
# 第十三章 数据流分析（二）（Dataflow Analysis）

冯 洋


# 什么是数据流分析？

- 在本课程中，我们只介绍基于静态分析的数据流分析技术（有动态的方法，但是不属于本课程关注点）
- 数据流分析通常是基于控制流图（CFG）进行展开的
- 回忆我们《第九章 机器无关的优化》
  - 到达定值分析
  - 活跃变量分析
  - 可用表达式分析


# 什么是数据流分析？

- 为什么数据流分析通常与CFG 一起讨论？有没有其他数据结构适合于数据流分析？


# 什么是数据流分析？

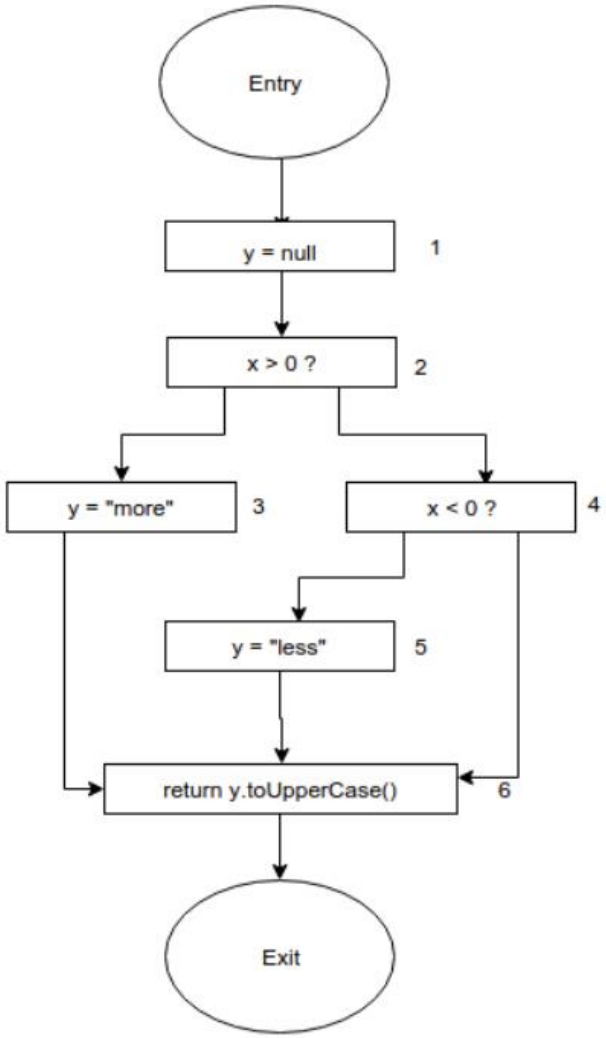
- 为什么数据流分析通常与CFG 一起讨论？有没有其他数据结构适合于数据流分析？
- CFG vs. AST
  - CFG 更加直观简洁
  - CFG 没有较多的冗余（思考冗余的危害）
  - CFG 可以很好地表达BasicBlock信息
  - AST 可以更好地明确错误（或者潜在漏洞信息）的上下文信息及依赖信息


# 数据流分析的经典应用

- 到达定值分析（Reaching Definition）
  - 未初始化变量检测
  - 指针使用错误检测
- 可用表达式分析（Available Expression）
  - 代码优化
- 活跃变量分析（Live Variable）
  - 寄存器分配
- 高活跃性表达式分析（Very Busy Expression）
  - 优化代码规模
  - 提升程序运行效率

