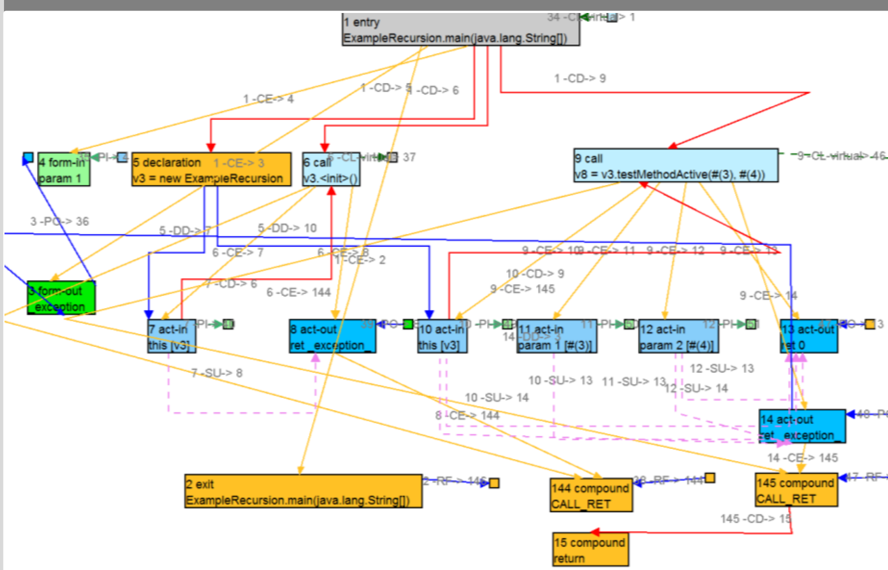
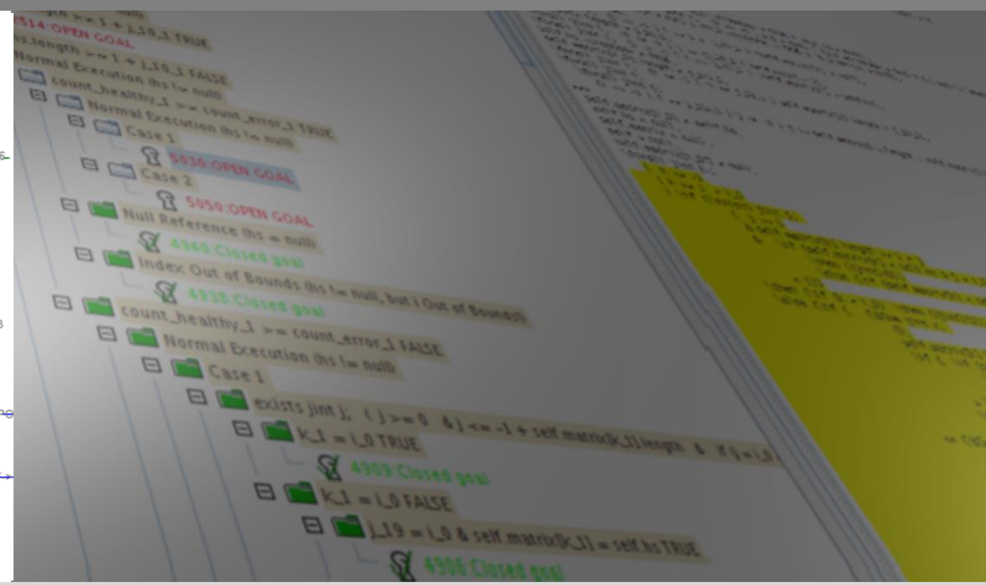
Karlsruhe Institute of Technology

Combining Graph-Based Information-Flow Analysis with KeY for Proving Non-Interference

KeY Symposium | 27.07.2016

INSTITUTE FOR APPLICATION-ORIENTED FORMAL VERIFICATION , FACULTY OF INFORMATICS





Agenda

Motivation

Objective

Preliminary

Combined Approach

Demonstration

Conclusion and future work

Motivation

- Current hybrid approach needs high degree of user interaction
- Program code has to be manually modified

- Proving of functional properties
- But KeY is capable of creating information flow proofs

There should be a way to use KeY’s information flow capabilities in a hybrid approach.

