Lightweight Structured Visualization of Assembler Control Flow based on Regular Expressions

Sibel Toprak, Arne Wichmann, and Sibylle Schupp
(Hamburg University of Technology)

Fan Li

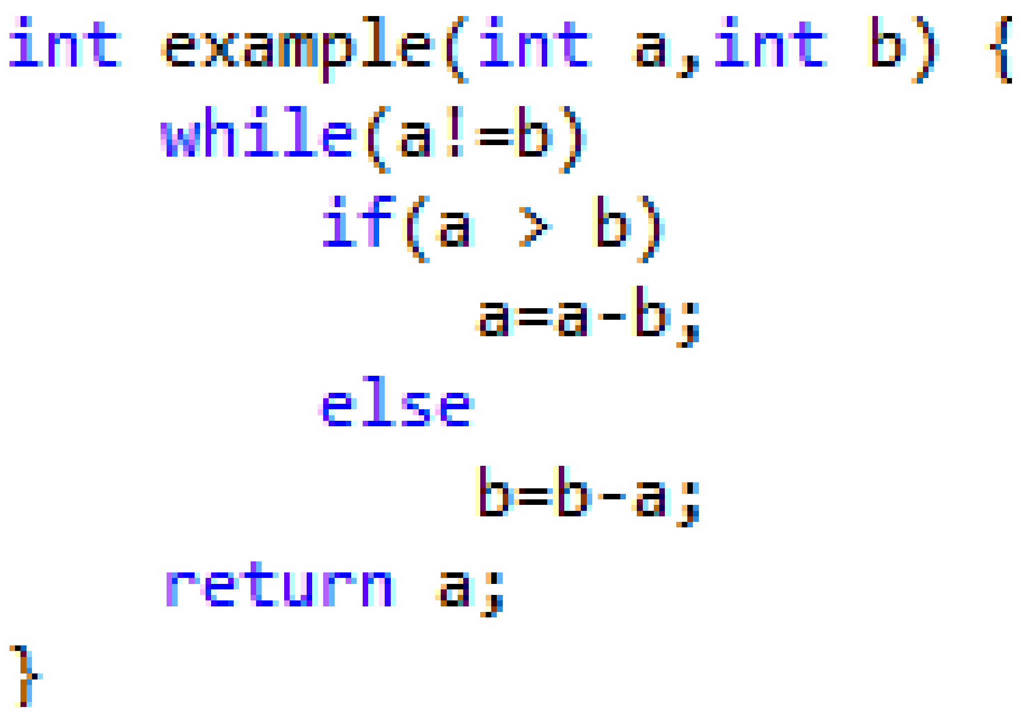
CISC850
Cyber Analytics

Highlights

- Used Regular Expressions (RE) to summarize software control-flow graph (CFG).
- Convert CFG to control flow blocks (CFB) using RE.
- Developed **regVIS** UI to analyze graphs in both CFG and CFB formats.
- Organized a 10-person usability study to compare CFG and CFB

What is Control-Flow Graph

C program to compute the greatest common divisor of two integers.



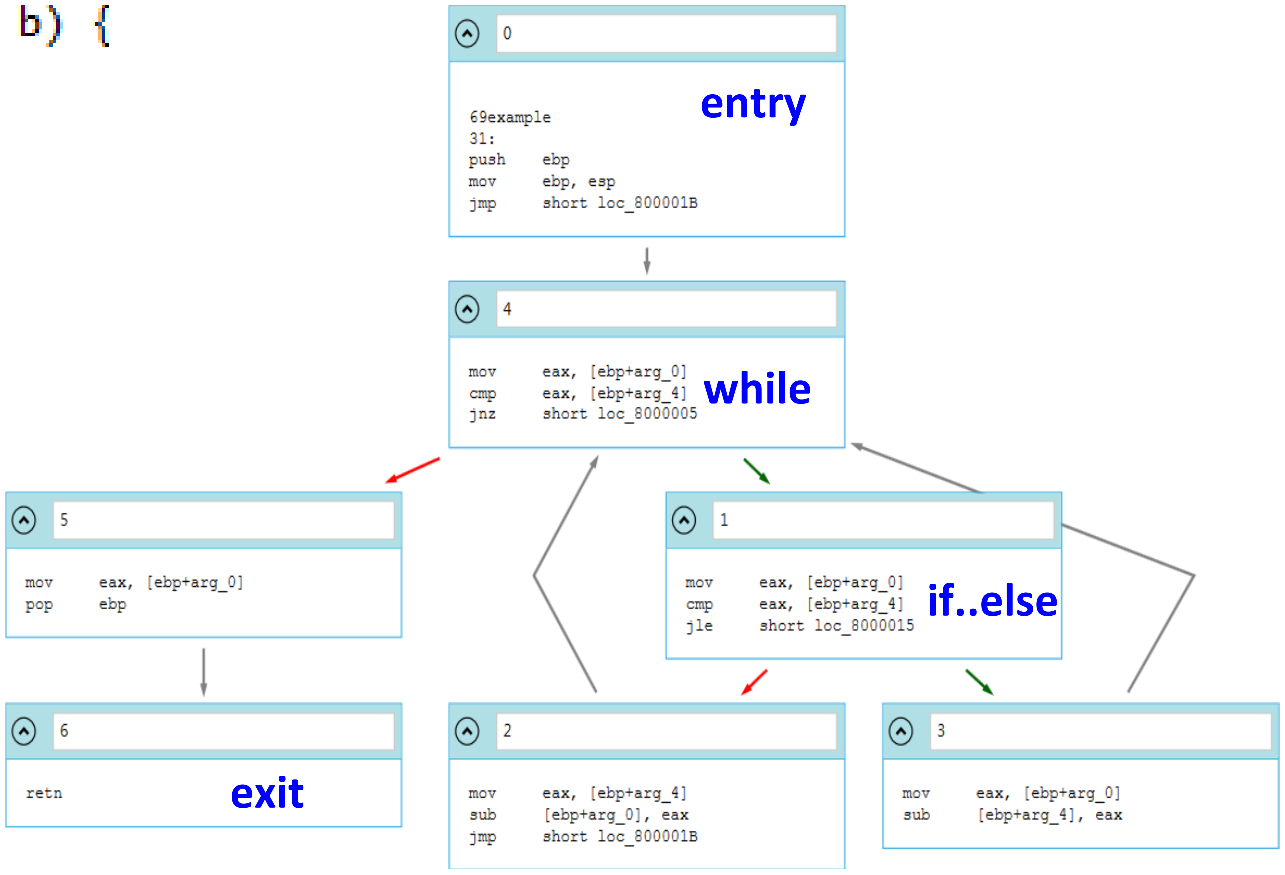
```
int example(int a,int b) {
    while(a!=b)
        if(a > b)
            a=a-b;
        else
            b=b-a;
    return a;
}
```