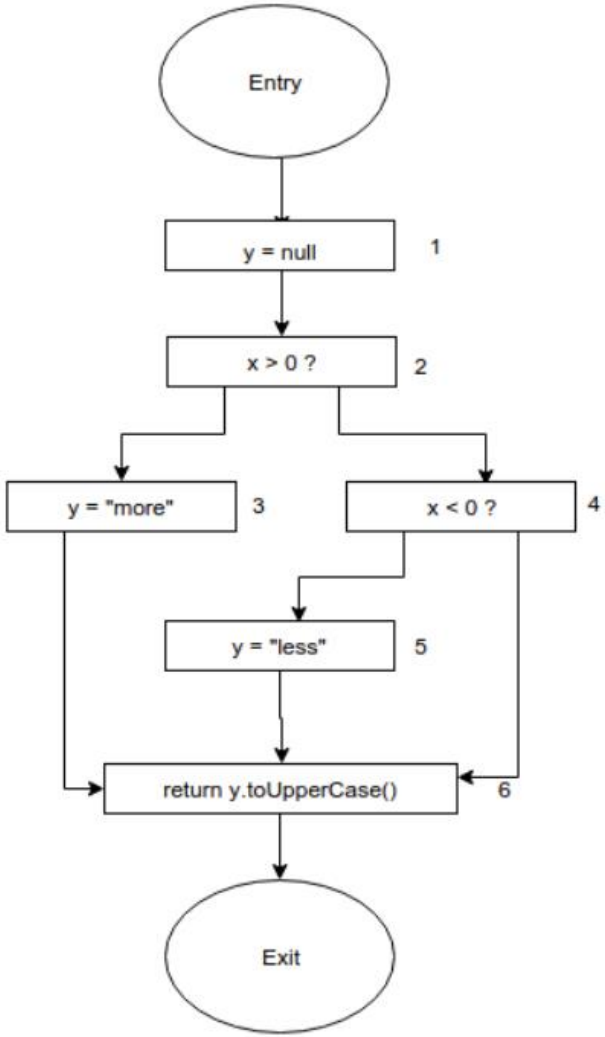
# 数据流分析进一步讨论

```
public String badCode(int x)
{
        String y = null;
        if (x > 0) {
                y = "more";
        } else if (x < 0) {
                y = "less";
        }
        return y.toUpperCase();
}
```



# 数据流分析进一步讨论

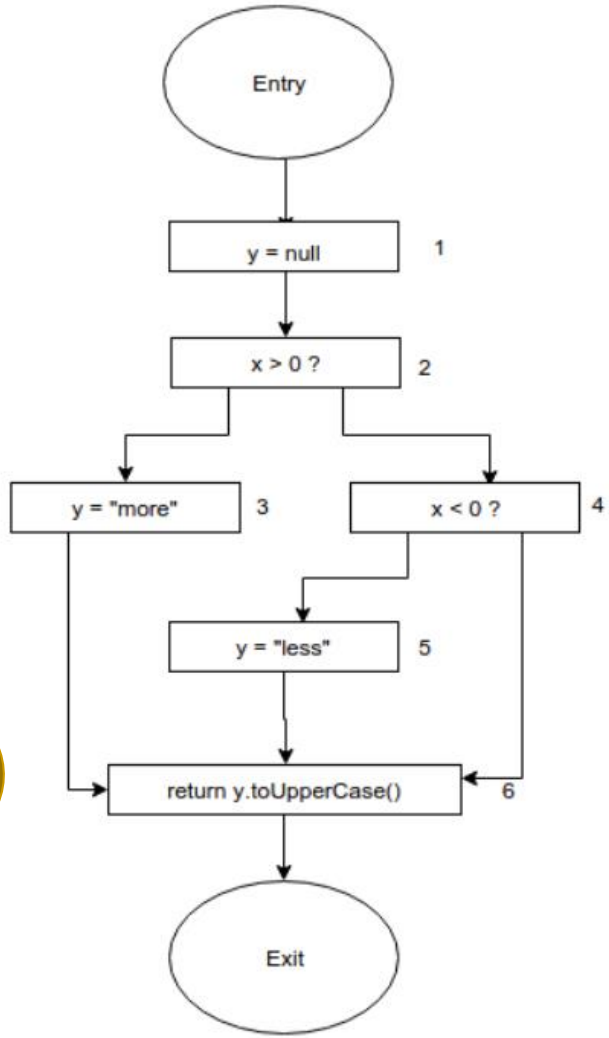
- 通过对局部变量的分析，获得变量对程序的全局影响
- 以右图为例：
  - 我们在程序点1给y赋值了一个null；
  - 我们可以知道在程序点3和5对y进行了又一次赋值；
  - 由此我们可以知道，y可能达到程序点6；
  - 由此我们可以知道，程序中可能存在一个null pointer 异常