Objective

Development and implementation of an approach, that can prove non-interference for complex systems

? Status Quo

- Two types of tools for information flow control
- Joana runs automatic but creates false positives
- KeY proofs are precise but interactive and time-costly

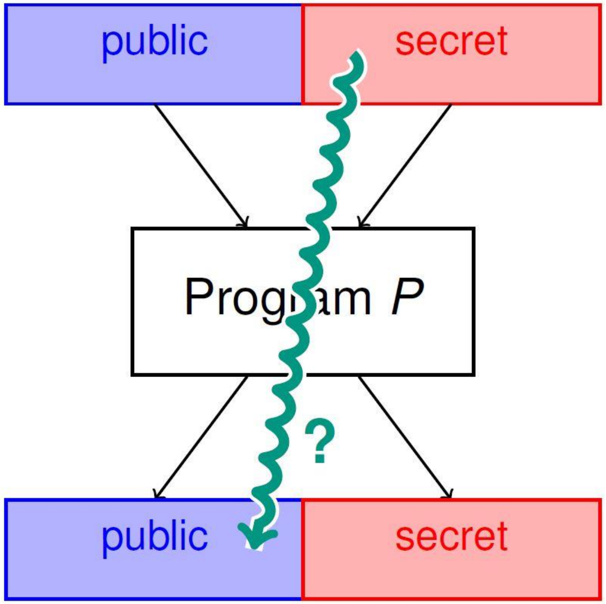
! Objective

- Combined approach for information flow proofs
- The approach should be automatic and precise
- KeY is called for as few as possible methods

Objective: Creation of an approach that creates *automatic* and *precise* information flow proofs.

Preliminary – Information Flow

- Observation of an information flow
- No flow from secret input to public output
- Guarantees End-to-End Security



Source: KlUl15

Preliminary – Non-Interference

Non-Interference

- A variation of the secret input must not lead to a variation of the public output.

$$\forall h_1, h_2, l\colon p(h_1, l) = p(h_2, l)$$

Source: SchSch12

- Example:

if $l = 5$ **then**
  $h \leftarrow h + 1$
else
  $l \leftarrow l + 1$

Secure, the results of l only depends on l

Preliminary – Non-Interference

Non-Interference

- A variation of the secret input must not lead to a variation of the public output.