Control-Flow Graph

Source code

```
int example(int a,int b) {
    while(a!=b)
        if(a > b)
            a=a-b;
        else
            b=b-a;
    return a;
}
```

Assembler code (decompiled from binary)



```
0:
69example
31:
push    ebp
mov     ebp, esp
jmp     short loc_800001B

4:
mov     eax, [ebp+arg_0]
cmp     eax, [ebp+arg_4]
jnz     short loc_8000005

5:
mov     eax, [ebp+arg_0]
pop     ebp

1:
mov     eax, [ebp+arg_0]
cmp     eax, [ebp+arg_4]
jle     short loc_8000015

6:
retn

2:
mov     eax, [ebp+arg_4]
sub     [ebp+arg_0], eax
jmp     short loc_800001B

3:
mov     eax, [ebp+arg_0]
sub     [ebp+arg_4], eax
```

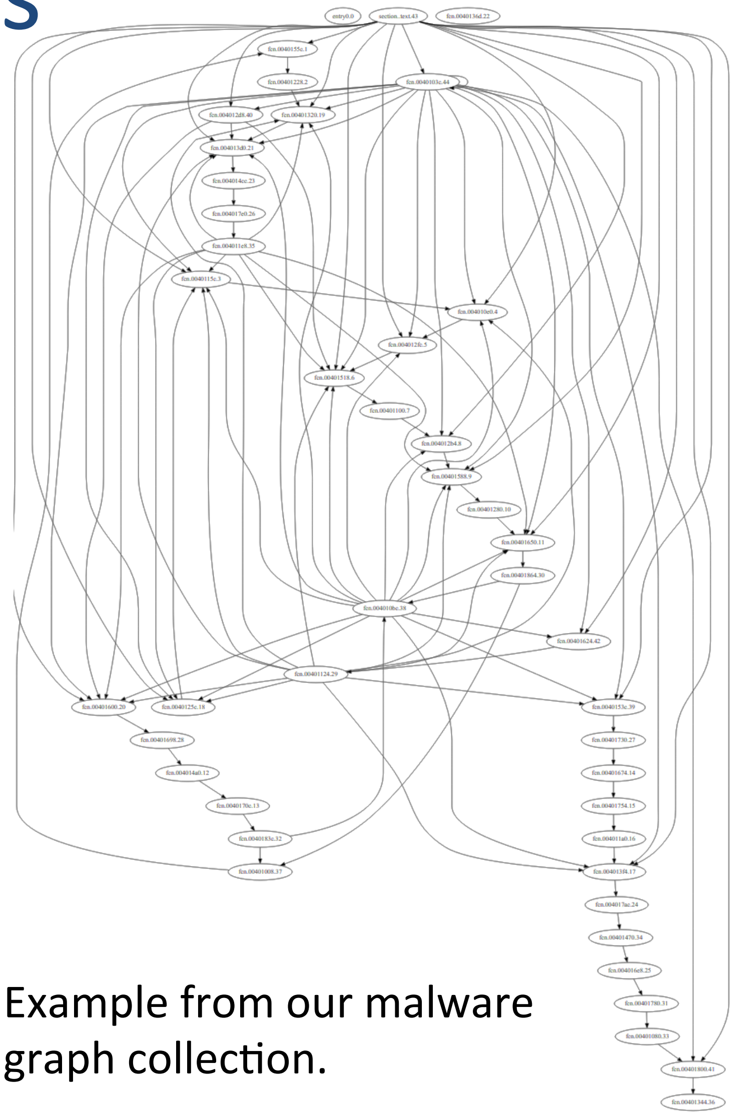
Challenges

Real world CFG are:

