THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Scalable, Sound, and Accurate Jump Table Analysis

Xiang Chen

COMP 5111 Presentation

2025-05-07

Contents

- Paper info: Huan Nguyen, Soumyakant Priyadarshan, and R. Sekar. 2024. Scalable, Sound, and Accurate Jump Table Analysis. In Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2024). Association for Computing Machinery, New York, NY, USA, 541–552. https://doi.org/10.1145/3650212.3680301
- Research group: http://seclab.cs.sunysb.edu/sekar/
- Area: Static binary analysis

1. Introduction

Disassembly

- The first step of all binary analysis tools: bug detection, retrofitting…

[1] SoK: All You Ever Wanted to Know About x86/x64 Binary Disassembly But Were Afraid to Ask, IEEE S&P '21

Disassembly

- The first step of all binary analysis tools: bug detection, retrofitting…
- Input: executable file and extracted `.text` section
- Output[1]: instruction boundary, control flow graph (jump table), function entry…

[1] SoK: All You Ever Wanted to Know About x86/x64 Binary Disassembly But Were Afraid to Ask, IEEE S&P '21

Disassembly

- The first step of all binary analysis tools: bug detection, retrofitting…
- Input: executable file and extracted `.text` section
- Output[1]: instruction boundary, control flow graph (jump table), function entry…

⚠️ Unlike source code analysis, each output is mutually dependent. The inaccuracy of one output has a cascading effect on the whole disassembly.

[1] SoK: All You Ever Wanted to Know About x86/x64 Binary Disassembly But Were Afraid to Ask, IEEE S&P '21

Disassembly

- The first step of all binary analysis tools: bug detection, retrofitting…
- Input: executable file and extracted `.text` section
- Output[1]: instruction boundary, control flow graph (jump table), function entry…

⚠️ Unlike source code analysis, each output is mutually dependent. The inaccuracy of one output has a cascading effect on the whole disassembly.

A two-fold example of jump table:

❌ Fail to identify all the jump targets will affect the soundness (coverage) of CFG

❌ Over-approximate the jump targets will add bogus edge in CFG

[1] SoK: All You Ever Wanted to Know About x86/x64 Binary Disassembly But Were Afraid to Ask, IEEE S&P '21

Jump Table Analysis - Definition

- Compiled from the `switch` statement in C/C++

```
switch(x) {
  case 0:
    return y+3;
  case 1:
    return 2*y;
  ...
  case 7:
    return y-4;
  default:
    return 0;
}
```

```
L1 :  mov $6000,%r8
      mov $8000,%r9
      mov %r11,(%r10)
      cmp (%r10),$7
      ja  L3
L2 :  mov %r11,%r12
      shl $2,%r12
      add %r9,%r12
      mov (%r12),%r13
      add %r8,%r13
      jmp *%r13
L3 :  ...
```

L1 : $R8 = 6000$
$R9 = 8000$
$*R10 = R11$
IF $(*R10 >_u 7)$
  JMP L3
L2 : $R12 = R11$
$R12 = R12 \ll 2$
$R12 = R9 + R12$
$R13 = *R12$
$R13 = R8 + R13$
JMP $*R13$
L3 : ...

```
; Jump table
0x8000  .L1
0x8004  .L2
...

; Jump Target
0x6000+.L1 .func1
0x6000+.L2 .func2
```

Figure 1: A C/C++ switch statement and its assembly code / IR

`R13 = R8 + *(R9 + R12 * 4)`

- A sound jump table analysis ensures that all possible targets of an indirect jump are identified $\rightarrow$ A complete CFG

Jump Table Analysis - Challenge

Four major types of jump table expression[1]:

- Basic: `*(TBase + Stride x Index)`
- PIC code: `Base + *(TBase + Stride x Index)` $\rightarrow$ Most Common (Figure 1)
- Fixed code size: `Base + Stride x Index` $\rightarrow$ Human written
- Nested (n=2): `∗(TBase2 + Stride2 × ∗(TBase1 + Stride1 × Index))`

Challenge:

Accurately recover the (multi-layer) jump table expression without patterns

[1] TBase - base address of jump table / Stride - The size of each jump table entry / Index - the offset in jump table

Jump Table Analysis - Related works

Most jump table analyses (1) compute a **backward** slice of the register used in the jump (2) recover the jump table and its size, stride (3) compute the expression and get the index range.

The four limitations in existing works:

