Applied Static Analysis

Interprocedural, Finite, Distributive, Subset Problems (IFDS problems)

Software Technology Group
Department of Computer Science
Technische Universität Darmstadt
Dr. Michael Eichberg

> If you find any issues, please directly report them: GitHub

Some of the images on the following slides are inspired by slides created by Eric Bodden.

For background information consult the seminal paper on IFDS by Reps et al. [1]
If you want to implement it, it is also worth reading the paper by Naeem et al. [2]

IFDS

- Solves a large class of interprocedural dataflow-analysis problems precisely in polynomial time by transforming them into a special kind of *graph-reachability problem*.
- Restrictions:
  - the set of dataflow facts must be a finite set
  - the dataflow functions must distribute over the confluence operator (either union or intersection).
- Examples:
  - reaching definitions
  - available expressions
  - live variables
  - **taint flow analysis**

Recall: distributive means: $f(a) \cup f(b) = f(a \cup b)$

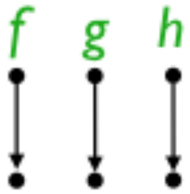
Graph-reachability problem: reachability along interprocedurally realizable paths. A realizable path mimics the call-return structure of a program's execution, and only paths in which "returns" can be matched with corresponding "calls" are considered.

IFDS - core idea

- use the methods' CFG as a foundation to build one supergraph spanning the entire program; that graph has a unique entry point
- *we say*: a fact f holds at stmt s ⇔ node (s,f) is reachable

IFDS - flow functions

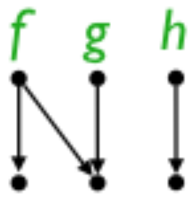
Identity: every fact is reachable if and only if it was reachable before.



IFDS - flow functions

Every fact is reachable if and only if it was previously reachable, and g is also reachable if f was reachable before.

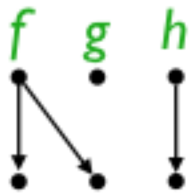
$$out(s) = \begin{cases} in(s) \cup Set(g) & \text{if } f \in in(s) \\ in(s) & \text{otherwise} \end{cases}$$



IFDS - flow functions

Every fact except g is reachable if and only if it was previously reachable; g is only reachable if f was reachable before.

$$out(s) = \begin{cases} in(s) \cup Set(g) & \text{if } f \in in(s) \\ in(s) \backslash Set(g) & \text{otherwise} \end{cases}$$



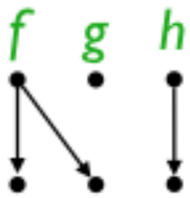
IFDS - flow functions

Possible application: taint analysis.

$$out(s) = \begin{cases} in(s) \cup Set(g) & \text{if } f \in in(s) \\ in(s) \backslash Set(g) & \text{otherwise} \end{cases}$$

Corresponding code:

```
g = f;
```

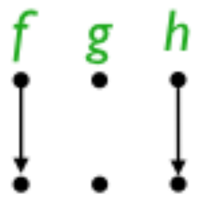


g is tainted(reachable) if and only if f was previously tainted.

IFDS - flow functions (killing values)

Even if g was reachable before, it now no longer.

$out(s) = in(s) \backslash Set(g)$



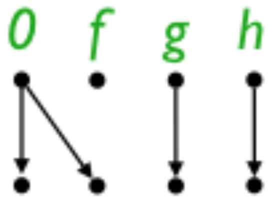
IFDS - flow functions (generating values)

0 is the tautological fact; it is always reachable; even if f was unreachable before, it now reachable.

$out(s) = in(s) \cup Set(f)$

Corresponding code:

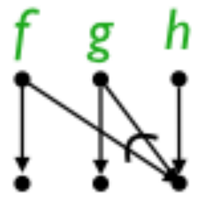
```
f = secret();
```



IFDS - *illegal* flow functions

Not distributive (e.g., full constant propagation); cannot be represented by IFDS:

$$out(s) = \begin{cases} in(s) \cup Set(h) & \text{if } Set(f,g) \subseteq in(s) \\ in(s) & \text{otherwise} \end{cases}$$