– Complicated

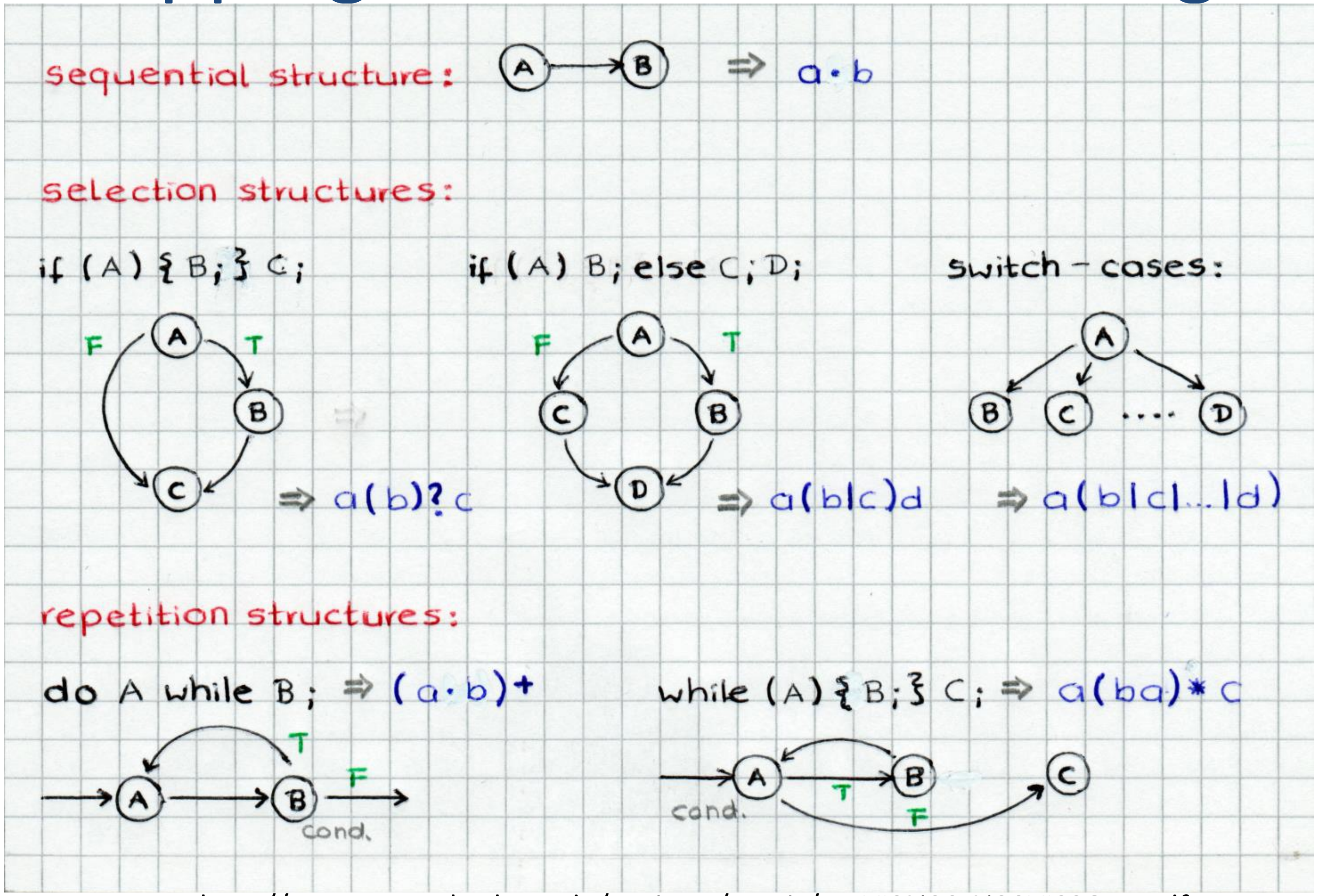
– Hard to reason

– Even hard to compare



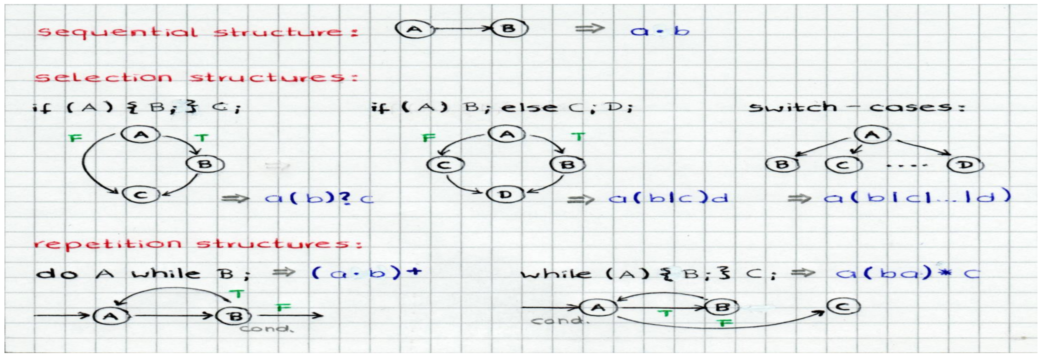
Example from our malware graph collection.

