No More Gotos: Decompilation Using Pattern-Independent Control-Flow Structuring and Semantics-Preserving Transformations

Khaled Yakdan[1] Sebastian Eschweiler[2] Elmar Gerhards-Padilla[2]
Matthew Smith[1]

[1]University of Bonn, Germany

[2]Fraunhofer FKIE, Germany

NDSS 2015

Agenda

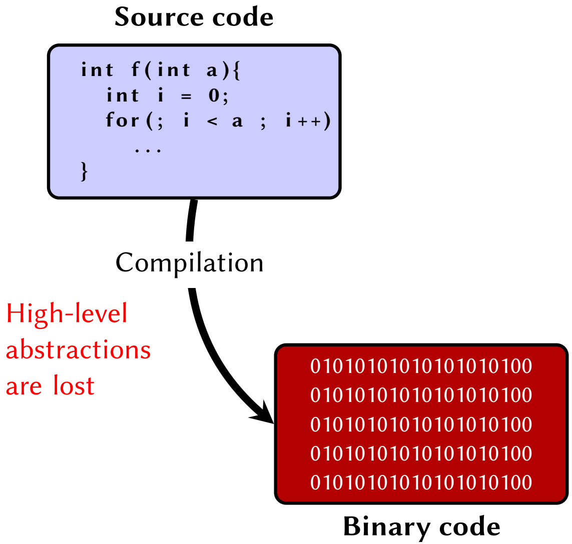
Motivation

Decompilation in Security

Binary code

Motivation

Decompilation in Security

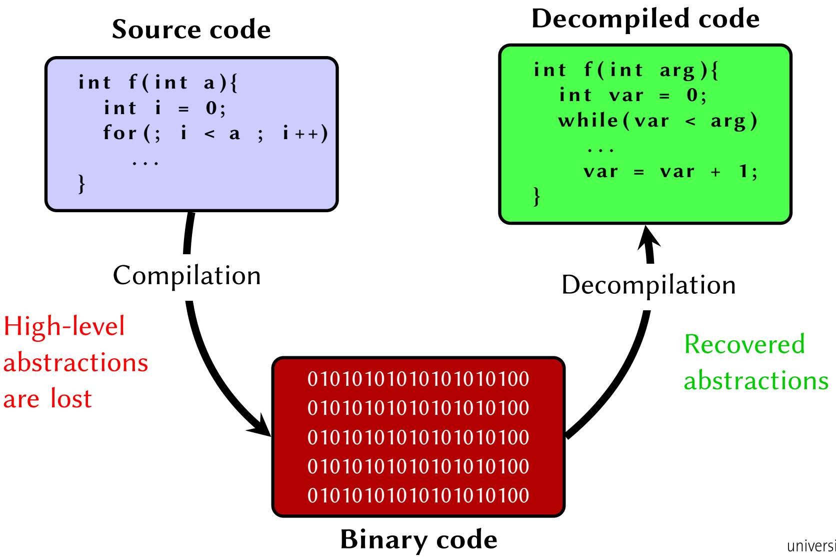


```
int f(int a){
    int i = 0;
    for(; i < a ; i++)
        ...
}
```

```
010101010101010100
010101010101010100
010101010101010100
010101010101010100
010101010101010100
```

Motivation

Decompilation in Security



```
int f(int a){
  int i = 0;
  for(; i < a ; i++)
     ...
}
```

```
int f(int arg){
  int var = 0;
  while(var < arg)
     ...
    var = var + 1;
}
```

```
01010101010101010100
01010101010101010100
01010101010101010100
01010101010101010100
01010101010101010100
```

Motivation
Decompilation in Security

- Manual reverse engineering

Motivation
Decompilation in Security

- Manual reverse engineering
- Apply source-based techniques to binary code

Motivation

Decompilation in Security

- Manual reverse engineering
- Apply source-based techniques to binary code
    - Find vulnerabilities, bugs

Motivation

Decompilation in Security

- Manual reverse engineering
- Apply source-based techniques to binary code
    - Find vulnerabilities, bugs
    - Taint tracking