$$\forall h_1, h_2, l \colon p(h_1, l) = p(h_2, l)$$

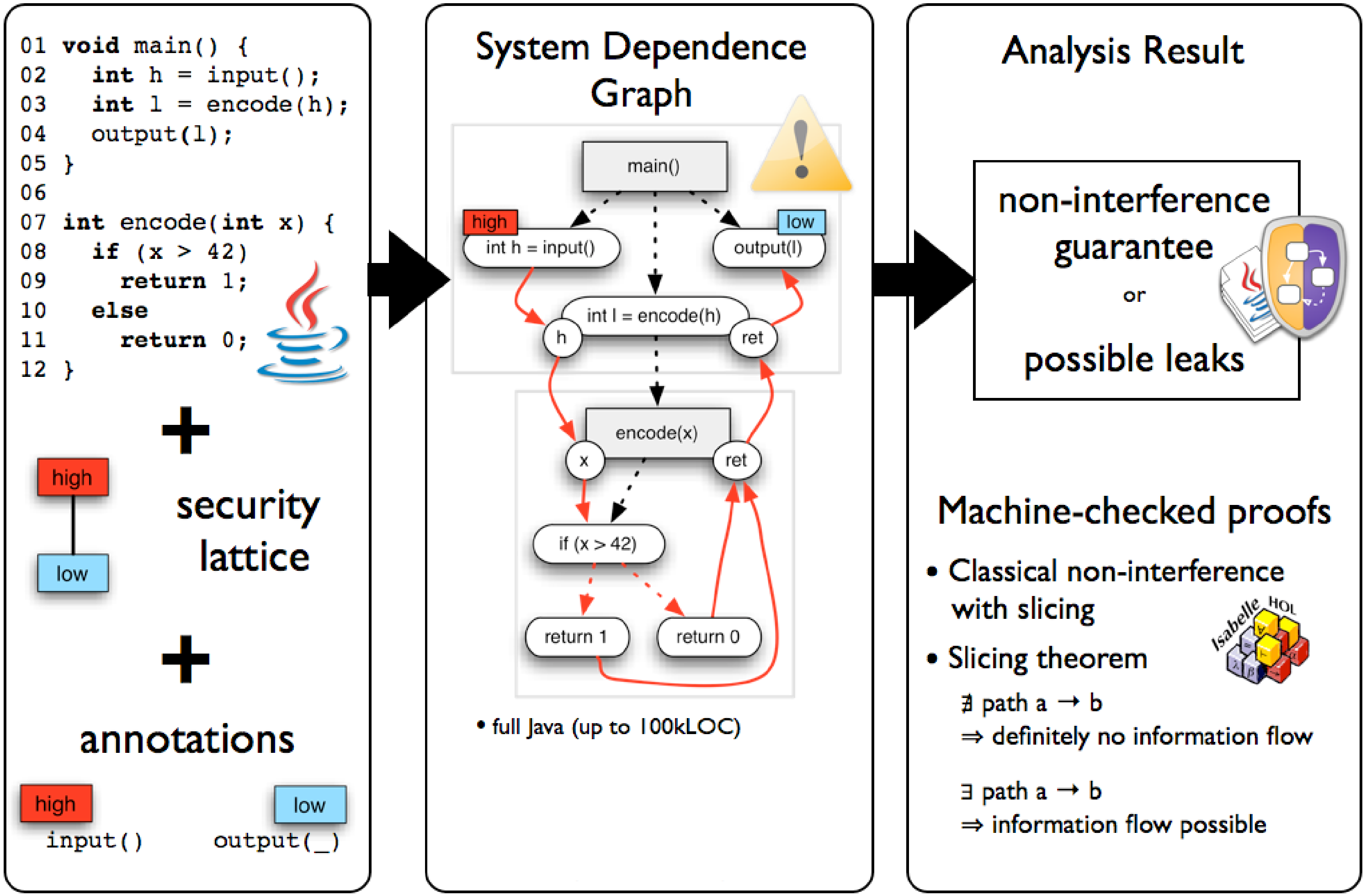
Source: SchSch12

- Example:

$$\left.\begin{array}{ll} 1: & \textbf{if } h = 3 \textbf{ then} \\ 2: & \quad l \leftarrow 5 \\ 3: & \textbf{else} \\ 4: & \quad skip \end{array}\right\}$$

Not secure, because the result of l depends on h

Preliminary – Joana

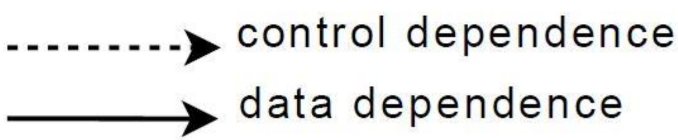


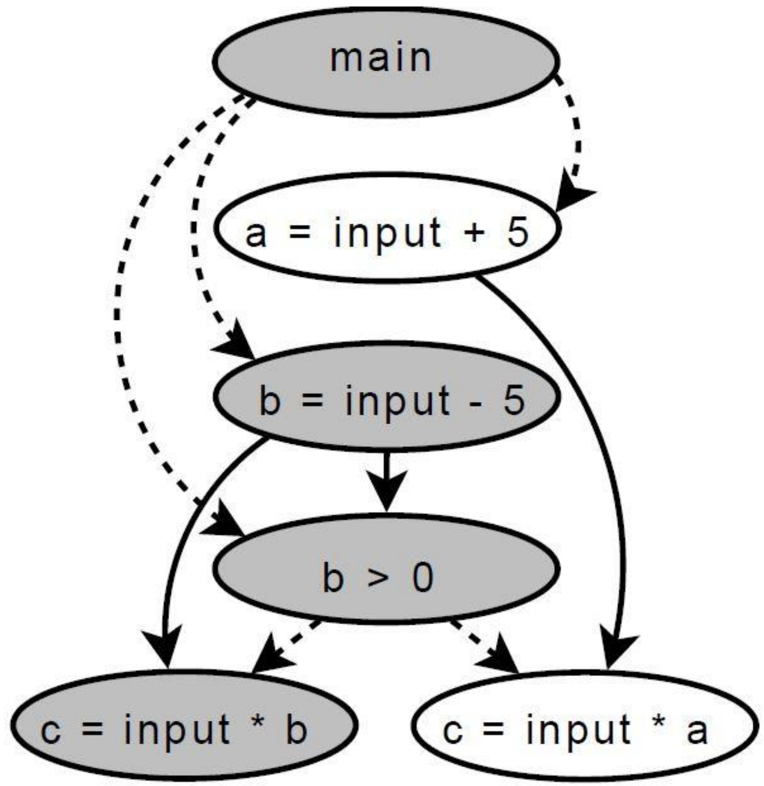
Source: Joa16

Preliminary – Joana

- Program Dependency Graph:

```
void main():
  a = input + 5;
  b = input − 5;
  if b > 0
    c = input ∗ b;
  else
    c = input ∗ a;
```