Mapping CFG Structures to Regex



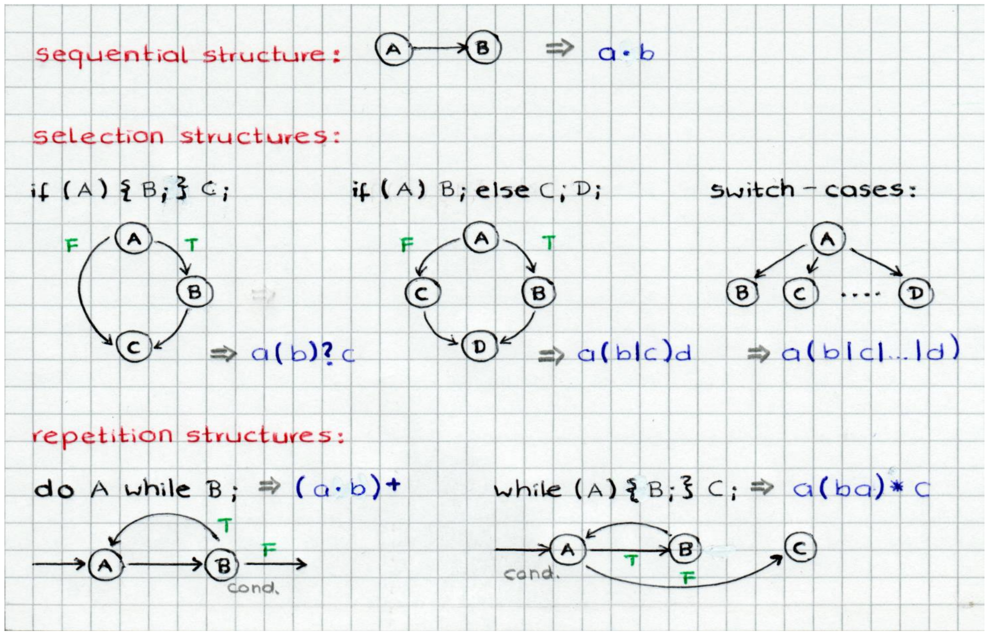
http://www.sts.tu-harburg.de/projects/regvis/regVIS%20-%20VISSOFT.pdf

