An overview of control flow graph flattening

Jan Cappaert, Bart Preneel
K.U.Leuven / ESAT / SCD-COSIC

Overview

- Introduction and related research
- CFG flattening
- Experiments and ideas
- Conclusions and future work

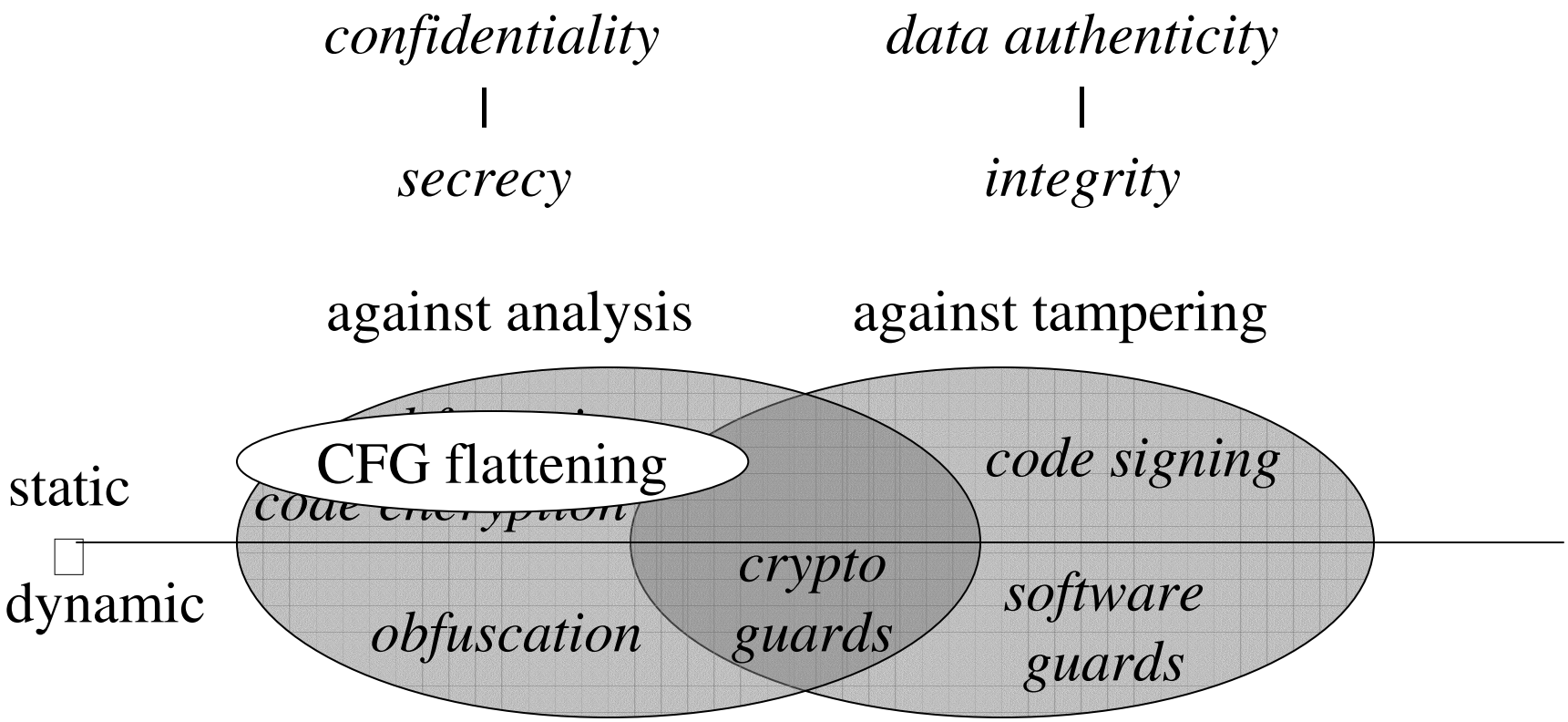
Introduction

- Software protection against
  - Analysis
  - Tampering
  - Plagiarism
- In a “white-box attack context”
  - Attacker has full privileges to the system
  - System behaves as a white box (vs. black box)

Introduction

- Software analysis
  - Static
    - No code execution
    - E.g.: disassembling, decompiling, …
  - Dynamic
    - Code executed
    - E.g.: debugging, tracing, emulation, …