Source: Griff12

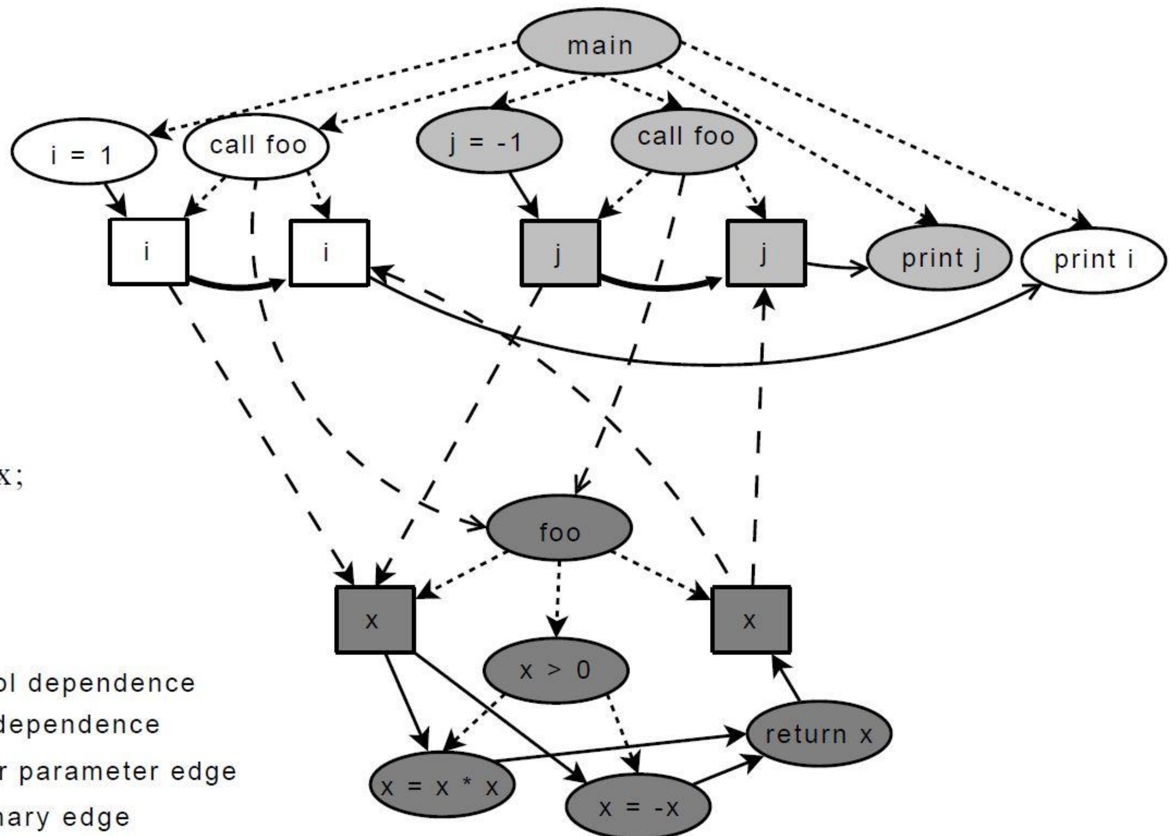
Preliminary – Joana

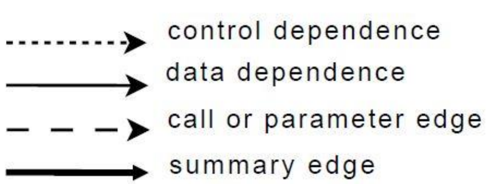
Karlsruhe Institute of Technology

- Extension of PDG‘s are System Dependency Graphs (SDGs)

```
void main():
    i = 1;
    j = -1;
    i = foo (i);
    j = foo (j);
    print i;
    print j;

int foo(x):
   if (x > 0)
      x = x * x;
   else
      x = -x;
   return x;
```





Source: Griff12

Preliminary – Joana

Karlsruhe Institute of Technology

Summary Edges

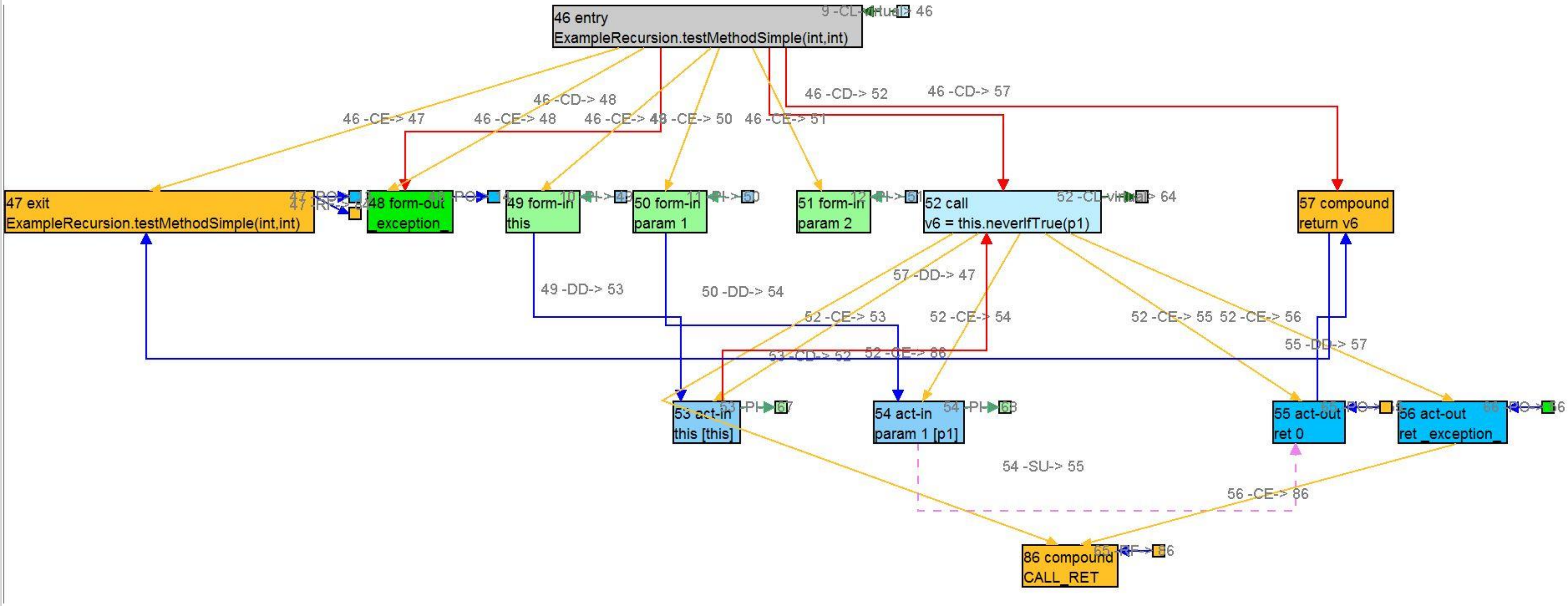
- Additional edge between actual-in and actual-out nodes
- Represent transitive flow from a parameter to a return value