Motivation
Decompilation in Security

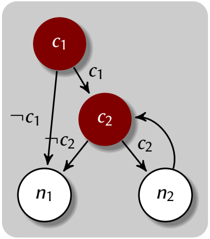
- Manual reverse engineering
- Apply source-based techniques to binary code
    - Find vulnerabilities, bugs
    - Taint tracking

Goal: Enhanced Structuredness

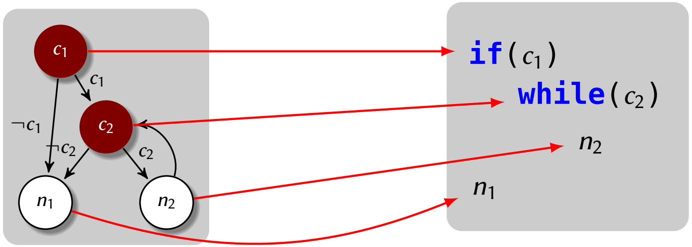
Effective control flow structure recovery to improve readability and enhance program analysis

Agenda

Control Flow Structuring



Control Flow Structuring



Structured vs. unstructured code

```
int f(int a, int b){
  int sum = 0;
  if(a < b){
    for(int i = a; i < b; i++)
      sum += i;
  }
  return sum;
}
```

Structured code

```
int f(int a, int b){
  int sum = 0;
  if(a >= b)
    goto Label_2;
  int i = a;
Label_1:
  if(i >= b)
    goto Label_2;
  sum += i;
  i++;
  goto Label_1;
Label_2:
  return sum;
}
```

Unstructured code

Control Flow Structuring
Structural Analysis

State of the art: **Structural Analysis** [Sharir80]

- Pattern-matching using a predefined set of region schemas (patterns)

Control Flow Structuring

Structural Analysis

State of the art: **Structural Analysis** [Sharir80]

- Pattern-matching using a predefined set of region schemas (patterns)
- Use goto statements if no match is found

Control Flow Structuring
Structural Analysis

State of the art: **Structural Analysis** [Sharir80]

- Pattern-matching using a predefined set of region schemas (patterns)
- Use goto statements if no match is found
- Example: Decompiling a P2P Zeus sample with Hex-Rays
  - 1,571 goto for 49,514 LoC
  - **1 goto for each 32 LoC**

Prior Work on Control-Flow Structuring

Improving vanilla structural analysis to recover more structure

- SESS Analysis [Engel et al., SCOPES 2011]
- Phoenix Decompiler [Schwartz et al., USENIX Security 2013]

Prior Work on Control-Flow Structuring

Improving vanilla structural analysis to recover more structure

- SESS Analysis [Engel et al., SCOPES 2011]
- Phoenix Decompiler [Schwartz et al., USENIX Security 2013]