Introduction

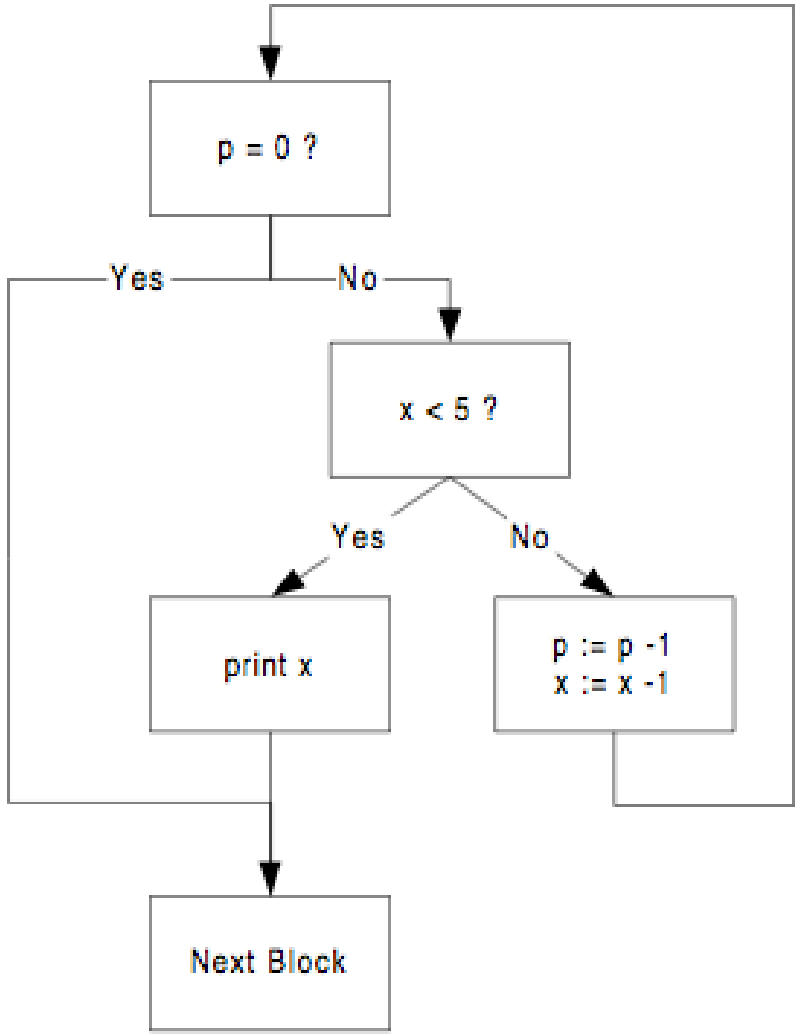


Introduction

- Control flow graph (CFG)
  - Nodes: basic blocks
  - Edges: control transfers
- Basic blocks
  - Group of statements always executed sequentially
- Control transfers
  - Transfer control from one block to another

Introduction

```
while (p) {
        if (x < 5) {
            print (x);
            break;
        }
        else {
            p = p - 1;
            x = x - 1;
        }
}
```

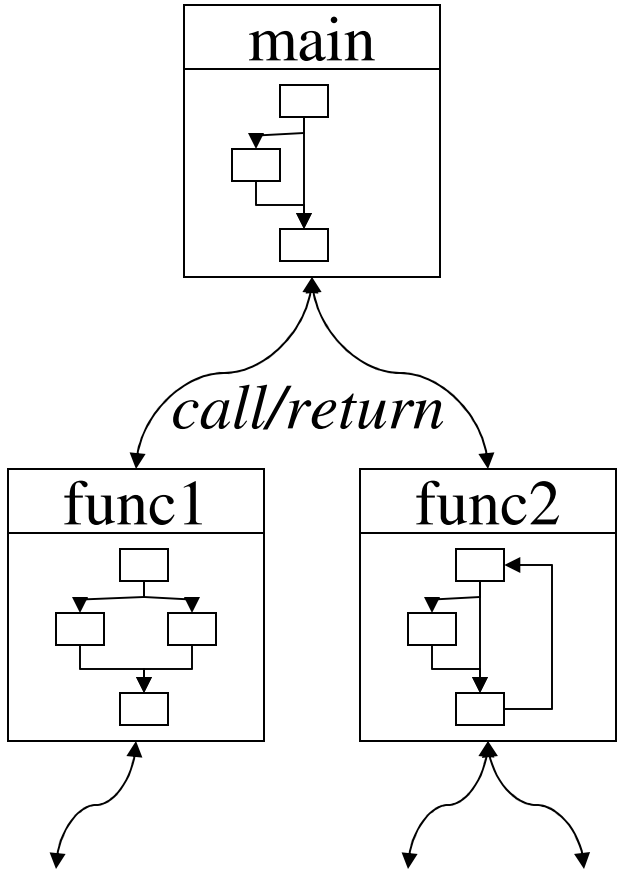


Introduction

- Why performing CFG analysis?
  - Data usage depending on control flow
  - Static analysis:
    - Flow-insensitive: incomplete, on 1 basic block
    - Flow-sensitive: more complete, over CFG