Preliminary – Joana

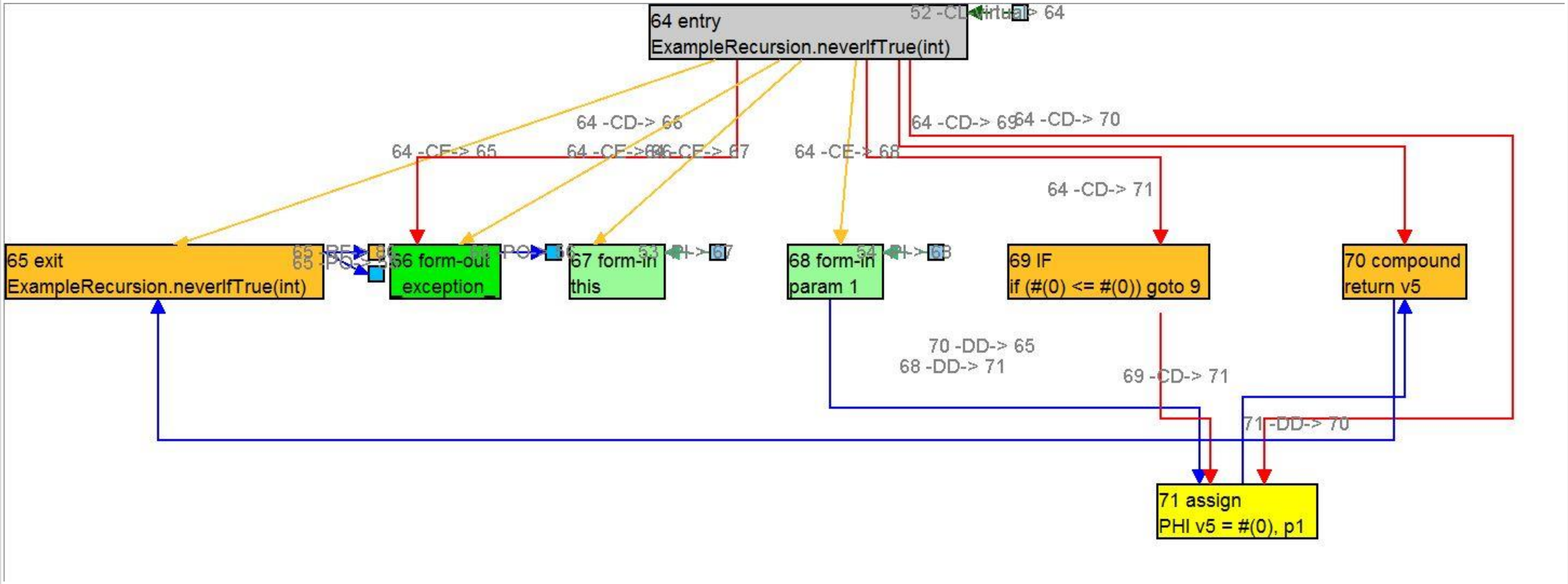
```
public int testMethodSimple(int high, int low) {
 low = neverIfTrue(high);
 return low;
}

public int neverIfTrue(int high) {
 int x = 0;
 if (x > 0) {
  x = high;
 }
 return x;
}
```