New control-flow structuring algorithm

Pattern-Independent Structuring

Semantics-Preserving Transformations

Agenda

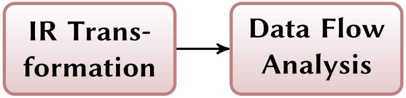
Dream

Decompiler for Reverse Engineering and Analysis of Malware

IR Trans-formation

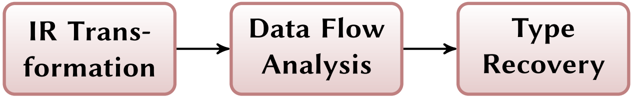
DREAM

Decompiler for Reverse Engineering and Analysis of Malware



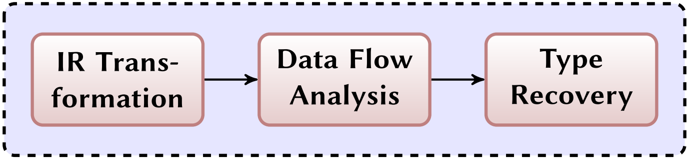
Dream

Decompiler for Reverse Engineering and Analysis of Malware



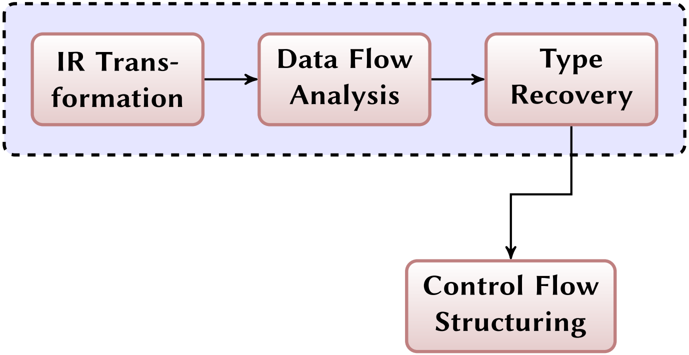
Dream