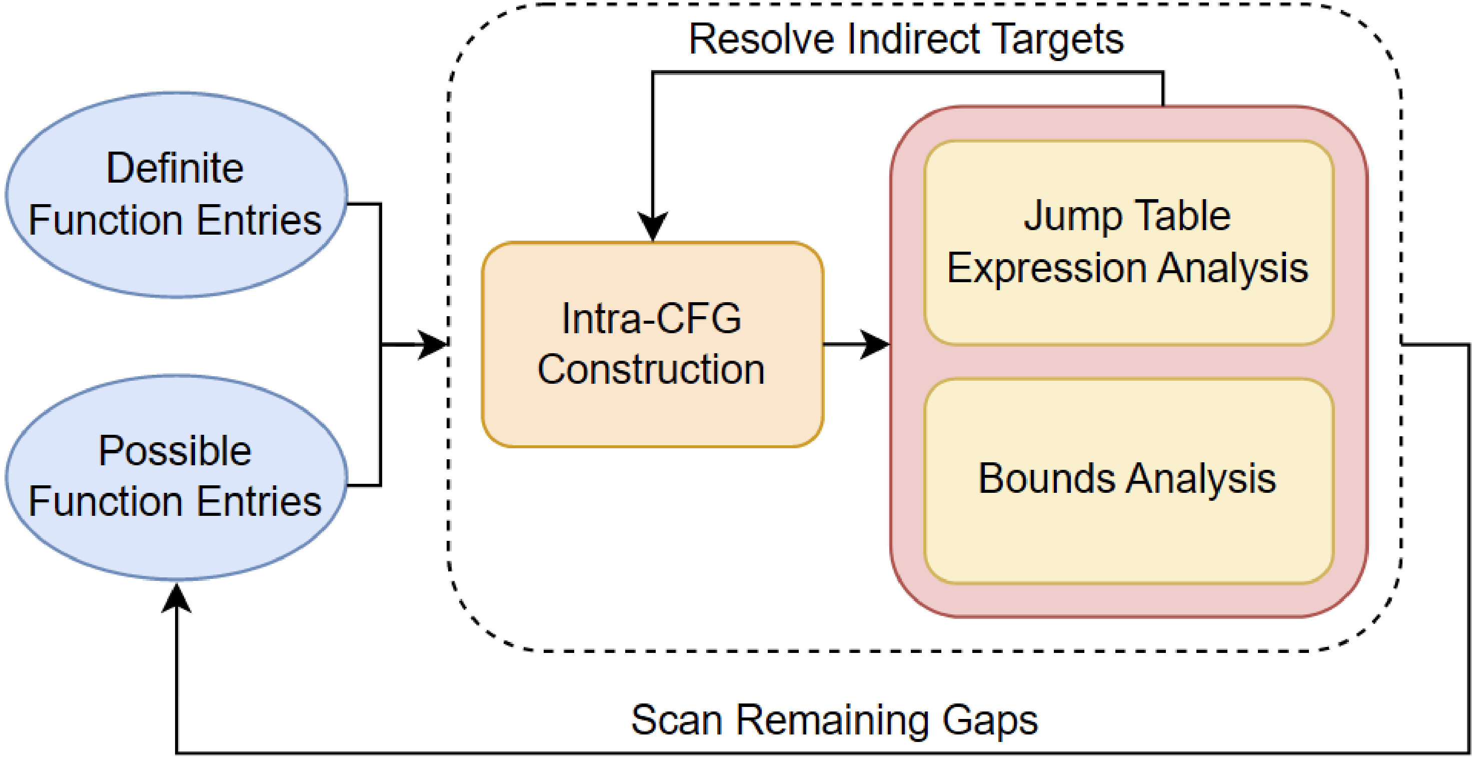
1. (Unsound) Limiting the length of backward slicing
2. (Unsound) Limiting the number of paths considered
3. (Not Scalable) Pattern-driven analysis based on the compiler
4. (Unsound) Limited capacity to analyze jump table bounds

💡 SJA leverage **forward analysis**, considers **all paths** to guarantee soundness.