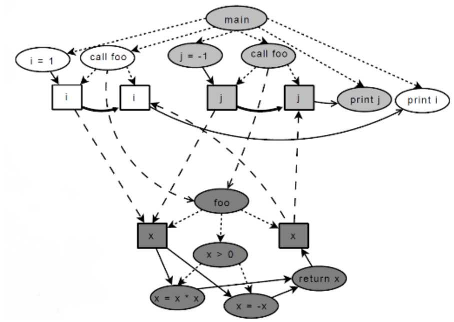
Preliminary – Joana



Preliminary – Joana



Combined Approach

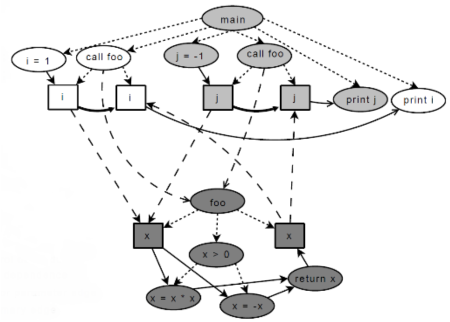


Joana

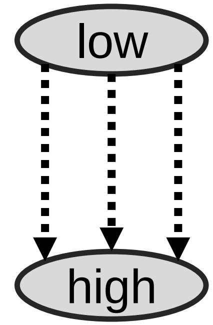
SDG

Combined Approach

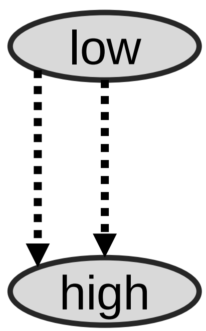
Joana



SDG



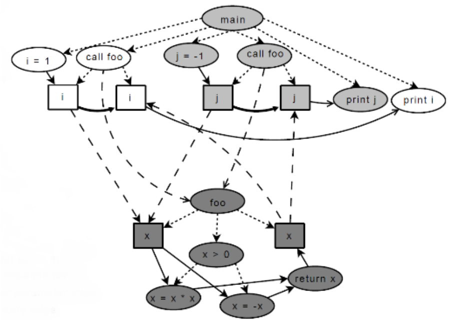
All path from low to high



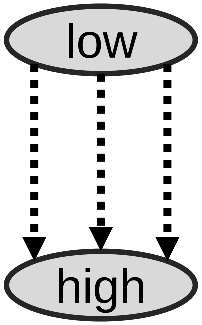
Validate summary edges

Combined Approach

Joana

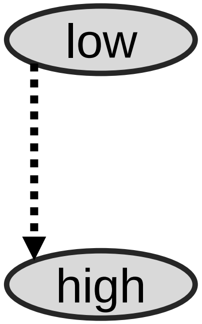


SDG



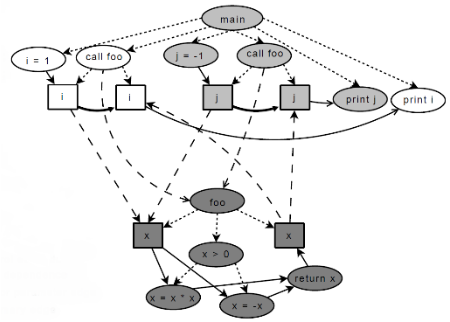
All path from low to high

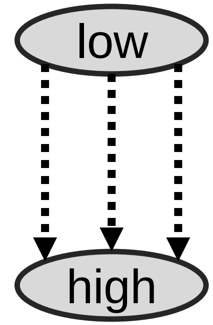
Validate summary edges



Information Flow leak

Combined Approach





Joana

SDG

All path from low to high

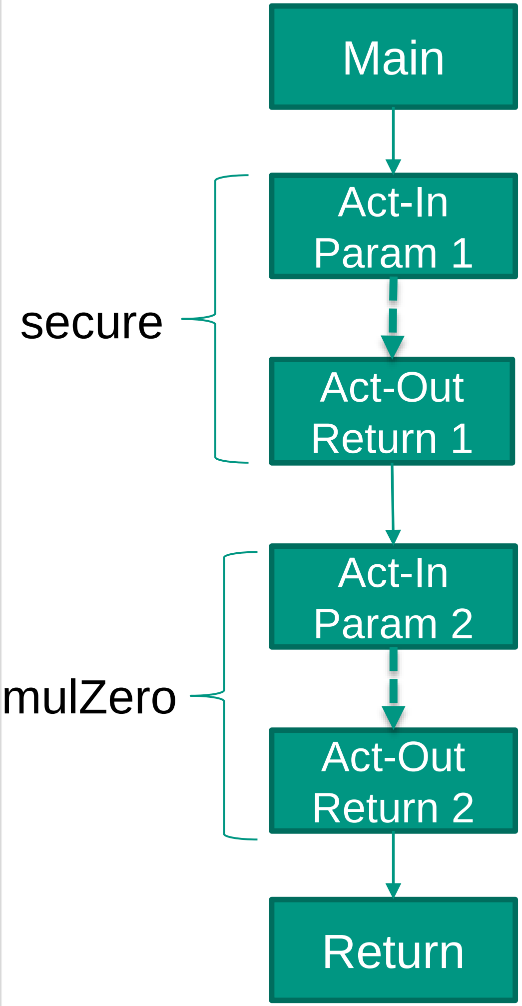
Non-Interference guarantee

low

high

Validate summary edges

Combined Approach - Distinction of cases

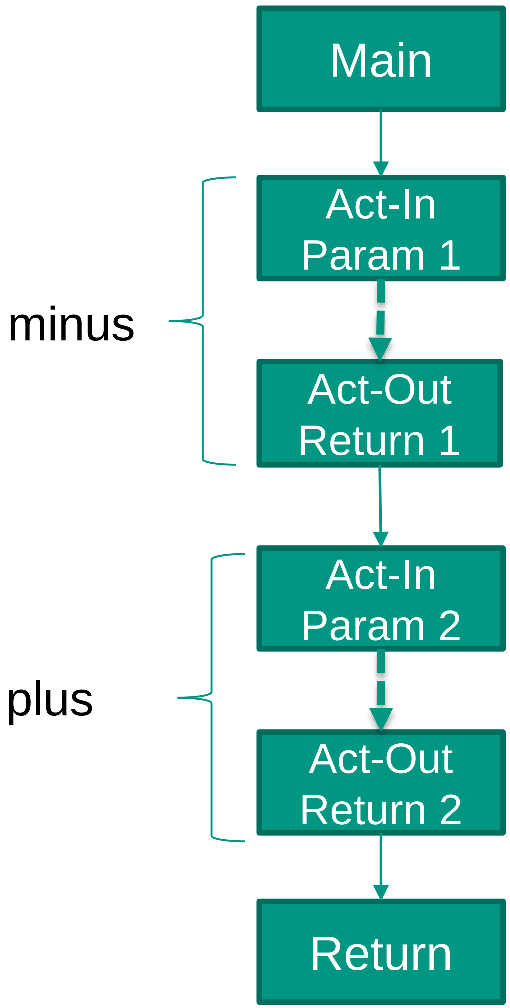



```