Decompiler for Reverse Engineering and Analysis of Malware



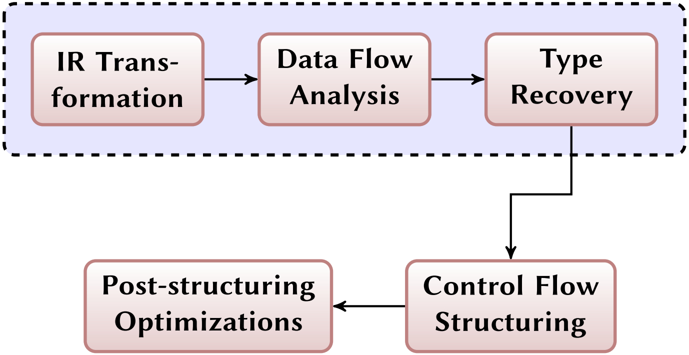
Dream

Decompiler for Reverse Engineering and Analysis of Malware



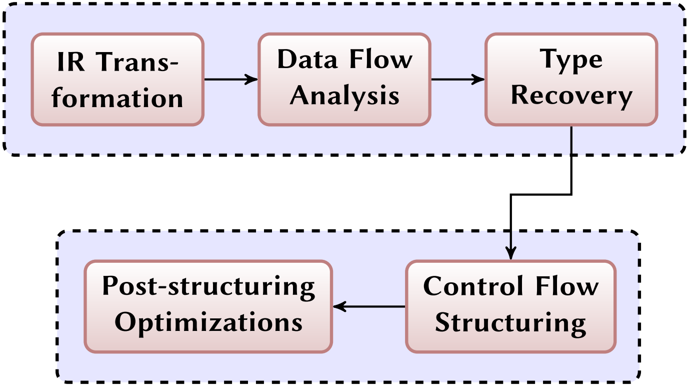
DREAM

Decompiler for Reverse Engineering and Analysis of Malware



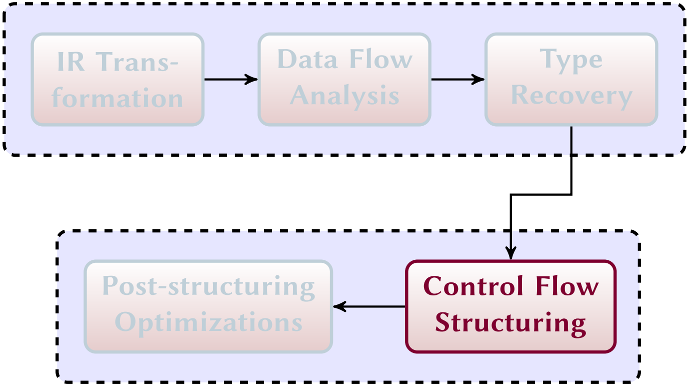
DREAM

Decompiler for Reverse Engineering and Analysis of Malware

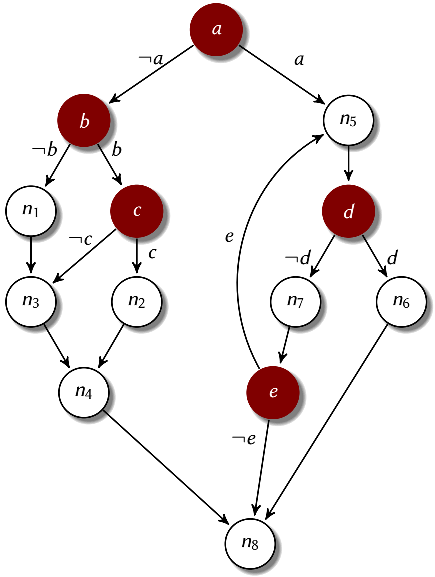


Dream

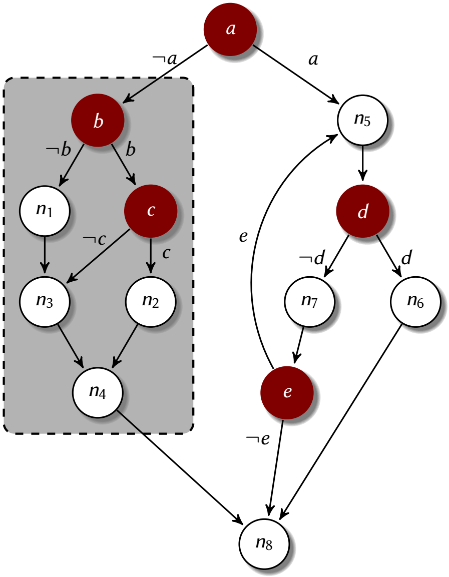
Decompiler for Reverse Engineering and Analysis of Malware