Related research

- Intra-procedural
  - CFG flattening
- Inter-procedural
  - Function pointers
  - Branch functions
    [Linn and Debray, 2003]



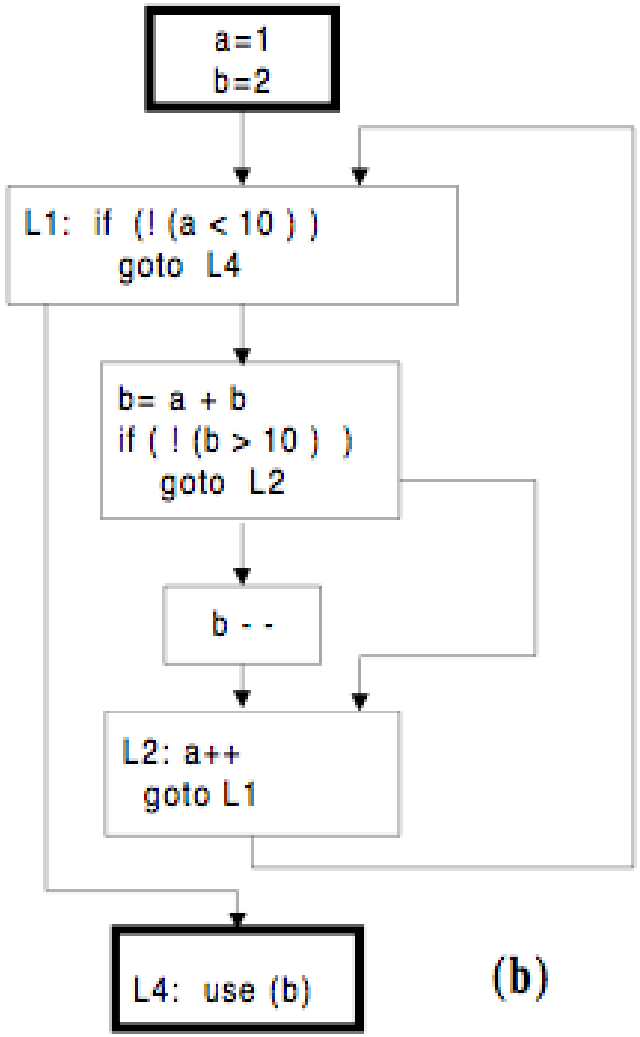
CFG transformations

- CFG flattening [Wang, 2000]
  - "degeneration of static program control flow"
- Control flow transformations [Collberg et al., 1997]
  - Opaque predicates
  - Loop/branch transformations

