Static Program Analysis (WS 2014)
Saarland University
Computer Science

Exercise Sheet 6

Sebastian Hack
Jan Reineke
Christian Hammer
Tomasz Dudziak

Interprocedural Slicing and Cache Analysis

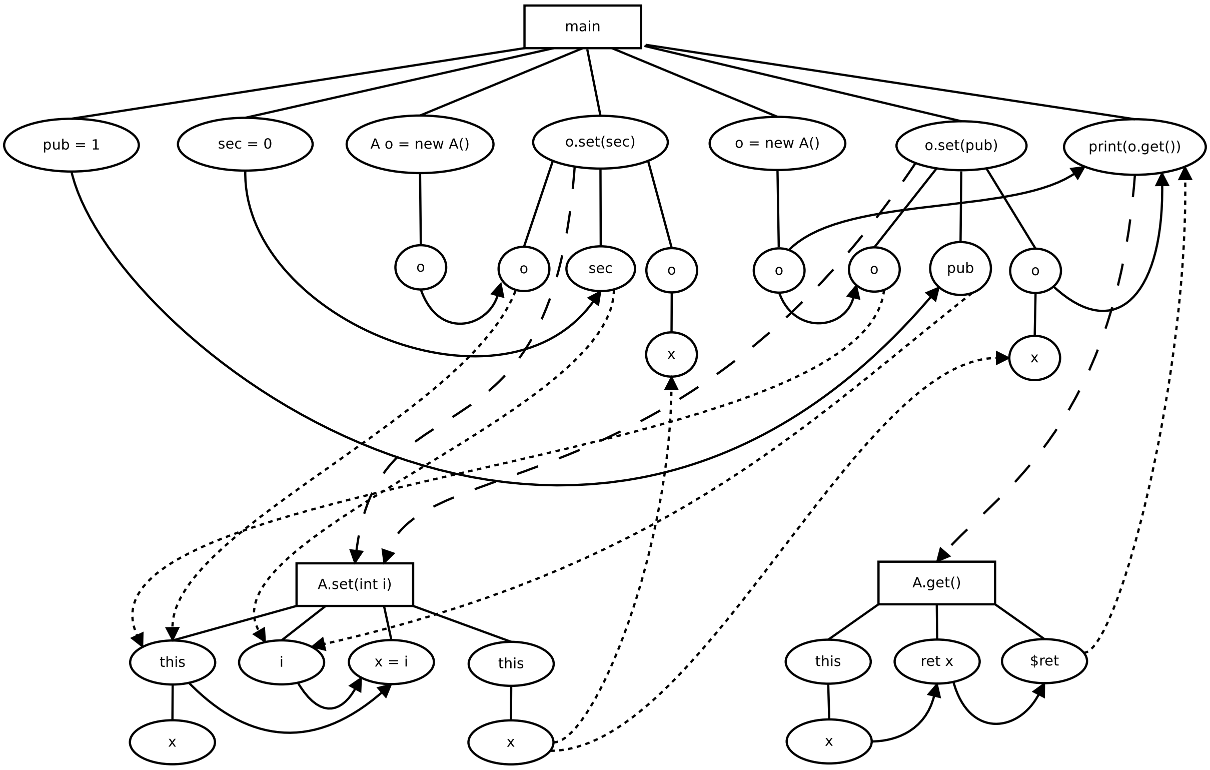

Figure 1: Control flow graph for the example program

Exercise 6.1: 6 points

Figure 1 contains a simplified PDG of a part of the sample Java program introduced in the lecture. Extend and analyze the PDG as follows.

1. Compute the context-*insensitive* backwards slice of the final call to `print(o.get())`.
2. Explain why the slice you just computed is imprecise.
3. Extend the graph by *summary edges*. A summary edge represents a path from a formal parameter to the return value inside the invoked function.
4. Compute a slice using the same slicing criterion as in point 1 but with the context-*sensitive* 2-phase slicer. Which nodes are marked in the first phase, which in the second?

Exercise 6.2: 6 points

Consider the following program.

```
read a;
read b;
read a;
if (a>b) {
    read c;
    read d;
} else {
    read e;
    read f;
}
read x;
read a;
```

1. Perform a *"may"* and a *"must"* cache analysis of this program assuming an LRU-cache with associativity 4 that is empty at the start of the program. Is it possible to determine whether the last access to a results in a cache hit or a cache miss? Does this change if we assume that the initial cache state is unknown?

2. We now assume that the cache uses the FIFO replacement policy. Could an analysis determine whether the last access to a results in a cache hit or a cache miss if the cache is empty at the start of the program? Does this change if we assume that the initial cache state is unknown?