Structure 1 - Sequential



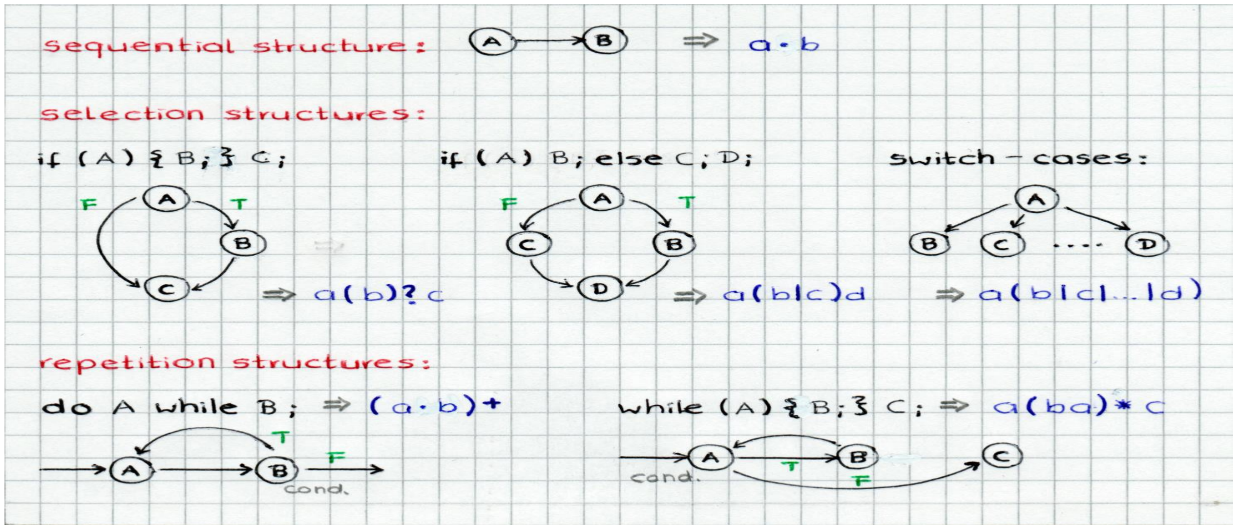
a · b

Structure 2 - Conditional



a (b | c) d

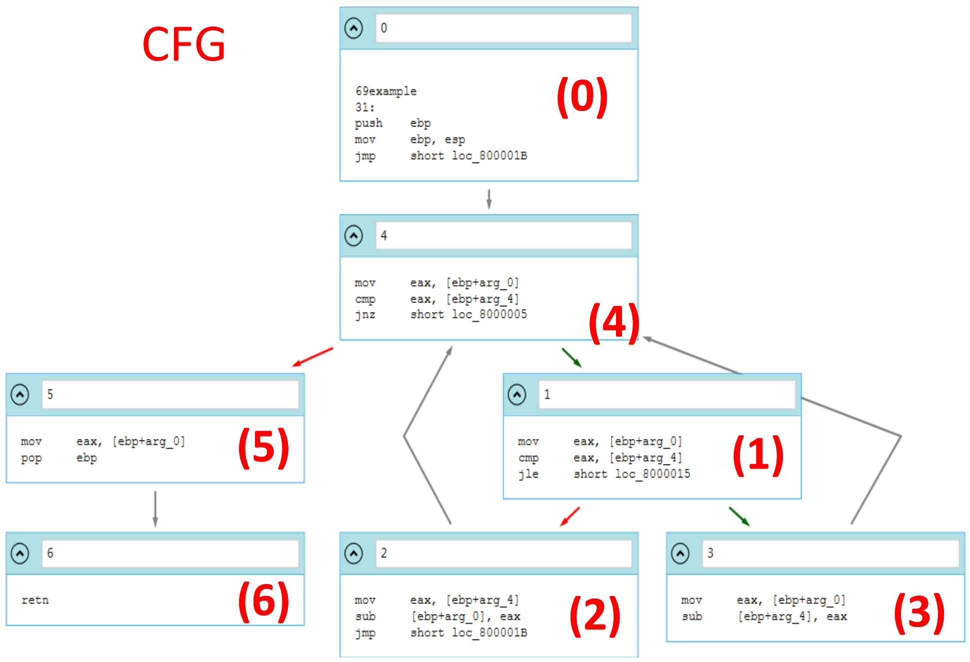
Structure 3 - Loop



$$(a \cdot b)^*$$

Convert CFG to CFB

CFG



```
0
69example
31:
push    ebp
mov     ebp, esp
jmp     short loc_800001B
```

```
4
mov     eax, [ebp+arg_0]
cmp     eax, [ebp+arg_4]
jnz     short loc_8000005
```

```
5
mov     eax, [ebp+arg_0]
pop     ebp
```

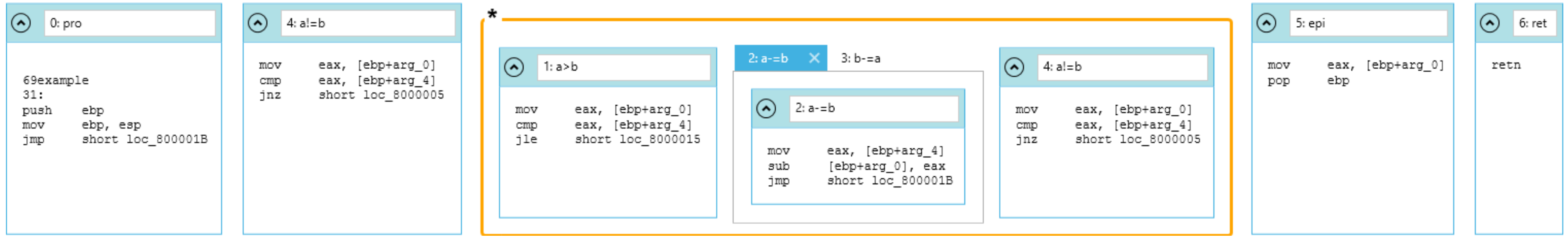
```
1
mov     eax, [ebp+arg_0]
cmp     eax, [ebp+arg_4]
jle     short loc_8000015
```

```
6
retn
```

```
2
mov     eax, [ebp+arg_4]
sub     [ebp+arg_0], eax
jmp     short loc_800001B
```

```
3
mov     eax, [ebp+arg_0]
sub     [ebp+arg_4], eax
```

CFB (Control flow blocks)



```
0: pro
69example
31:
push    ebp
mov     ebp, esp
jmp     short loc_800001B
```

```
4: a!=b
mov     eax, [ebp+arg_0]
cmp     eax, [ebp+arg_4]
jnz     short loc_8000005
```

```
1: a>b
mov     eax, [ebp+arg_0]
cmp     eax, [ebp+arg_4]
jle     short loc_8000015
```

```
2: a-=b
mov     eax, [ebp+arg_4]
sub     [ebp+arg_0], eax
jmp     short loc_800001B
```