public int testMethodTwoAfter(int high, int low) {
    low = secure(high);
    low = mulZero(high);
    return low;
}

private int secure(int low) {
    return low;
}
public int mulZero(int high) {
    return 0 * high;
}
```

- The path is interrupted if we can prove non-interference for one of the methods

Combined Approach - Distinction of cases



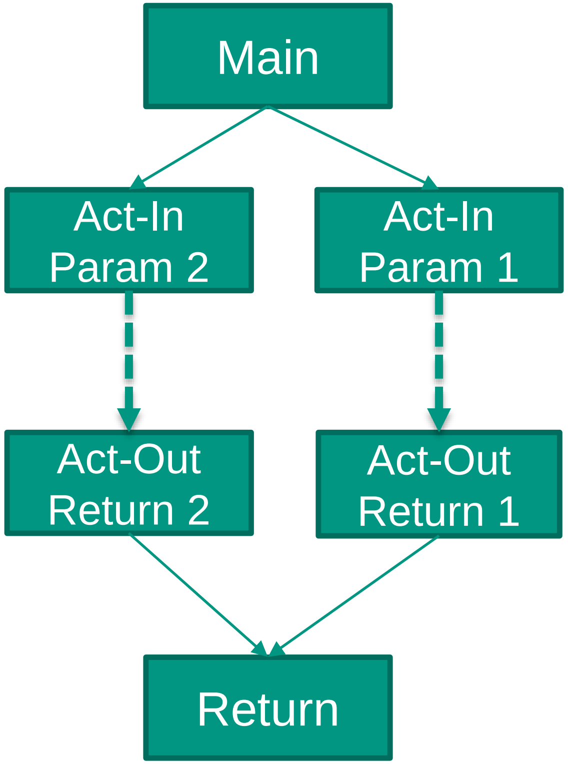

```
public int testMethod(int high, int low) {
   low = plus(low, high);
   low = minus(low, high);
   return low;
}

private int minus(int low, int high) {
   low = low - high;
   return low;
}

private int plus(int low, int high) {
   low = low + high;
   return low;
}
```

- It can be that the two methods together have to be proven

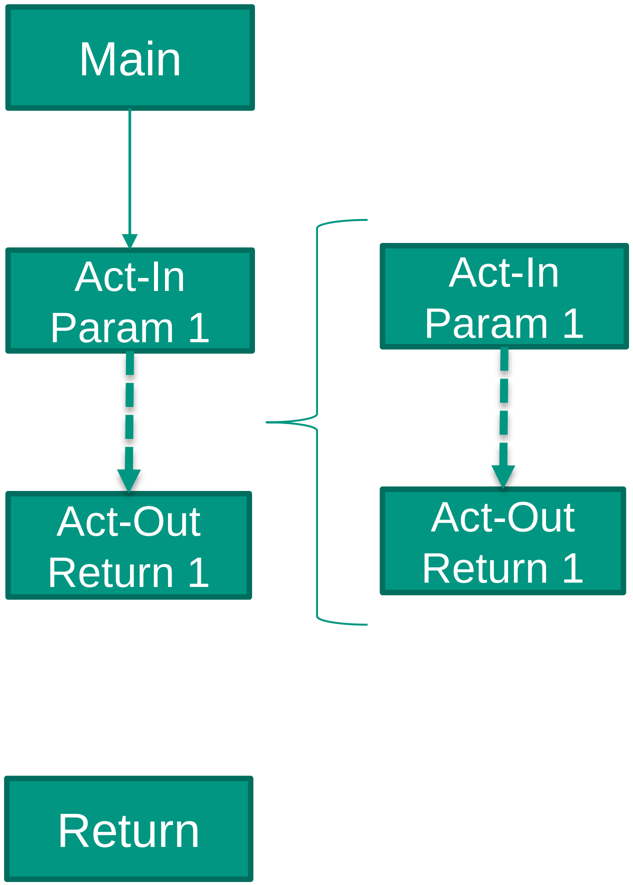
Combined Approach - Distinction of cases




```java
public int testMethodAdd(int high, int low) {
    int i = method1(high);
    int j = method2(high);
    return i + j;
}
```