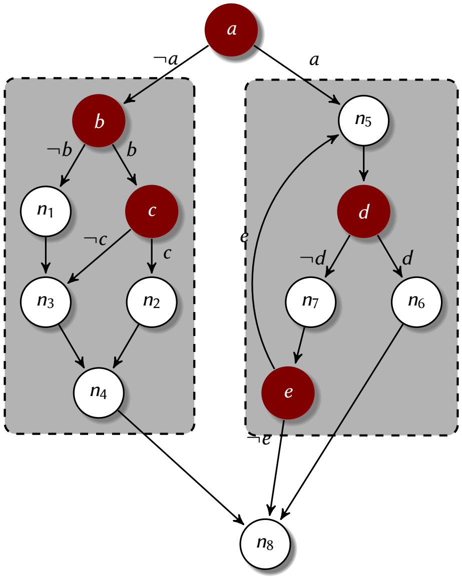
Running Example



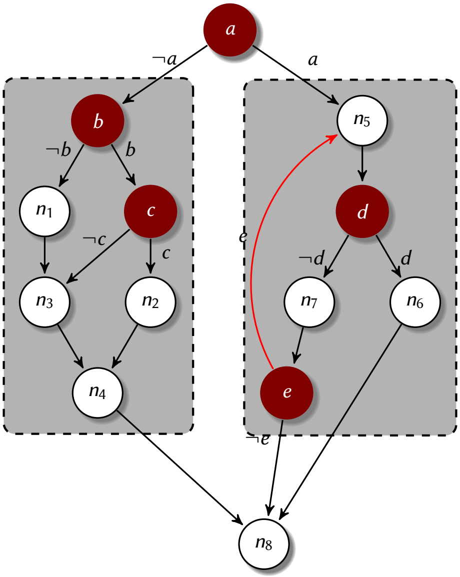
Running Example



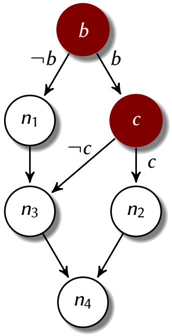
Running Example



Running Example

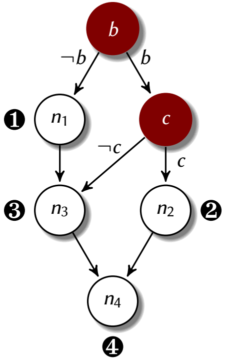


Acyclic Regions



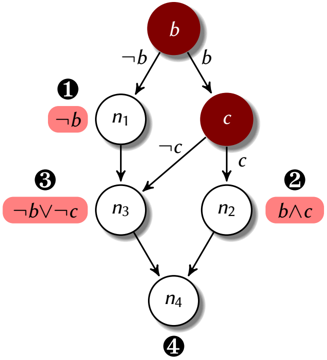
Acyclic Regions

- Lexical order



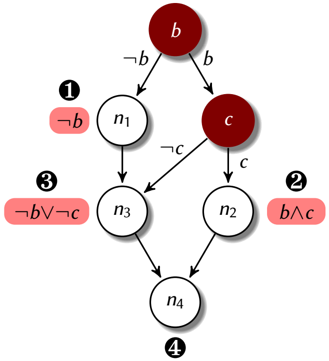
Acyclic Regions

- Lexical order
- Reaching conditions



Acyclic Regions

- Lexical order
- Reaching conditions
- Initial AST as a sequence of **if** constructs



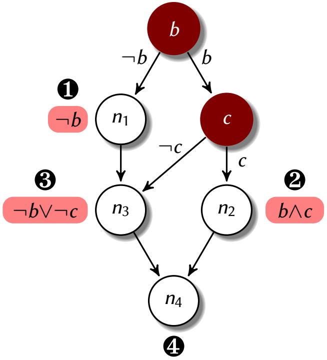
```
if (¬b)
   n1
if (b ∧ c)
   n2
if (¬b ∨ ¬c)
   n3
n4
```

Acyclic Regions

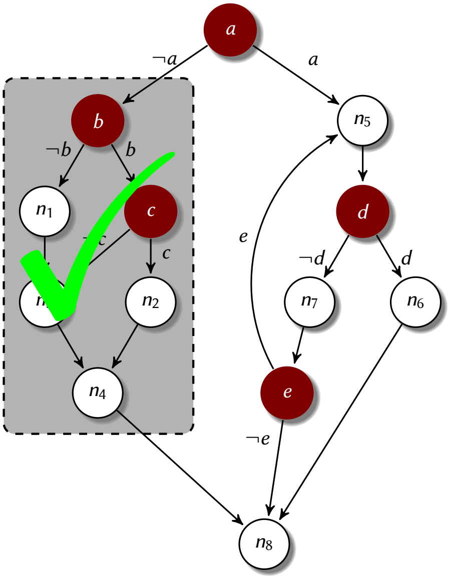
- Lexical order
- Reaching conditions
- Initial AST as a sequence of **if** constructs
- Refine initial AST to find **switch**, **if-else** constructs

```
if (¬b)
   n1
if (b ∧ c)
   n2
if (¬b ∨ ¬c)
   n3
n4
```

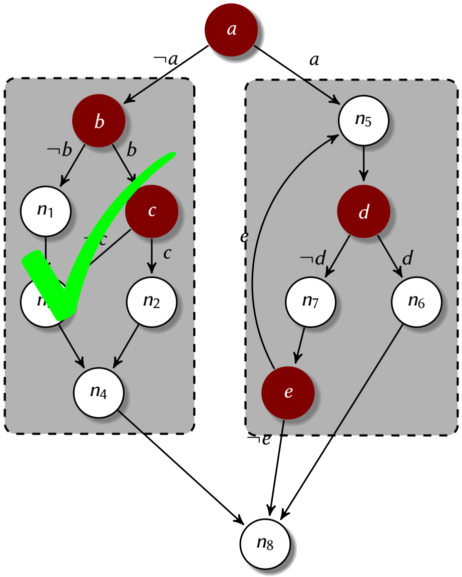
```
if (¬b)
   n1
if (b ∧ c)
   n2
else
   n3
n4
```