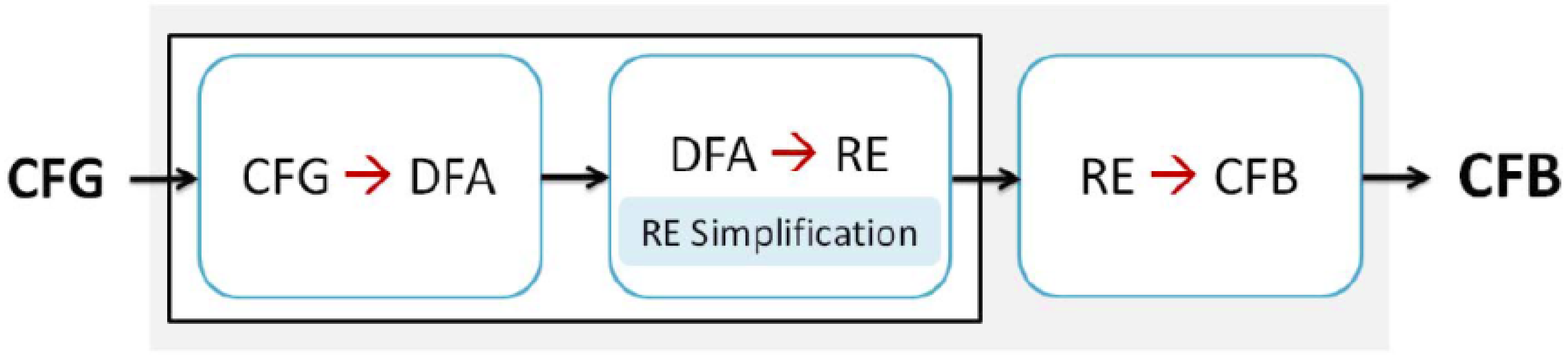
```
4: a!=b
mov     eax, [ebp+arg_0]
cmp     eax, [ebp+arg_4]
jnz     short loc_8000005
```

```
5: epi
mov     eax, [ebp+arg_0]
pop     ebp
```

```
6: ret
retn
```

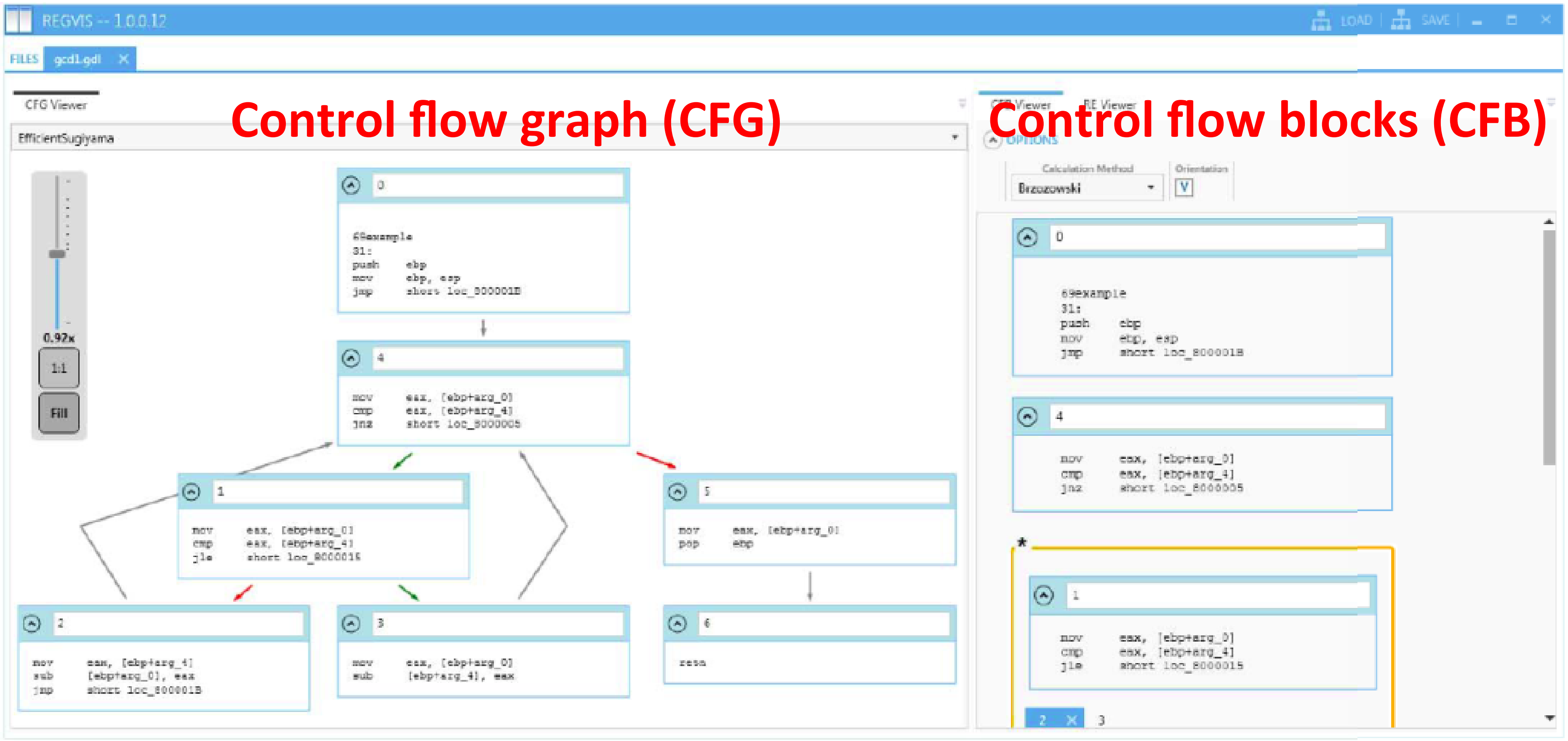
$$0 \cdot 4\,(1 \cdot (2 + 3) \cdot 4)^* \cdot 5 \cdot 6$$

Workflow



CFG – Control Flow Graph
DFA – Deterministic Finite Automaton
RE – Regular Expressions
CFB – Control Flow Block

regVIS: Regex Graph Visualization



Block 0:

```
69example
31:
push    ebp
mov     ebp, esp
jmp     short loc_800001B
```