🤔 SJA's sensitivity: path, field, flow-sensitive, context-insensitive

2. Workflow of SJA

Pre-analysis: function detection[1] → intra-procedural and recursive analysis



[1] A Principled Approach for Function Recognition in COTS Binaries. DSN '17

3. Key algorithm in SJA

Abstract Interpretation (AI) Review

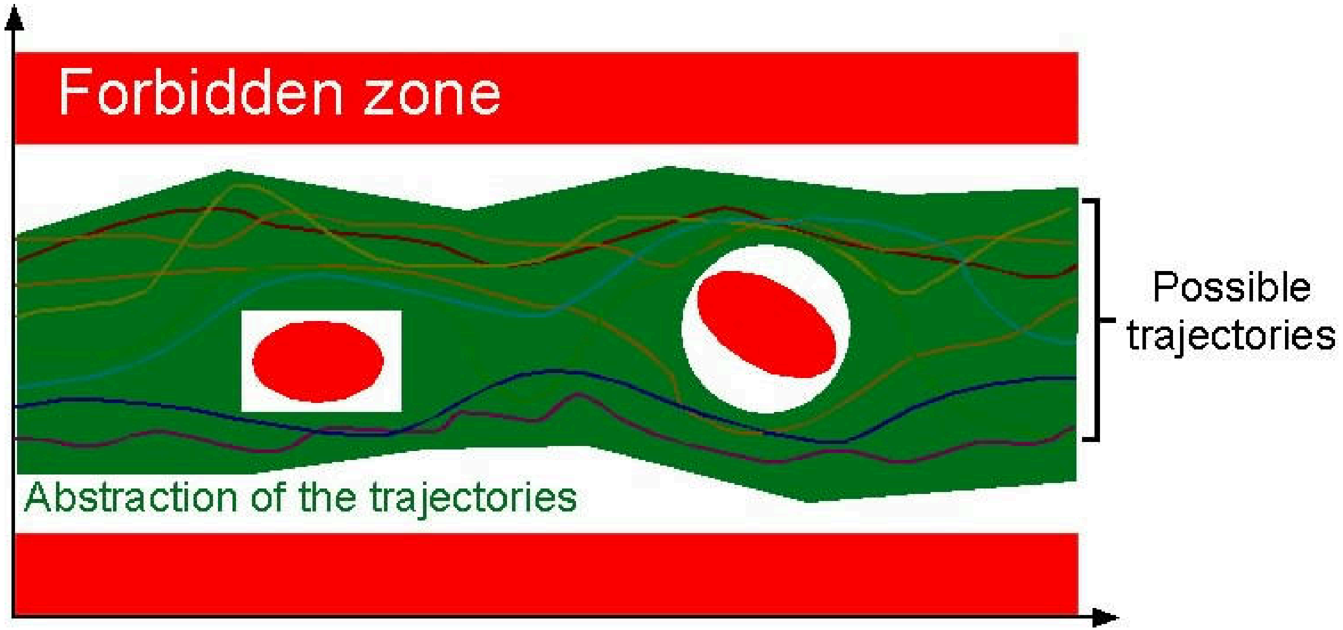


Figure 3: High-level idea of AI

x	y	$x +_a y$	$x *_a y$
$+ve$	$+ve$	$+ve$	$+ve$
$+ve$	$-ve$	$\top$	$-ve$
$-ve$	$+ve$	$\top$	$-ve$
$-ve$	$-ve$	$-ve$	$+ve$

Figure 4: A simple sign domain

Abstract Interpretation (AI) Review

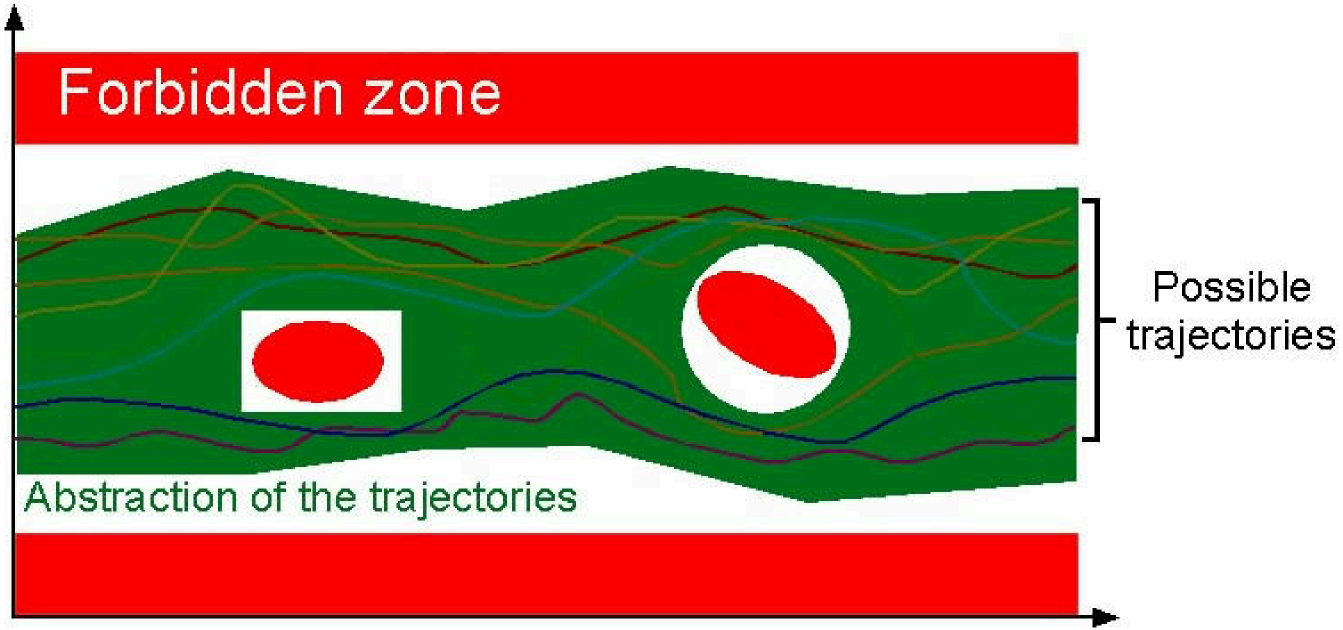


Figure 3: High-level idea of AI

x	y	$x +_a y$	$x *_a y$
$+ve$	$+ve$	$+ve$	$+ve$
$+ve$	$-ve$	$\top$	$-ve$
$-ve$	$+ve$	$\top$	$-ve$
$-ve$	$-ve$	$-ve$	$+ve$

Figure 4: A simple sign domain

SJA uses AI to model the formula of jump table expression as proposed in the previous four types. Specifically, it designs a new **domain** for the jump table patterns.