- Two methods are called independently and are both relevant to the result
- Non-Interference has to be proven for both methods

Combined Approach - Distinction of cases




```java
public int testMethodOneInOther(int high, int low) {
    low = inSecure(low, high);
    return low;
}

private int inSecure(int low, int high) {
    int i = 5;
    low = mulZero(low, high) + i;
    return low;
}

public int mulZero(int low, int high) {
    low = 0 * high;
    return low;
}
```

- We always try to delete summary edges bottom up

Combined Approach

Karlsruhe Institute of Technology

Theorem 1

If we can interrupt every path from source to sink in the SDG with the help of KeY, then non-interference holds for the complete program.

- Joana can guarantee non-interference
- Reason for false positives:
  - Approximation: addition of unnecessary edges
- Our approach deletes some of these additional edges
- KeY's non-interference property guarantees that we can delete these edges

After our approach run successfully Joana guarantees that non-interference holds

Demonstration

```
public int testMethodActive(int high, int low) {
   int i = identity(low, high);
   int j = neverIfTrue(low, high);
   int k = secure(low);
   return i + j + k;
}
public int identity(int low, int high) {
   low = low + high;
   low = low - high;
   return llow;
}
public int neverIfTrue(int low, int high) {
   int x = 0;
   if (x > 0) {
      low = high;
   }
   return low;
}
```

Demonstration

1. The approach generates the corresponding .jar file
2. Joana is executed with the .jar file as input
3. The generated SDG is annotated

p1: HIGH **p2: LOW**

```
public int testMethodActive(int high, int low) {
    int i = identity(low, high);
    int j = neverIfTrue(low, high);
    int k = secure(low);
    return i + j + k;   exit: LOW
}
```

4. Information Flow Analysis is performed
5. Heuristic choses a summary edge to verify with KeY

Demonstration

Karlsruhe Institute of Technology

6. The approach generates .java and .key file

```
public class testFile2{
    /*@ requires true;
      @ determines \result \by this, l; */
    public int identity(int l, int h) {
        l = l + h;
        l = l - h;
        return l;
    }
```

```
\profile "Java Profile";
\javaSource "proofs";
\proofObligation "#Proof Obligation Settings
name = proofs.testFile2[proofs.testFile2\\:\\:identity(int,int)].Non-interference contract.0
contract = proofs.testFile2[proofs.testFile2\\:\\:identity(int,int)].Non-interference contract.0
class=de.uka.ilkd.key.proof.init.InfFlowContractPO
";
```

Demonstration

7. KeY proves non-interference and returns *proven*

8. The same procedure is executed for the method *neverIfTrue(int low, int high)*

9. The approach returns that there is no information flow in the program

Conclusion

- The Combined Approach runs automatic and guarantees non-interference
- The number of calls of KeY depends strongly on the heuristic that choses the order of summary edges
- In the worst case the main method has to be proven with KeY

Future Work

- Decreasing the sufficient set of methods
- Optimization of the approach to minimize time- and user-effort:
  - Creation of information flow based loop-invariants
  - Extraction of context information from Joana to KeY
- Evaluation of the approach

Quellen

- [KlUl15] V. Klebanov, M. Ulbrich. *Applications of Formal Verification - Verification of Information Flow Properties,* KIT – Institut für Theoretische Informatik, Vorlesungsfolien, Sommersemester 2015.
- [SaMy03] A. Sabelfeld, A. C. Myers. *Language-Based Information-Flow Security*, IEEE Journal on selected aread in communications, vol. 21, no. 1, Januar 2003
- [Sch15] P. H. Schmitt. *Formale Systeme,* KIT – Institut für Theoretische Informatik, Vorlesungsskript Winter 2013/2014, Version: 30. April 2015.
- [SchSch12] M. Demleitner. *Verification of Information Flow Properties of Java Programs without Approximations*, Karlsruher Institute of Technology (KIT), Springer Verlag, 2012.
- [Giff12] D. Giffhorn, *Slicing of Concurrent Programs and its Application to Information Flow Control* Karlsruher Institute of Technology (KIT), 2012.
- [Joa16] http://pp.ipd.kit.edu/projects/joana/, accessed: 25.07.2016