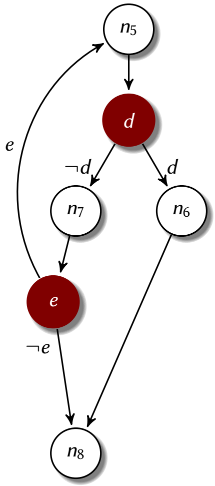
Running Example



Running Example

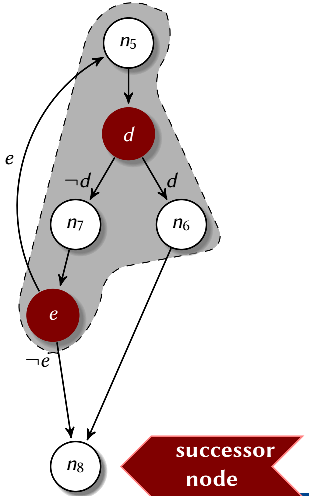


Cyclic Regions



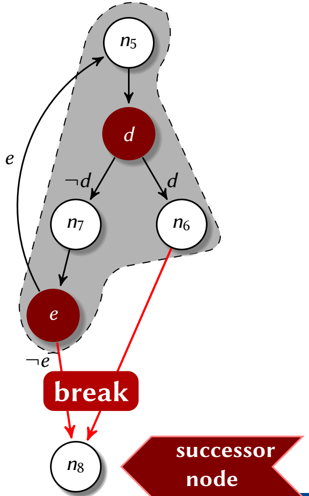
Cyclic Regions

- Identify loop nodes and successor node



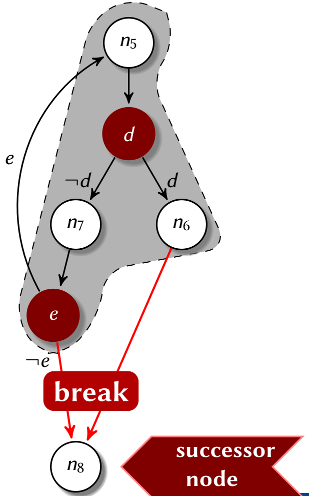
Cyclic Regions

- Identify loop nodes and successor node
- Replace edges to the successor node by **break** statements



Cyclic Regions

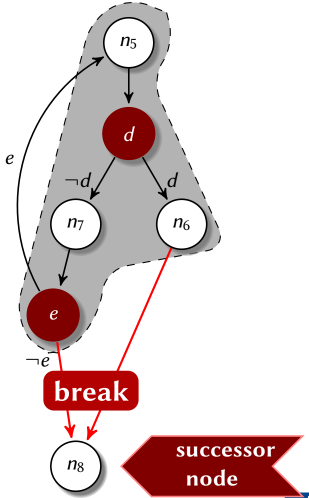
- Identify loop nodes and successor node
- Replace edges to the successor node by **break** statements
- Structure loop body B_{AST}



Cyclic Regions

- Identify loop nodes and successor node
- Replace edges to the successor node by **break** statements
- Structure loop body B_{AST}
- Initial AST: **while** (1) $\{B_{AST}\}$

```
while (1)
   ...
   if (¬e)
      break
```