Jump Table Expression Analysis

Domain $\mathcal{D} ::= \mathbf{D} \cup \{\top, \bot\}$

$\mathbf{D} ::= B \mid B{+}S{\times}I$ where $B,S \in \mathbb{Z}$, $S \neq 0$, $I \in (\mathbf{V} \cup *\mathbf{D})$,

V is the set of variables in the program

Expressions $e ::= c \mid v \mid e{+}e \mid e{-}e \mid e{\times}e \mid e{\ll}c \mid *e$

(v is a variable, c is a constant)

Base cases

c	c
v	$0{+}1{\times}v$

Recursive cases

X	$B_1{+}S_1{\times}I$	$B_1{+}S_1{\times}I$
Y	$B_2{+}S_2{\times}I$	B_2
$X{+}Y$	$(B_1{+}B_2){+}(S_1{+}S_2){\times}I$	$(B_1{+}B_2){+}S_1{\times}I$
$X{-}Y$	$(B_1{-}B_2){+}(S_1{-}S_2){\times}I$	$(B_1{-}B_2){+}S_1{\times}I$
$X{\times}Y$	$\top$	$(B_1{\times}B_2){+}(S_1{\times}B_2){\times}I$
$X{\ll}Y$	$\top$	$(B_1{\times}2^{B_2}){+}(S_1{\times}2^{B_2}){\times}I$
$*Y$	$0{+}1{\times}*(B_2{+}S_2{\times}I)$	$0{+}1{\times}*B_2$

Figure 5: Abstract domain for jump table

L1 :	$R8 = 6000$	$R8 = 6000$	B−1
	$R9 = 8000$	$R9 = 8000$	B−1
	$*R10 = R11$	$*R10 = 0{+}1{\times}R11$	B−2
	IF $(*R10 >_u 7)$	N/A	
	JMP L3	N/A	
L2 :	$R12 = R11$	$R12 = 0{+}1{\times}R11$	B−2
	$R12 = R12 \ll 2$	$R12 = 0{+}4{\times}R11$	R−4.2
	$R12 = R9{+}R12$	$R12 = 8000{+}4{\times}R11$	R−1.2
	$R13 = *R12$	$R13 = 0{+}1{\times}*(8000{+}4{\times}R11)$	R−5.1
	$R13 = R8{+}R13$	$R13 = 6000{+}1{\times}*(8000{+}4{\times}R11)$	R−1.2
	JMP $*R13$	N/A	
L3 :	. . .	N/A	

Figure 6: Illustration of abstraction domain

Jump Table Expression Analysis

Domain $\mathcal{D} ::= \mathbf{D} \cup \{\top,\bot\}$

$\mathbf{D} ::= B \mid B+S\times I$ where $B,S\in\mathbb{Z}$, $S\neq 0$, $I\in(\mathbf{V}\cup *\mathbf{D})$,

V is the set of variables in the program

Expressions $e ::= c \mid v \mid e+e \mid e-e \mid e\times e \mid e \ll c \mid *e$

(v is a variable, c is a constant)

Base cases

c	c
v	$0+1\times v$

Recursive cases

X	$B_1+S_1\times I$	$B_1+S_1\times I$
Y	$B_2+S_2\times I$	B_2
$X+Y$	$(B_1+B_2)+(S_1+S_2)\times I$	$(B_1+B_2)+S_1\times I$
$X-Y$	$(B_1-B_2)+(S_1-S_2)\times I$	$(B_1-B_2)+S_1\times I$
$X\times Y$	$\top$	$(B_1\times B_2)+(S_1\times B_2)\times I$
$X\ll Y$	$\top$	$(B_1\times 2^{B_2})+(S_1\times 2^{B_2})\times I$
$*Y$	$0+1\times *(B_2+S_2\times I)$	$0+1\times *B_2$

Figure 5: Abstract domain for jump table

L1 :	$R8=6000$	$R8=6000$	B−1
	$R9=8000$	$R9=8000$	B−1
	$*R10=R11$	$*R10=0+1\times R11$	B−2
	IF $(*R10>_u 7)$	N/A	
	JMP L3	N/A	
L2 :	$R12=R11$	$R12=0+1\times R11$	B−2
	$R12=R12\ll 2$	$R12=0+4\times R11$	R−4.2
	$R12=R9+R12$	$R12=8000+4\times R11$	R−1.2
	$R13=*R12$	$R13=0+1\times *(8000+4\times R11)$	R−5.1
	$R13=R8+R13$	$R13=6000+1\times *(8000+4\times R11)$	R−1.2
	JMP $*R13$	N/A	
L3 :	...	N/A	

Figure 6: Illustration of abstraction domain

- Propagate through assignment

Jump Table Expression Analysis

Domain $\mathcal{D} ::= \mathbf{D} \cup \{\top, \bot\}$

$\mathbf{D} ::= B \mid B+S\times I$ where $B,S \in \mathbb{Z}$, $S \neq 0$, $I \in (\mathbf{V} \cup *\mathbf{D})$,

$\mathbf{V}$ is the set of variables in the program

Expressions $e ::= c \mid v \mid e+e \mid e-e \mid e\times e \mid e \ll c \mid *e$

(v is a variable, c is a constant)