# 数据流分析进一步讨论

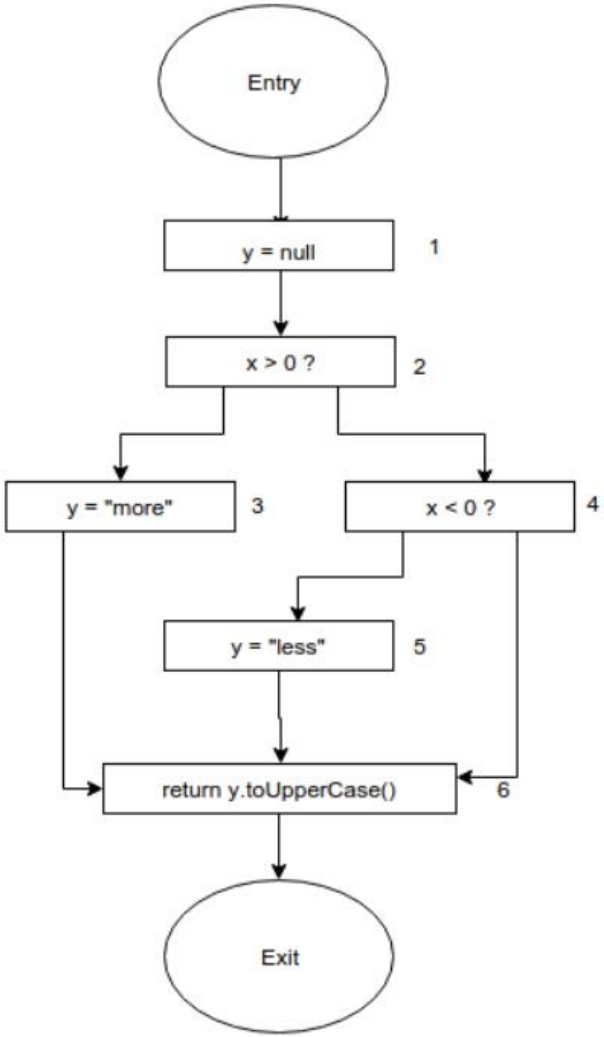
- 通过对局部变量的分析，获得变量对程序的全局影响
- 以右图为例：
  - 我们在程序点1给y赋值了一个null；
  - 我们可以知道在程序点3和5对y进行了又一次赋值；
  - 由此我们可以
  - 由此我们可
    个null poi

例子可能有点意思？那么有没有正式一点的表达？



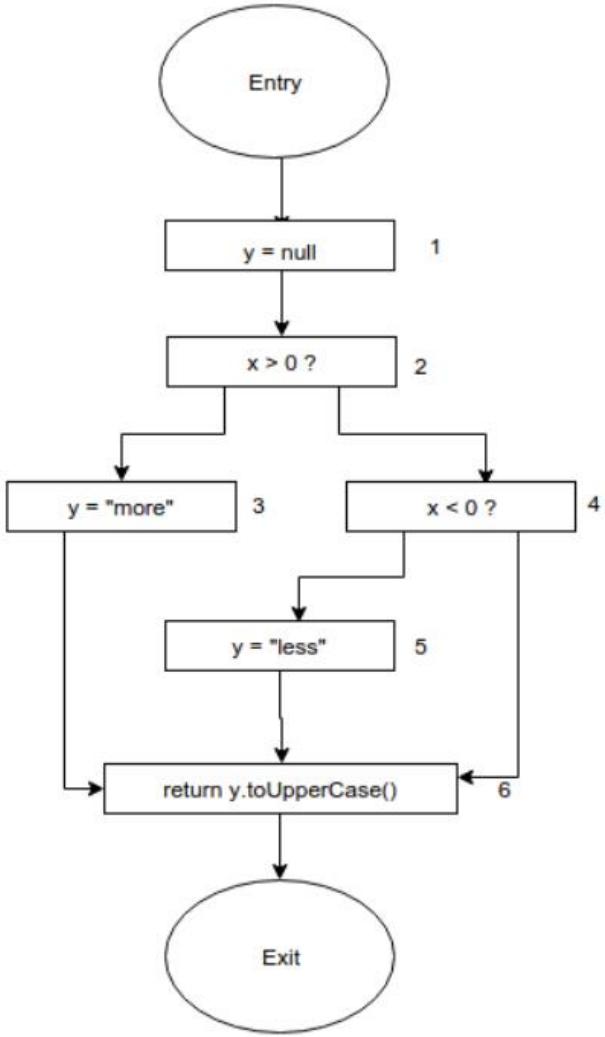
# 数据流分析进一步讨论

- 数据流分析中的关键问题：
  - 向前分析/向后分析（forward or backward）
  - 方法是过程间分析与过程内分析
  - 结论是 may or must link



# 数据流分析进一步讨论

- 在前向分析中，我们主要关注，在程序点 p 之前的节点及其中的数据流变化，分析的内容主要集中在 p 的前驱节点信息；
- 在后向分析中，我们主要关注，在程序点 p 之后的节点及其中的数据流变化，分析的内容主要几种在 p 的后继节点信息；