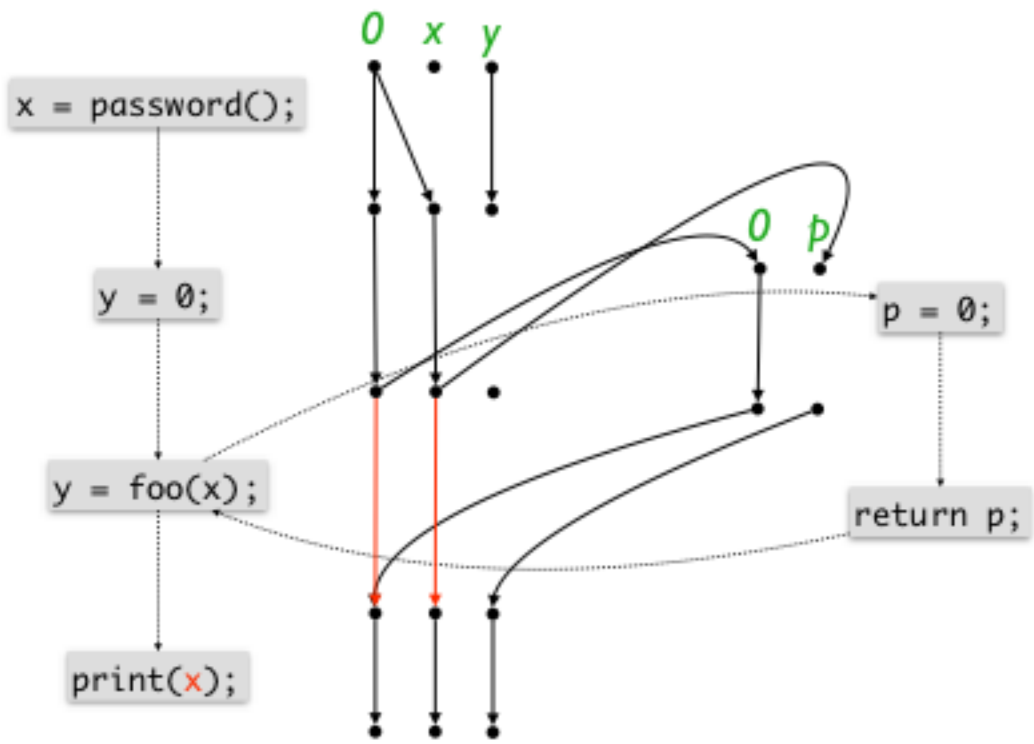
Exploded Supergraph

- A program is represented using a directed graph $G^x = (N^x, E^x)$ called a supergraph.
- G consists of a collection of flow graphs $G^1, G^2, \ldots$ (one for each procedure), one of which, G_{main}, represents the program's main procedure.
    - Each flowgraph G has a unique start node s_P , and a unique exit node e_p.
    - The other nodes of the flowgraph represent the statements and predicates of the procedure in the usual way, except that
    - a **procedure call is represented by two nodes, a call node and a return-site node.**
      (This is usually not explicitly implemented.)
- In addition to the ordinary intraprocedural edges that connect the nodes of the individual flowgraphs, for each procedure call, represented by call-node c and return-site node r, G^x has three edges:
    - An intraprocedural call-to-return-site edge from c to n
      (Most often just the identity function.)
    - An interprocedural call-to-start edge from c to the start node of the called procedure;
    - An interprocedural exit-to-return-site edge from the exit node of the called procedure to r.

Exploded Supergraph - example

```
void main() {
    int x = password();
    int y = 0;

    y = foo(x);

    print(y);
}

int foo(int p) {
   p = 0;

   return p;
}
```

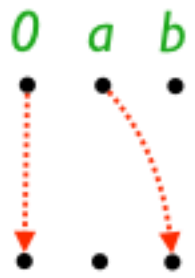
Exploded Supergraph - example cont'd



On-the-fly algorithm

- Pre-computing the entire exploded super-graph is typically too expensive and not required
- Idea: compute only the fragment actually reachable from $(0, s0)$
  (Compute this fragment on the fly.)
- **Store procedure summaries** once they have been computed.