Base cases

c	c
v	$0+1\times v$

Recursive cases

X	$B_1+S_1\times I$	$B_1+S_1\times I$
Y	$B_2+S_2\times I$	B_2
$X+Y$	$(B_1+B_2)+(S_1+S_2)\times I$	$(B_1+B_2)+S_1\times I$
$X-Y$	$(B_1-B_2)+(S_1-S_2)\times I$	$(B_1-B_2)+S_1\times I$
$X\times Y$	$\top$	$(B_1\times B_2)+(S_1\times B_2)\times I$
$X\ll Y$	$\top$	$(B_1\times 2^{B_2})+(S_1\times 2^{B_2})\times I$
$*Y$	$0+1\times *(B_2+S_2\times I)$	$0+1\times *B_2$

Figure 5: Abstract domain for jump table

L1 :	$R8=6000$	$R8=6000$	B−1
	$R9=8000$	$R9=8000$	B−1
	$*R10=R11$	$*R10=0+1\times R11$	B−2
	IF $(*R10>_u 7)$	N/A	
	JMP L3	N/A	
L2 :	$R12=R11$	$R12=0+1\times R11$	B−2
	$R12=R12\ll 2$	$R12=0+4\times R11$	R−4.2
	$R12=R9+R12$	$R12=8000+4\times R11$	R−1.2
	$R13=*R12$	$R13=0+1\times *(8000+4\times R11)$	R−5.1
	$R13=R8+R13$	$R13=6000+1\times *(8000+4\times R11)$	R−1.2
	JMP $*R13$	N/A	
L3 :	. . .	N/A	

Figure 6: Illustration of abstraction domain

- Propagate through assignment
- Apply union at CFG merge points

Jump Table Expression Analysis

Domain $\mathcal{D} ::= \mathbf{D} \cup \{\top, \bot\}$

$\mathbf{D} ::= B \mid B{+}S{\times}I$ where $B,S \in \mathbb{Z}$, $S \neq 0$, $I \in (\mathbf{V} \cup *\mathbf{D})$,

$\mathbf{V}$ is the set of variables in the program

Expressions $e ::= c \mid v \mid e{+}e \mid e{-}e \mid e{\times}e \mid e{\ll}c \mid *e$

(v is a variable, c is a constant)

Base cases

c	c
v	$0{+}1{\times}v$

Recursive cases

X	$B_1{+}S_1{\times}I$	$B_1{+}S_1{\times}I$
Y	$B_2{+}S_2{\times}I$	B_2
$X{+}Y$	$(B_1{+}B_2){+}(S_1{+}S_2){\times}I$	$(B_1{+}B_2){+}S_1{\times}I$
$X{-}Y$	$(B_1{-}B_2){+}(S_1{-}S_2){\times}I$	$(B_1{-}B_2){+}S_1{\times}I$
$X{\times}Y$	$\top$	$(B_1{\times}B_2){+}(S_1{\times}B_2){\times}I$
$X{\ll}Y$	$\top$	$(B_1{\times}2^{B_2}){+}(S_1{\times}2^{B_2}){\times}I$
$*Y$	$0{+}1{\times}*(B_2{+}S_2{\times}I)$	$0{+}1{\times}*B_2$

Figure 5: Abstract domain for jump table

L1 :	$R8{=}6000$	$R8{=}6000$	B−1
	$R9{=}8000$	$R9{=}8000$	B−1
	$*R10{=}R11$	$*R10{=}0{+}1{\times}R11$	B−2
	IF ($*R10{>}_u 7$)	N/A	
	JMP L3	N/A	
L2 :	$R12{=}R11$	$R12{=}0{+}1{\times}R11$	B−2
	$R12{=}R12 \ll 2$	$R12{=}0{+}4{\times}R11$	R−4.2
	$R12{=}R9{+}R12$	$R12{=}8000{+}4{\times}R11$	R−1.2
	$R13{=}*R12$	$R13{=}0{+}1{\times}*(8000{+}4{\times}R11)$	R−5.1
	$R13{=}R8{+}R13$	$R13{=}6000{+}1{\times}*(8000{+}4{\times}R11)$	R−1.2
	JMP $*R13$	N/A	
L3 :	. . .	N/A	

Figure 6: Illustration of abstraction domain

- Propagate through assignment
- Apply union at CFG merge points
- Check abstract value at indirect jumps

Jump Table Expression Analysis

Domain $\mathcal{D} ::= \mathbf{D} \cup \{\top, \bot\}$

$\mathbf{D} ::= B \mid B{+}S{\times}I$ where $B,S \in \mathbb{Z}$, $S \neq 0$, $I \in (\mathbf{V} \cup *\mathbf{D})$,

$\mathbf{V}$ is the set of variables in the program

Expressions $e ::= c \mid v \mid e{+}e \mid e{-}e \mid e{\times}e \mid e{\ll}c \mid *e$

(v is a variable, c is a constant)

Base cases

c	c
v	$0{+}1{\times}v$

Recursive cases

X	$B_1{+}S_1{\times}I$	$B_1{+}S_1{\times}I$
Y	$B_2{+}S_2{\times}I$	B_2
$X{+}Y$	$(B_1{+}B_2){+}(S_1{+}S_2){\times}I$	$(B_1{+}B_2){+}S_1{\times}I$
$X{-}Y$	$(B_1{-}B_2){+}(S_1{-}S_2){\times}I$	$(B_1{-}B_2){+}S_1{\times}I$
$X{\times}Y$	$\top$	$(B_1{\times}B_2){+}(S_1{\times}B_2){\times}I$
$X{\ll}Y$	$\top$	$(B_1{\times}2^{B_2}){+}(S_1{\times}2^{B_2}){\times}I$
$*Y$	$0{+}1{\times}*(B_2{+}S_2{\times}I)$	$0{+}1{\times}*B_2$