A control flow graph

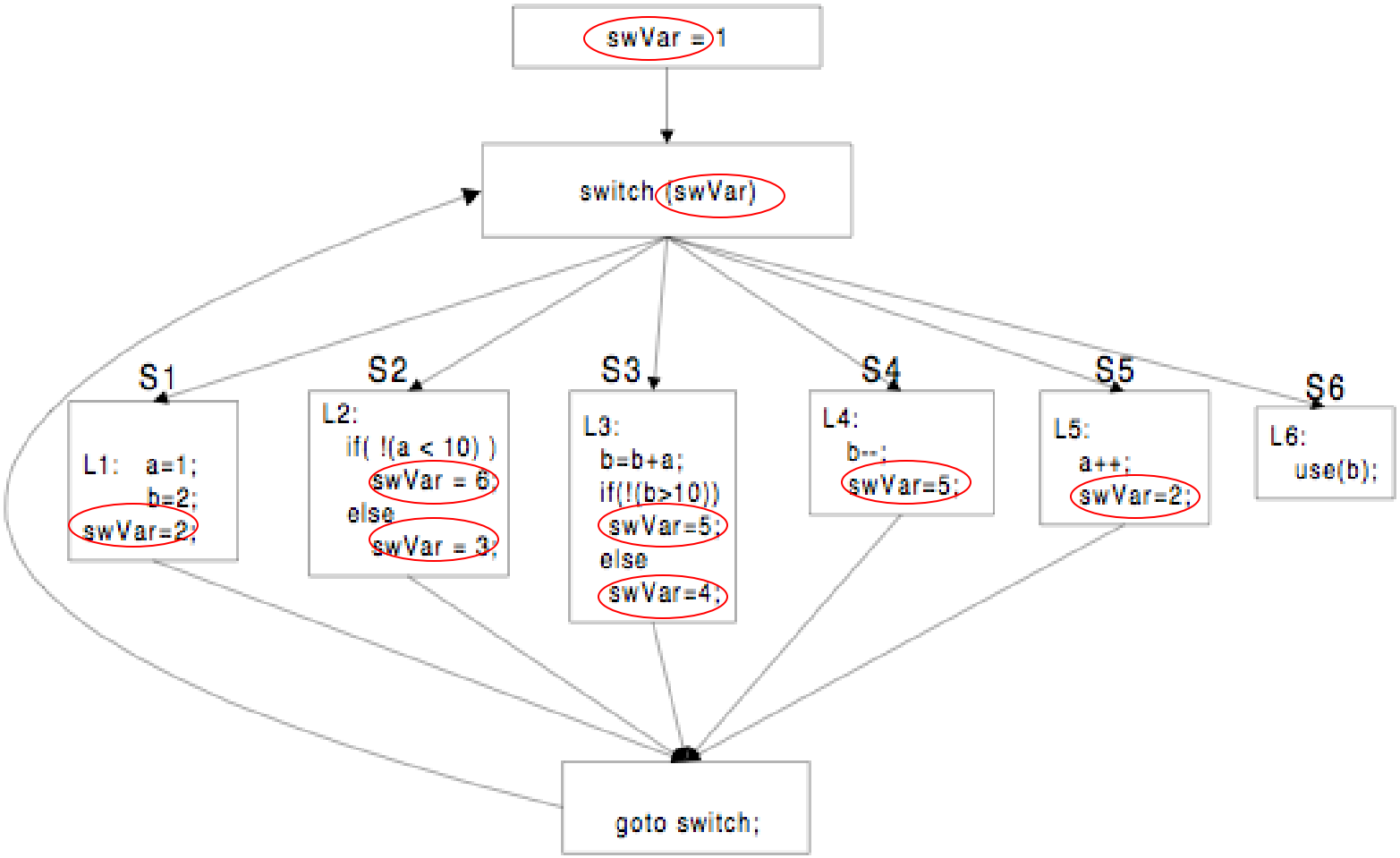
```
int a, b;
a=1;
b=2;
while(a<10){
     b=a+b;
     if(b>10)
        b--;
     a++;
}
use(b);
```

(a)



(b)