Block 4:

```
mov     eax, [ebp+arg_0]
cmp     eax, [ebp+arg_4]
jnz     short loc_8000005
```

Block 1:

```
mov     eax, [ebp+arg_0]
cmp     eax, [ebp+arg_4]
jle     short loc_8000015
```

Block 5:

```
mov     eax, [ebp+arg_0]
pop     ebp
```

Block 2:

```
mov     eax, [ebp+arg_4]
sub     [ebp+arg_0], eax
jmp     short loc_800001B
```

Block 3:

```
mov     eax, [ebp+arg_0]
sub     [ebp+arg_4], eax
```

Block 6:

```
retn
```

Functions:

1. Convert any CFG to CFB
2. Side-by-side visualization of CFG and CFG

Usability study

User: 10 computer science students

Hello-World Training
- 15 minutes

Self-assessment, reading, tutorial

CFG:
Parser #1
- 25 minutes

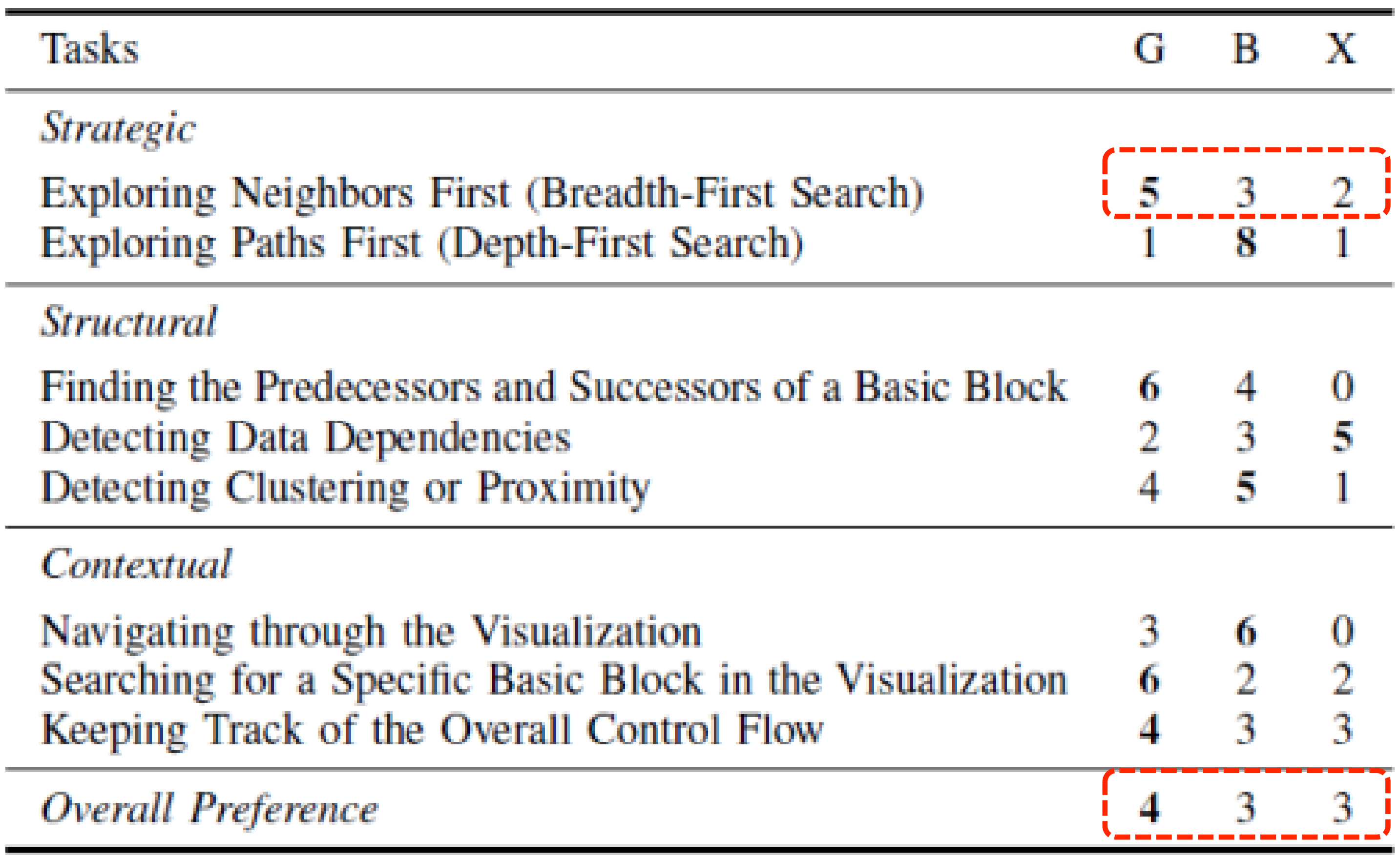
CFB:
Parser #2
- 25 minutes

Graph analysis using CFG and CFB and answer questions

Feedback
- 30 minutes

Discuss their preferences for different use cases.