# 数据流分析进一步讨论

- Must Analysis 主要用于分析与明确：一个变量或者表达式定义D 必然会到达程序点p
  - 如果D在得到p的所有路径上，均出现了至少一次；
  - D中表达的变量在p之前的路径上，没有被kill；
- Must Analysis 提供了一种保障信息
- 例子
  - 常量传播中，我们必须获得传播的常量在某个程序点p的确定情况

# 数据流分析进一步讨论

- May Analysis 主要用于分析与明确：一个变量或者表达式定义D 在某个程序点p的可能状态
- May Analysis 确定了可能性
- 例子
  - 活跃变量分析中，当且仅当一个变量的值在被覆盖前可能被再次使用的时候，我们认为其是活跃的，

# 数据流分析进一步讨论

- 数据流分析问题的定义，从基本块或者程序点的角度来看，我们重点需要思考如下几个问题：
  - 数据流分析技术在开展过程中，我们需要关注的是什么信息？
  - 当我们的目标程序点p存在多个汇合信息流时，我们应该选择什么样的汇合操作符？（并？交？）
  - 在每个代码行的动态行为下，我们的数据信息是怎么改变的？


# 数据流分析进一步讨论

- 我们的一个数据流分析问题，通常情况下包含如下组件：
  - 一个已经完成建模的CFG
  - 研究的变量域D
  - 初始状态
  - 交汇运算符
  - 每一个CFG 节点n，其中的数据流函数


# 数据流分析进一步讨论

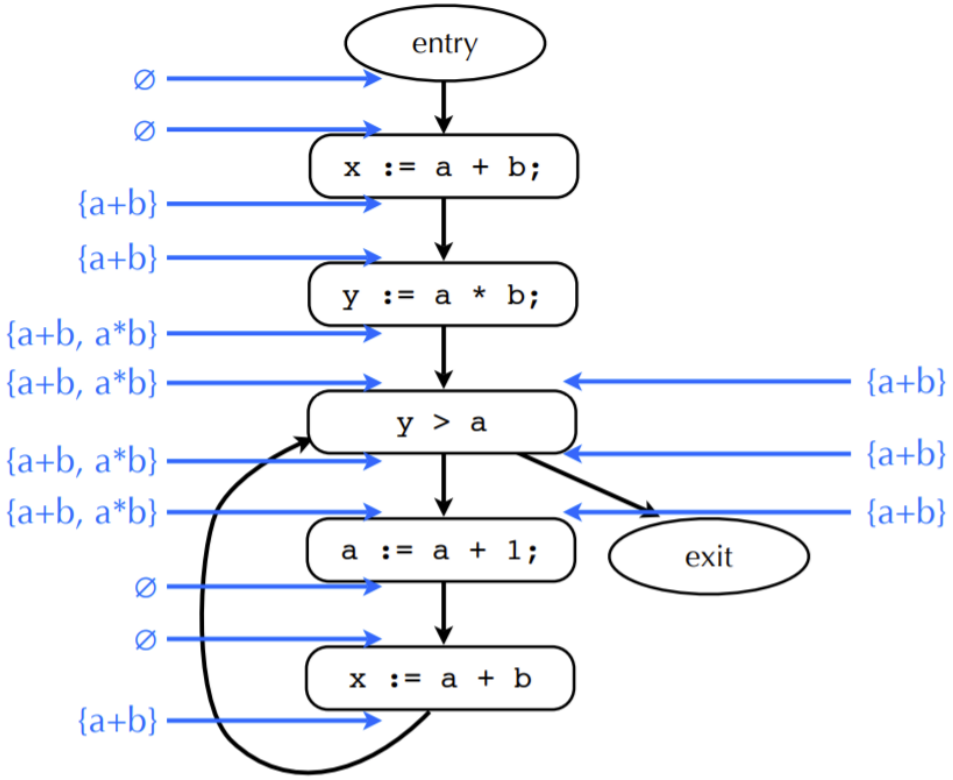
| | **May** | **Must** |
|---|---|---|
| **Forward** | Reaching definitions | Available expressions |
| **Backward** | Live variables | Very busy expressions |