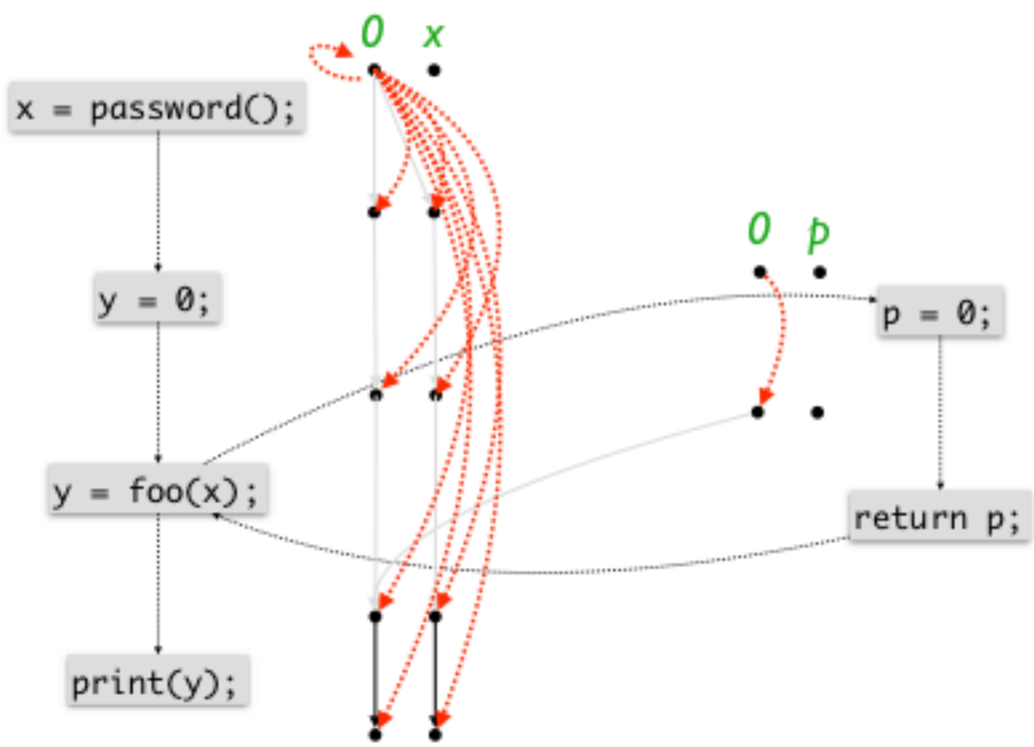
Summary Edges



- This means: *in any context in which a holds before the call, it is true that b holds after the call*.
- 0 always holds, can be represented implicitly

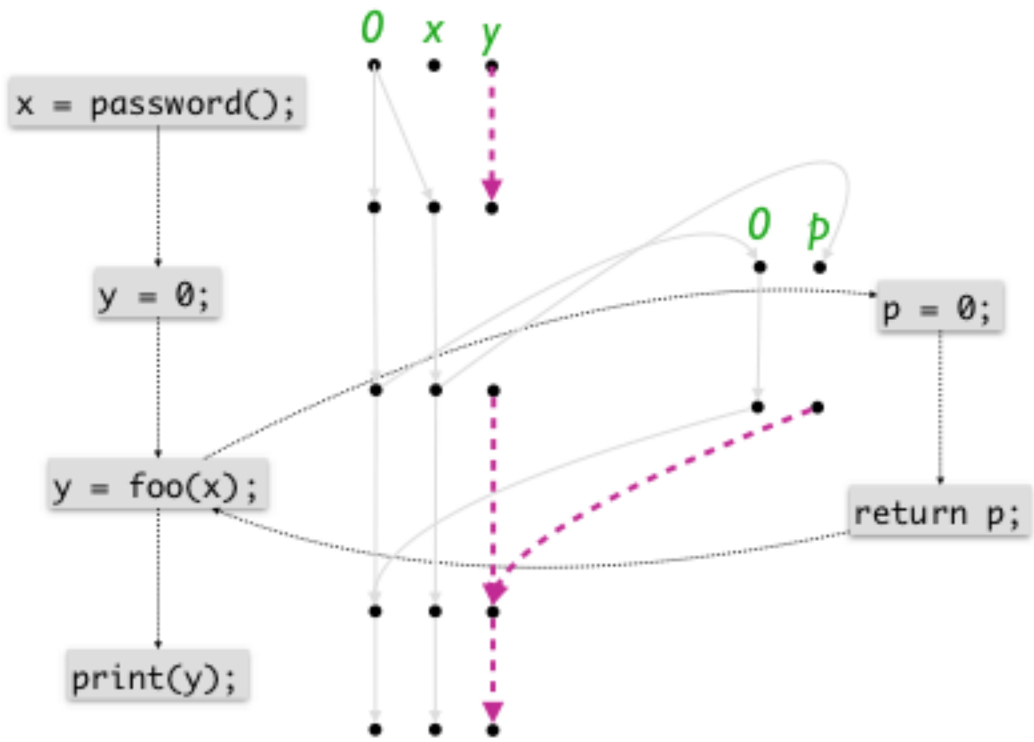
Exploded Supergraph - example cont’d

The red, dotted flow functions represent our summaries.



Exploded Supergraph - example cont’d

The red, dashed flow functions were never computed.

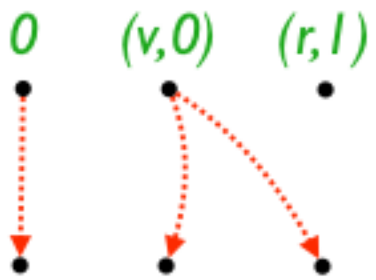


Limitations

The domain has to be (reasonably) finite.

(Counter)Example: linear constant propagation

```
r = v+1
```



> ... in any context in which v is 0 before the call, it is true that r is 1 after the call.

Though the set of int-typed values is finite, it is far too large to be practical!

References

1. Reps, T., Horwitz, S., & Sagiv, M. (1995). Precise interprocedural dataflow analysis via graph reachability. the 22nd ACM SIGPLAN-SIGACT symposium (pp. 49–61). New York, New York, USA: ACM. http://doi.org/10.1145/199448.199462↩
2. Naeem, N. A., Lhoták, O., & Rodriguez, J. (2010). Practical Extensions to the IFDS Algorithm. In Aliasing in Object-Oriented Programming. Types, Analysis and Verification (Vol. 6011, pp. 124–144). Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-11970-5_8↩