A control flow graph - flattened



CFG flattening - steps

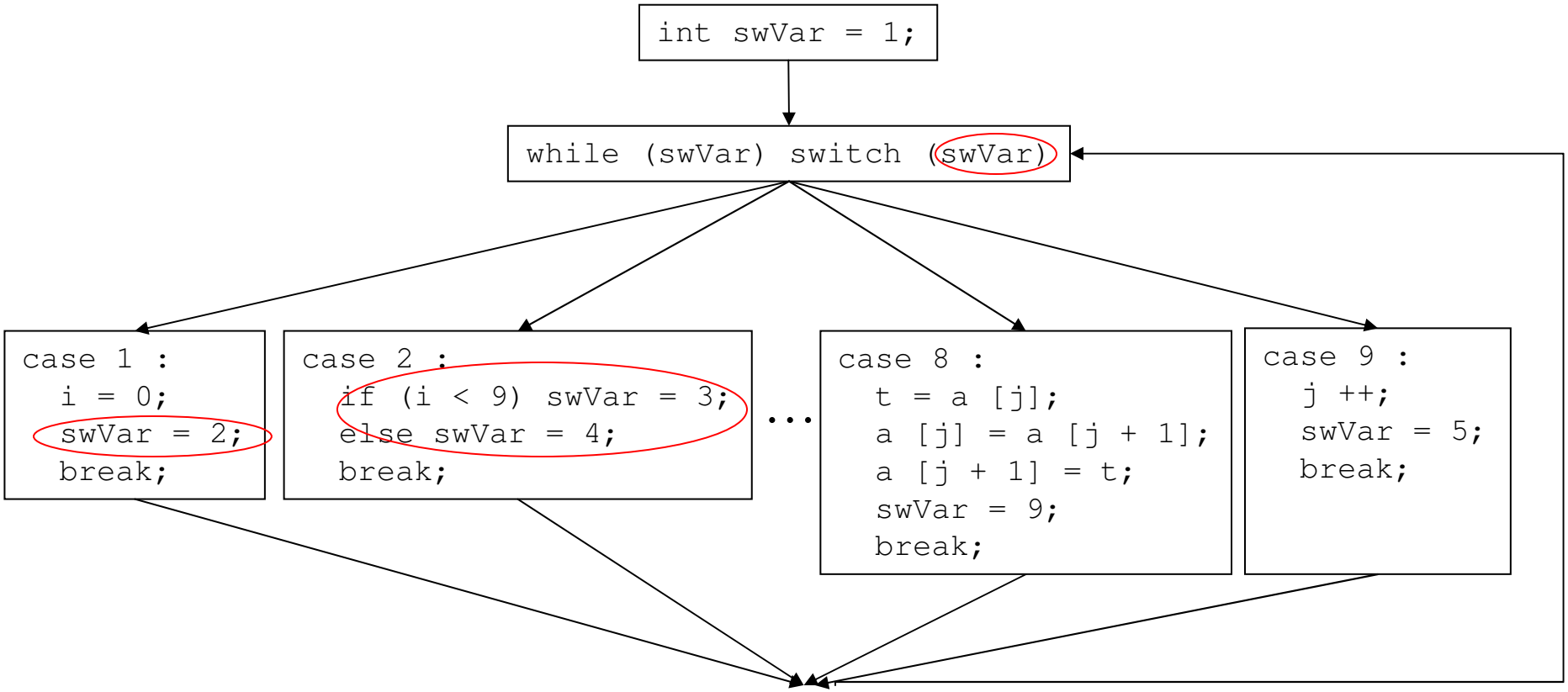
[Wang, 2001]

1. High-level constructs $\rightarrow$ `if`-then-`goto`
2. `goto` targets $\rightarrow$ dynamically determined
   $\Rightarrow$ common flattened form
3. Further hindrance of data flow analysis
   - Index computation (hard)
   - Aliasing (NP-complete …)

Experiments and ideas

```
for (i = 0; i < 9; i ++) {
    for (j = 0; j < 9 - i; j ++) {
        if (a [j] > a [j + 1]) {
            t = a [j];
            a [j] = a [j + 1];
            a [j + 1] = t;
        }
    }
}
```

Experiments and ideas



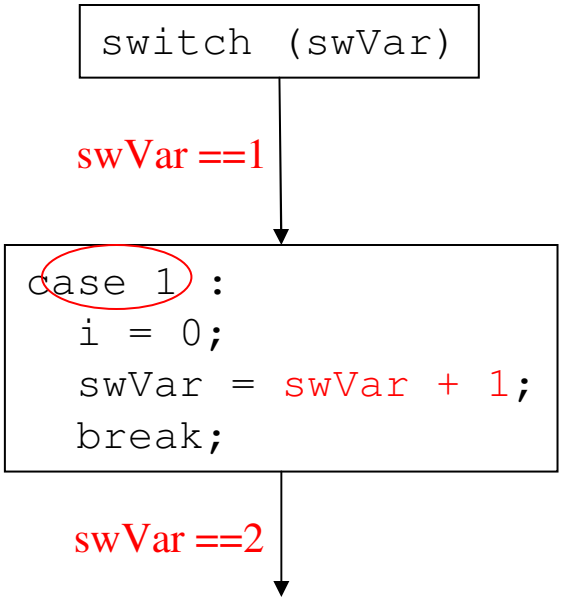
A flattened CFG - attacks

- Use-def analysis: 1↔2↔[3,4]↔…
- Forward? Backward? What if

```
swVar = swVar + constant;
swVar = swVar + condition * constant
```

- Constant propagation: 1→2→[3,4]→…
- Backward?