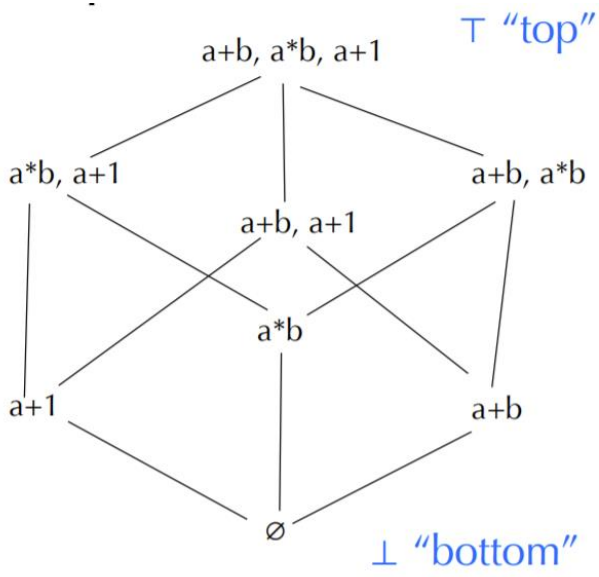
# 数据流分析进一步讨论

- 第九章中介绍的算法，更多地是完成了一种meet-over-all-paths (MOP) 的解决方案。其虽然有效，但是存在一些缺点：
  - 1. 没有考虑到各个节点之间的关系，只考虑了所有路径的可能性，通过遍历路径的朴素算法获得结果；
  - 2. 由于遍历算法的低效性，在实践当中的应用会遇到很多困难；
  - 3.
- 格分析通常被认为是一种更为高效的数据流分析方法

# 数据流分析进一步讨论

- 数据流分析也可以基于Lattice（格？）分析进行展开



# 数据流分析进一步讨论

- 偏序集（poset）表示一个集合P，其中包含偏序关系≤（偏序的概念？）
  - A partial order is a pair (P,≤) such that
    - ≤ is a relation over P (≤ ⊆ P×P)
    - ≤ is reflexive, anti-symmetric, and transitive
  - 偏序结构可以构成一个格，如果他们满足如下条件：
    - 如果P 有一个明确的最大下限，以及一个明确的最低上限
  - ⊓ is the meet operator: x ⊓ y 表示x与y的最大下限：
    - x ⊓ y ≤ x and x ⊓ y ≤ y
    - if z ≤ x and z ≤ y then z ≤ x ⊓ y
  - ⊔ is the join operator: x ⊔ y 表示x与y的最低上限：
    - x ≤ x ⊔ y and y ≤ x ⊔ y
    - if x ≤ z and y ≤ z then x ⊔ y ≤ z

# 数据流分析进一步讨论

- 偏序集非常常见：
  - P表示英语词汇表，如果 ≤ 表示子串关系，那么（P, ≤） 构成了一个偏序集
  - P表示英语词汇表，如果 ≤ 表示“长度小于或等于”关系，那么（P, ≤） 构成了一个偏序集（对吗？）
  - P表示整数集合，如果 ≤ 表示“小于或等于”关系，那么（P, ≤） 构成了一个偏序集（对吗？）

# 数据流分析进一步讨论

- 偏序集非常常见：
  - P表示英语词汇表，如果 ≤ 表示子串关系，那么（P, ≤） 构成了一个偏序集
  - P表示英语词汇表，如果 ≤ 表示“长度小于或等于”关系，那么（P, ≤） 构成了一个偏序集（对吗？）
    - Reflexive: yes.
    - Anti-Symmetric: NO
    - Transitive: yes.
  - P表示整数集合，如果 ≤ 表示“小于或等于”关系，那么（P, ≤） 构成了一个偏序集（对吗？）
    - Reflexive: yes.
    - Anti-Symmetric: YES
    - Transitive: yes.

# 数据流分析进一步讨论

- 完全格（Complete Lattices）：一个格中的所有subset 均包含一个最大下界与一个最小上界，且上下界均在格中，则我们称该格为完全格
- 在每个完全格中，都存在一个“上界元素”与“下界元素”，构成了该格的上界与下界
- 格中的单调函数（monotonic）与分发式函数（distributive）


# 数据流分析进一步讨论

- 单调格（monotonic）与分配格（distributive）
- 单调函数：
  - A function f: L → L (where L is a lattice) is monotonic iff for all x,y in L: x ⊆ y implies f(x) ⊆ f(y).
- 分配函数：
  - A function f: L → L (where L is a lattice) is distributive iff for all x,y in L: f(x meet y) = f(x) meet f(y).
- 一般而言，所有的分配格均是单调格，但是反之不成立