Result: Feedback



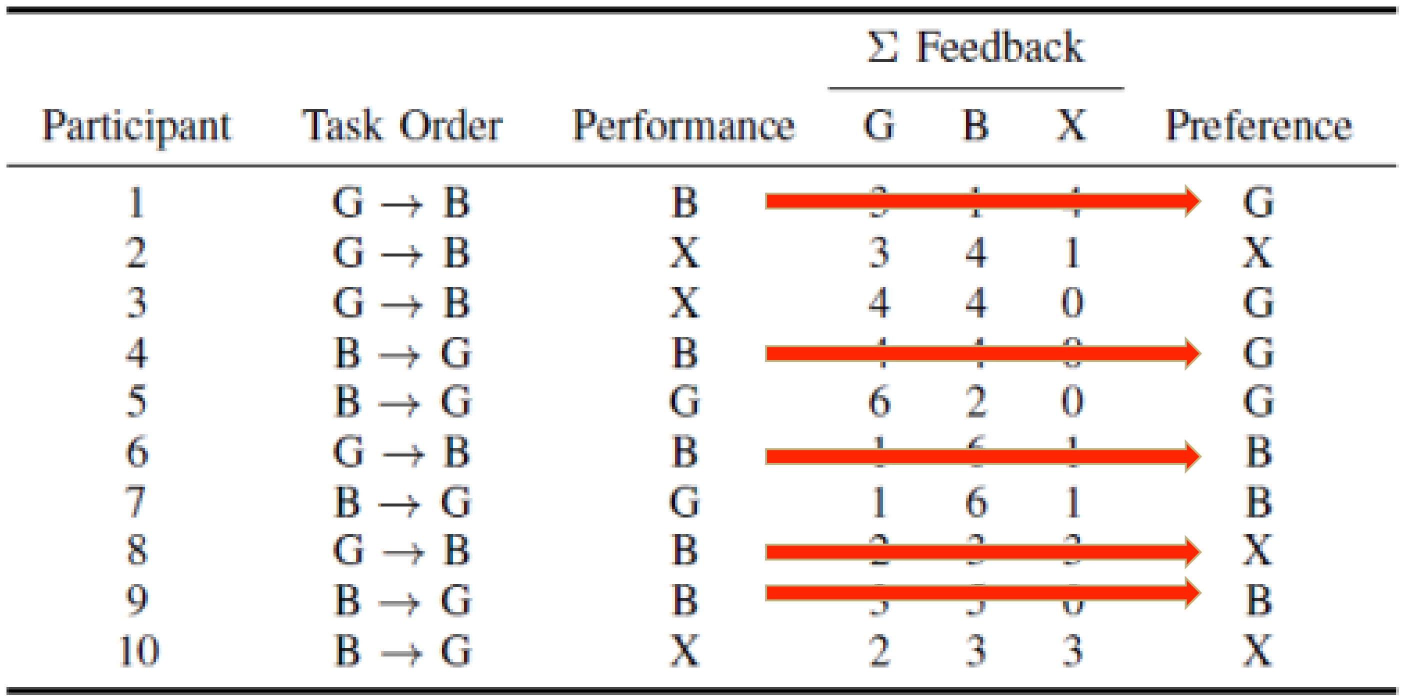
Tasks	G	B	X
Strategic			
Exploring Neighbors First (Breadth-First Search)	**5**	3	2
Exploring Paths First (Depth-First Search)	1	**8**	1
Structural			
Finding the Predecessors and Successors of a Basic Block	**6**	4	0
Detecting Data Dependencies	2	3	**5**
Detecting Clustering or Proximity	4	**5**	1
Contextual			
Navigating through the Visualization	3	**6**	0
Searching for a Specific Basic Block in the Visualization	**6**	2	2
Keeping Track of the Overall Control Flow	**4**	3	3
Overall Preference	**4**	3	3

G: Graph, B: Block, X: Undecided

Tasks

Feedbacks

Result: Performance



			Σ Feedback			
Participant	Task Order	Performance	G	B	X	Preference
1	$G \rightarrow B$	B	3	1	4	G
2	$G \rightarrow B$	X	3	4	1	X
3	$G \rightarrow B$	X	4	4	0	G
4	$B \rightarrow G$	B	4	4	0	G
5	$B \rightarrow G$	G	6	2	0	G
6	$G \rightarrow B$	B	1	6	1	B
7	$B \rightarrow G$	G	1	6	1	B
8	$G \rightarrow B$	B	2	3	3	X
9	$B \rightarrow G$	B	3	5	0	B
10	$B \rightarrow G$	X	2	3	3	X

G: Graph, B: Block, X: Undecided

Testers generally performed better using the CFB method despite a lack of preference.

Summary

- Used Regular Expressions (RE) to summarize software control-flow graph (CFG).
- Convert CFG to control flow blocks (CFB) using RE.
- Developed **regVIS** UI to analyze graphs in both CFG and CFB formats.
- Organized a 10-person usability study to compare CFG and CFB

Key Learnings / To Do

- RE provide a new way for control flow graph visualization
  - TODO: Try CFB in malware analysis

- RE can reduce graphs into concise mathematical forms
  - TODO: Try RE for graph feature extraction for data mining or machine learning.

Thanks to Dr. Cavazos in helping selecting this publication.