A flattened CFG - attacks



- Solution: one-way function
  e.g.: $x \rightarrow g^x \bmod p$

  `switch(ow(swVar))` or

  `swVar = ow(swVar)` …
- What if g changes at runtime? …

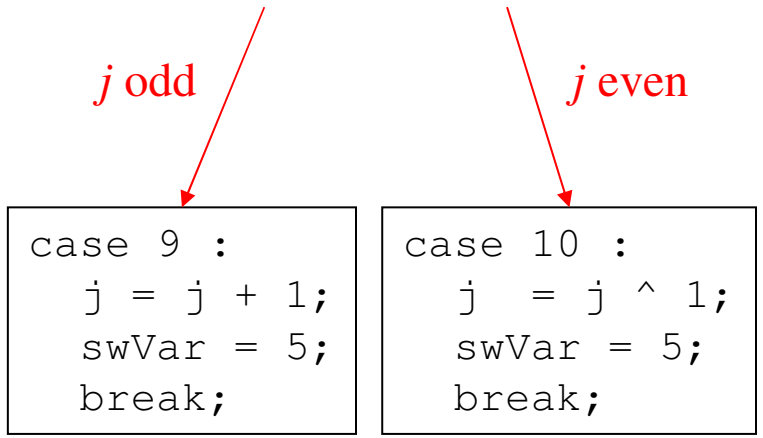
Additional ideas

- Relative updates of `swVar`
  - Conditions versus opaque predicates
- One-way functions, lookup tables, hash chains, …

- Aliasing + pointer permutation blocks
- Equivalent / almost equivalent blocks
  - Random / targeted conditions
- Block refactoring (splitting, merging, …)

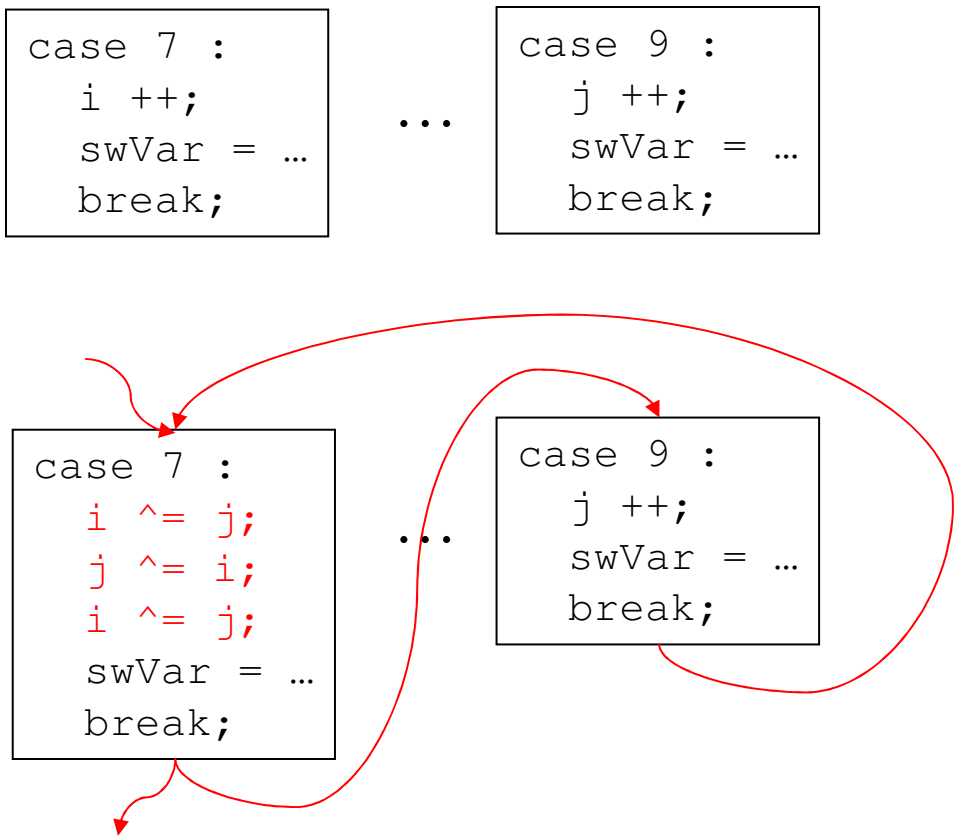
Additional ideas

- Almost equivalent block
  - Under certain conditions



Additional ideas

- Block refactoring
  - Swap pointers
  - Swap data



Conclusions

- Static CFG flattening
  - Common form; no explicit control flow
  - Control flow analysis requires data flow analysis
  - Data flow analysis can be hard (NP-complete) under certain conditions (e.g. general pointers)

Further work

- Formalization of ideas
  - One way function versus backward analysis
  - Hash chains and related
  - Basic block refactoring
- Implementation and performance overhead
- Interested?

talk to me

jan.cappaert@esat.kuleuven.be