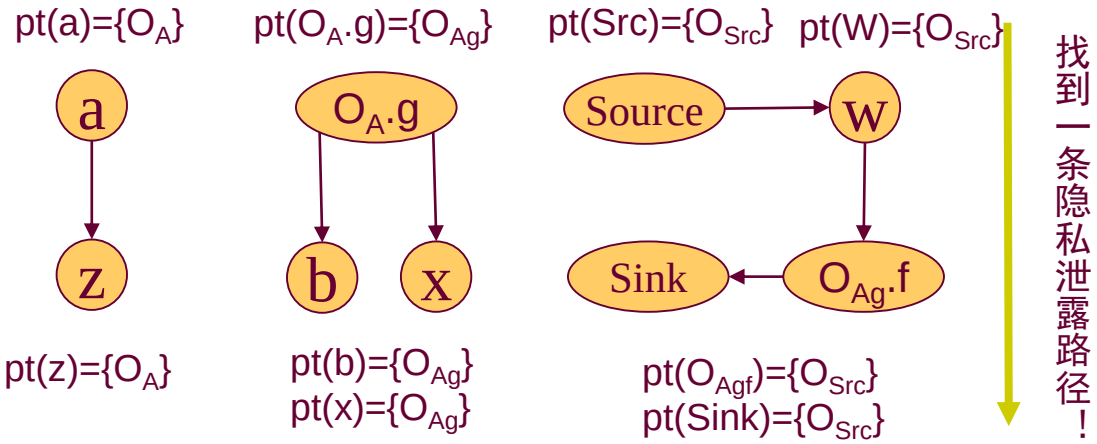
分析的精度如何？

New: a = new A();

Assign: z=a;
Assign: w= source();

Load: b=a.g;
Load: x=z.g;
Load : sink=b.f;

Store: x.f=w;



找到一条隐私泄露路径！

Soot分析框架

- Input：
  - Java源码、jar包、apk…
- 转换为便于分析的中间代码
  - Jimple/Shimple…
- 遍历程序结构
  - Scene/SootClass/SootMethod/Body/Unit…
- 输出
  - 插桩后的程序、分析的结果…

Soot 能提供哪些分析?

- Call-graph construction
- Points-to analysis
- Def/use chains
- Intra-procedural data-flow analysis
- Inter-procedural data-flow analysis
- Aliasing
- Taint analysis

Soot 能提供哪些分析?

Code Analysis Problem	Tools
Bytecode Instrumentation	Soot
Callgraph Construction	Soot + FLOWDROID
Points To Set Construction	Soot + FLOWDROID
Data Flow Analysis	Soot + FLOWDROID
Runtime Value Extraction	Soot + FLOWDROID + HARVESTER
Runtime Environments	Soot + FLOWDROID + FUZZDROID
Library Summaries	Soot + FLOWDROID + STUBDROID
Source / Sink Detection	Soot + SUSI

TABLE I
ANDROID CODE ANALYSIS PROBLEMS AND TOOLS TO SOLVE THEM