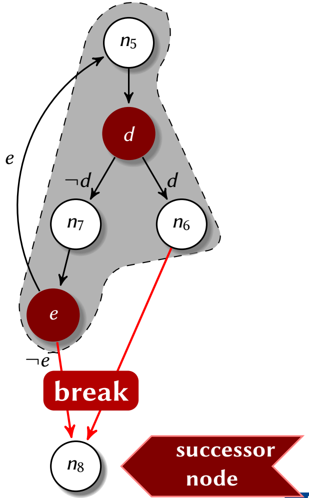


Cyclic Regions

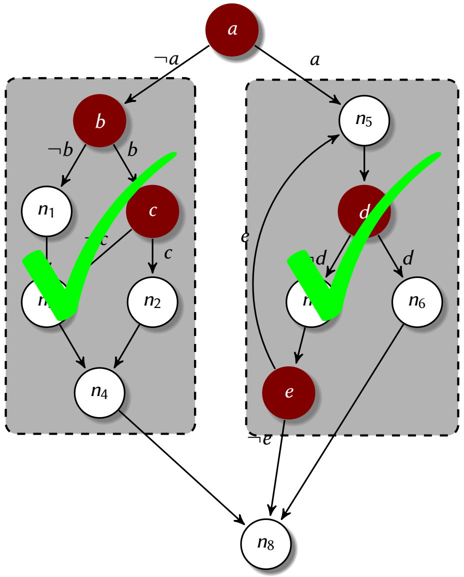
- Identify loop nodes and successor node
- Replace edges to the successor node by **break** statements
- Structure loop body B_{AST}
- Initial AST: **while** (1) $\{B_{AST}\}$
- Infer loop type and condition

```
while (1)
   ...
   if (¬e)
      break
```

```
do
   ...
while (e)
```



Running Example



Agenda

Readability Enhancements

```
int f(int a1){
  int v2 = 0;
  while((a1 <= 1 && v2 <= 100)
        || (a1 > 1 && v2 <= 10)){
     printf("inside_loop");
     ++v2;
     --a1;
  }
  printf("loop_terminated");
  return v2;
}
```

DREAM

```
signed int f(signed int a1){
  signed int v2;
  v2 = 0;
  while ( a1 > 1 ){
    if ( v2 > 10 )
      goto LABEL_7;
LABEL_6:
    printf("inside_loop");
    ++v2;
    --a1;
  }
  if ( v2 <= 100 )
    goto LABEL_6;
LABEL_7:
  printf("loop_terminated");
  return v2;
}
```

Hex-Rays

Evaluation

Metrics

- Correctness
- Structuredness
- Compactness

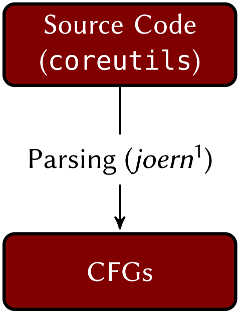
Correctness
Experiment Setup

Source Code
(`coreutils`)

[1] [Yamaguchi et al. IEEE S&P 2014]

Correctness

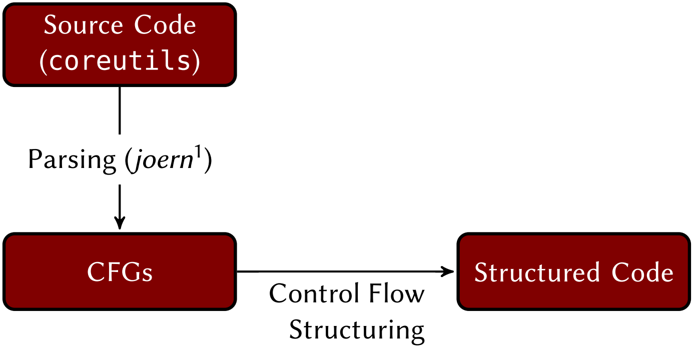
Experiment Setup



[1][Yamaguchi et al. IEEE S&P 2014]

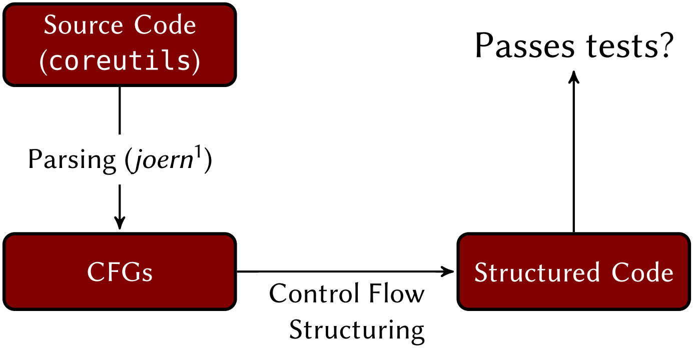
Correctness

Experiment Setup



[1] [Yamaguchi et al. IEEE S&P 2014]

Correctness
Experiment Setup



[1] [Yamaguchi et al. IEEE S&P 2014]