Figure 5: Abstract domain for jump table

L1 :	$R8=6000$	$R8=6000$	B−1
	$R9=8000$	$R9=8000$	B−1
	$*R10=R11$	$*R10=0+1\times R11$	B−2
	IF $(*R10>_u 7)$	N/A	
	JMP L3	N/A	
L2 :	$R12=R11$	$R12=0+1\times R11$	B−2
	$R12=R12\ll 2$	$R12=0+4\times R11$	R−4.2
	$R12=R9+R12$	$R12=8000+4\times R11$	R−1.2
	$R13=*R12$	$R13=0+1\times *(8000+4\times R11)$	R−5.1
	$R13=R8+R13$	$R13=6000+1\times *(8000+4\times R11)$	R−1.2
	JMP $*R13$	N/A	
L3 :	...	N/A	

Figure 6: Illustration of abstraction domain

- Propagate through assignment
- Apply union at CFG merge points
- Check abstract value at indirect jumps
- Retrieve jump table entries iteratively

Jump Table Expression Analysis

Domain $\mathcal{D} ::= \mathbf{D} \cup \{\top, \bot\}$

$\mathbf{D} ::= B \mid B{+}S{\times}I$ where $B,S \in \mathbb{Z}$, $S \neq 0$, $I \in (\mathbf{V} \cup *\mathbf{D})$,

$\mathbf{V}$ is the set of variables in the program

Expressions $e ::= c \mid v \mid e{+}e \mid e{-}e \mid e{\times}e \mid e{\ll}c \mid *e$

(v is a variable, c is a constant)

Base cases

c	c
v	$0{+}1{\times}v$

Recursive cases

X	$B_1{+}S_1{\times}I$	$B_1{+}S_1{\times}I$
Y	$B_2{+}S_2{\times}I$	B_2
$X{+}Y$	$(B_1{+}B_2){+}(S_1{+}S_2){\times}I$	$(B_1{+}B_2){+}S_1{\times}I$
$X{-}Y$	$(B_1{-}B_2){+}(S_1{-}S_2){\times}I$	$(B_1{-}B_2){+}S_1{\times}I$
$X{\times}Y$	$\top$	$(B_1{\times}B_2){+}(S_1{\times}B_2){\times}I$
$X{\ll}Y$	$\top$	$(B_1{\times}2^{B_2}){+}(S_1{\times}2^{B_2}){\times}I$
$*Y$	$0{+}1{\times}*(B_2{+}S_2{\times}I)$	$0{+}1{\times}*B_2$

Figure 5: Abstract domain for jump table

L1 :	$R8=6000$	$R8=6000$	B−1
	$R9=8000$	$R9=8000$	B−1
	$*R10=R11$	$*R10=0+1\times R11$	B−2
	IF $(*R10>_u 7)$	N/A	
	JMP L3	N/A	
L2 :	$R12=R11$	$R12=0+1\times R11$	B−2
	$R12=R12 \ll 2$	$R12=0+4\times R11$	R−4.2
	$R12=R9+R12$	$R12=8000+4\times R11$	R−1.2
	$R13=*R12$	$R13=0+1\times *(8000+4\times R11)$	R−5.1
	$R13=R8+R13$	$R13=6000+1\times *(8000+4\times R11)$	R−1.2
	JMP $*R13$	N/A	
L3 :	. . .	N/A	

Figure 6: Illustration of abstraction domain

- Propagate through assignment
- Apply union at CFG merge points
- Check abstract value at indirect jumps
- Retrieve jump table entries iteratively
- Update new edge to CFG

Bounds Analysis

SJA uses bound analysis to identify the range of jump table. It novelly supports **bi-directional** equality relationships by classifying them into distinct classes.

$$
\begin{aligned}
U &:= Z \\
V &:= 3 \cdot U + 4 \\
Y &:= -X + 4 \\
Z &:= 3 \cdot X + 2
\end{aligned}
$$

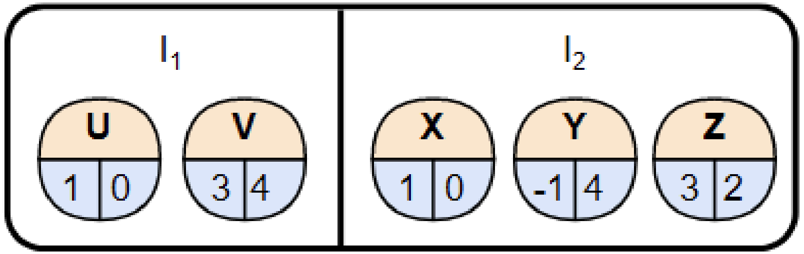


Fig. 6: Equality representation using *equivalence class*

```
if (y < 5) goto ...
x = y
if (x < 5) ...
```

```
x = y
if (y < 5) goto ...
if (x < 5) ...
```

The concrete value of the base variable (I_1, I_2) is extracted from jump instruction, for example: cmp rax, 3; jle $\rightarrow$ rax $\in (-\infty, 3)$

4. Evaluation

Setup

- Implementation: based on the LISC[1] binary lifter and GCC's RTL IR
- Benchmark from the x86-sok[2]: C/C++ binaries compiled with GCC and LLVM
- Ground-Truth:
  - Hook the compiler backend to generate jump table
  - Generate CFG from angr, Dyninst, Ghidra

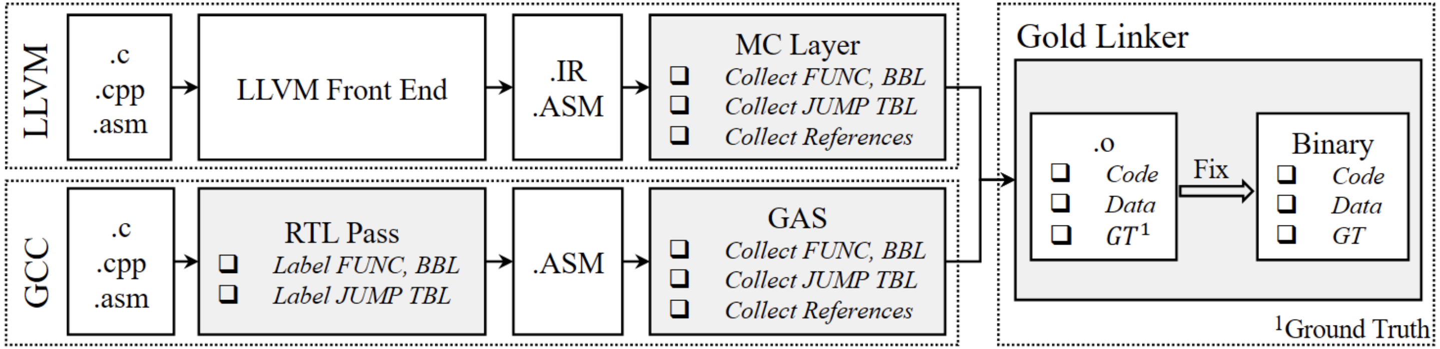


Figure 8: Get ground truth from the compiler pipeline

[1] Lifting Assembly to Intermediate Representation: A Novel Approach Leveraging Compilers, ASPLOS 16

[2] SoK: All You Ever Wanted to Know About x86/x64 Binary Disassembly But Were Afraid to Ask, IEEE S&P '21

Accuracy

- Metric: the identified jump targets for each jump table
- SJA achieves a precision, recall and f1-score of 97.4%, 99.8%, and 98.6%