Correctness
Results

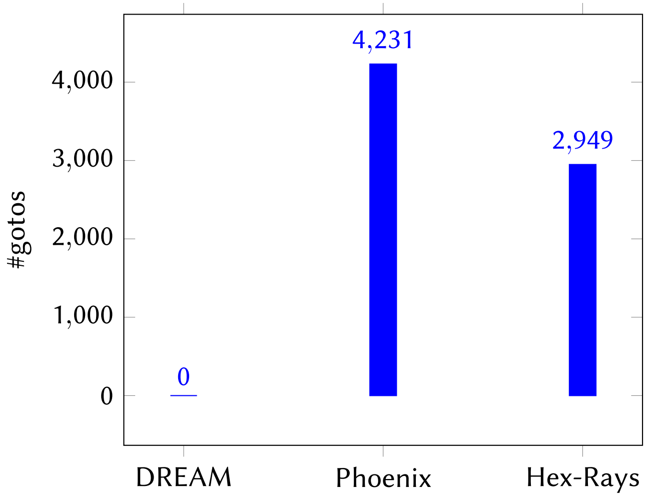
Considered Functions F	$\lvert F \rvert$	**Number of** gotos
Functions after preprocessor	1,738	219
Functions correctly parsed by *joern* [2]	1,530	129
Functions passed tests after structuring	1,530	0

[2] Errors have been reported to *joern*'s authors and are fixed in the current release

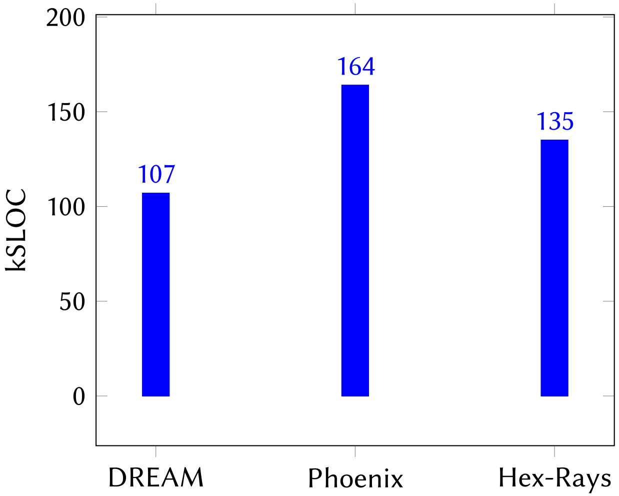
Structuredness and Compactness

- Tested decompilers
  - DREAM
  - Phoenix (academic state of the art)
  - Hex-Rays (industry state of the art)
- Structuredness
  - Number of gotos
- Compactness
  - Total lines of code
  - Compact functions

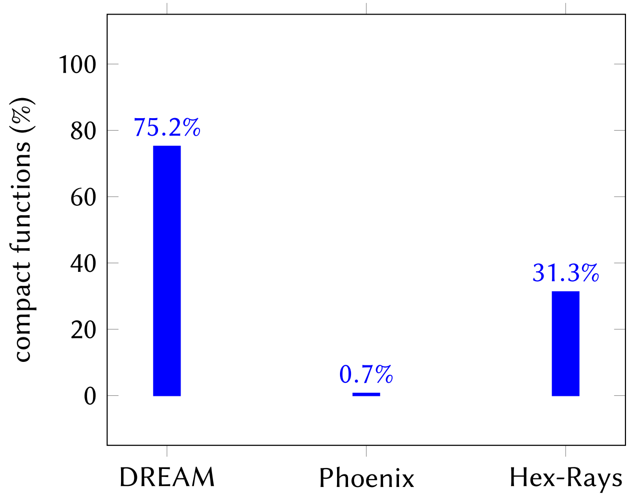
Structuredness



Compactness



Compactness



Agenda

Conclusion

- Novel control flow structuring algorithm
  - pattern-independent structuring
  - semantics-preserving transformations
- Dream decompiler
  - goto-free decompiled code
  - compact code
  - good readability

Thank You!

Questions?