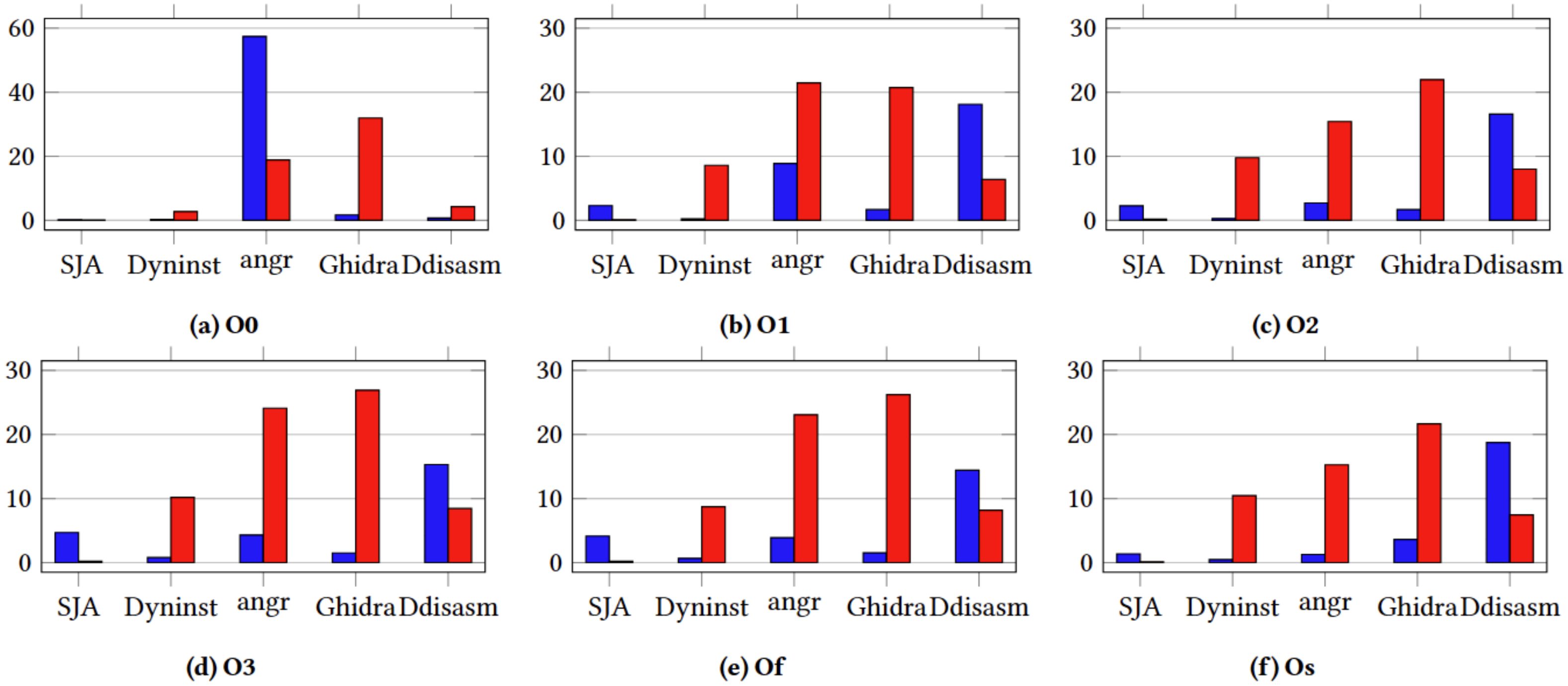


Figure 9: Error rates in recognizing jump table entries. Blue and red bars are (1-Precision) and (1-Recall), respectively (%)

Ablation Study

- VSA's[1] abstract domain[2]: value-set analysis, no pattern-matching, high FP/FN
- Unidirectional propagation[3]: No equality relation, high FP

	VSA's domain Unidirectional	**Domain $\mathcal{D}$** Unidirectional	**Domain $\mathcal{D}$ Bidirectional**
Precision	80.4	79.9	97.4
Recall	78.6	99.8	99.8

Figure 10: Comparison between SJA and previous techniques

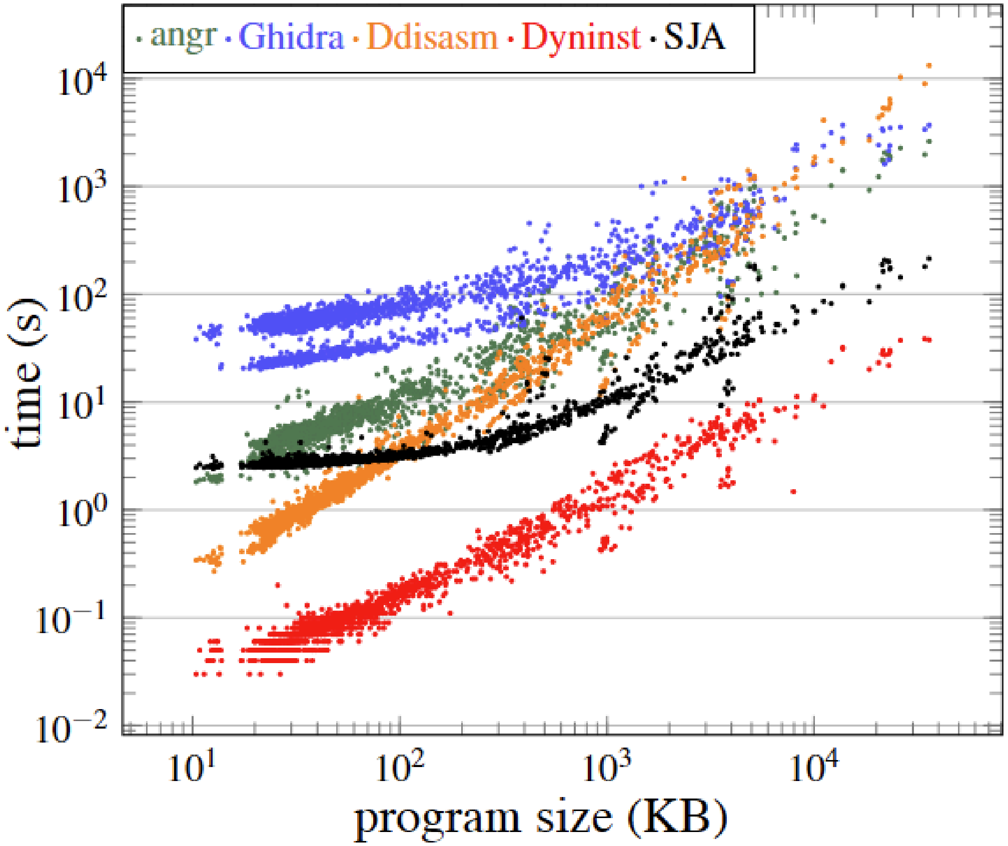
[1] WYSINWYX: What you see is not what you eXecute, TOPLAS '10

[2] Used by angr (IEEE S&P '16) and Dyninst (ISSTA '16)

[3] Used by Dyninst

Scalability

SJA's runtime exhibits a linear relation with the binary size, e.g. 36MB in 3 min



5. Conclusion

One sentence to conclude

Leveraging the linear nature of jump table, SJA adapts the VSA to compute a sound jump table expression.

Why the paper is accepted

- Focus on a very fundamental, but under-studied problem
- Principled approach, following the style of their research group

Problems about this paper

- Not evaluated on other architectures (e.g. ARM, MIPS, …)
- The soundness relies on the accuracy of the pre-analysis (e.g. function detection)
- The CFG ground truth is not well established (e.g. indirect jump)

Any Question?