TYGR: Type Inference on Stripped Binaries using Graph Neural Networks

Chang Zhu*, Ziyang Li*, Anton Xue, Ati Priya Bajaj, Wil Gibbs, Yibo Liu,
Rajeev Alur, Tiffany Bao, Hanjun Dai, Adam Doupé, Mayur Naik,
Yan Shoshitaishvili, Ruoyu Wang, and Aravind Machiry

ASU Arizona State University

Penn University of Pennsylvania

PURDUE UNIVERSITY

* Equal contribution

Compilation

```
int main (int count)
{
  return count++;
}
```

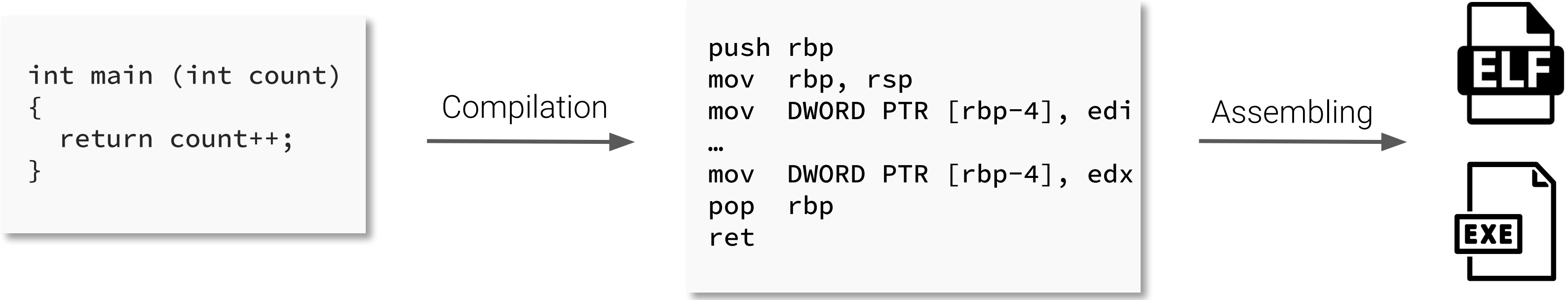
Compilation

```
int main (int count)
{
  return count++;
}
```

Compilation

```
push rbp
mov  rbp, rsp
mov  DWORD PTR [rbp-4], edi
…
mov  DWORD PTR [rbp-4], edx
pop  rbp
ret
```

Compilation



```
int main (int count)
{
  return count++;
}
```

Compilation

```
push rbp
mov  rbp, rsp
mov  DWORD PTR [rbp-4], edi
…
mov  DWORD PTR [rbp-4], edx
pop  rbp
ret
```

Assembling

Loss of Information

- Variable types
- Variable names
- Control flow structures
- Comments

```
int main (int count)
{
  return count++;
}
```

Compilation

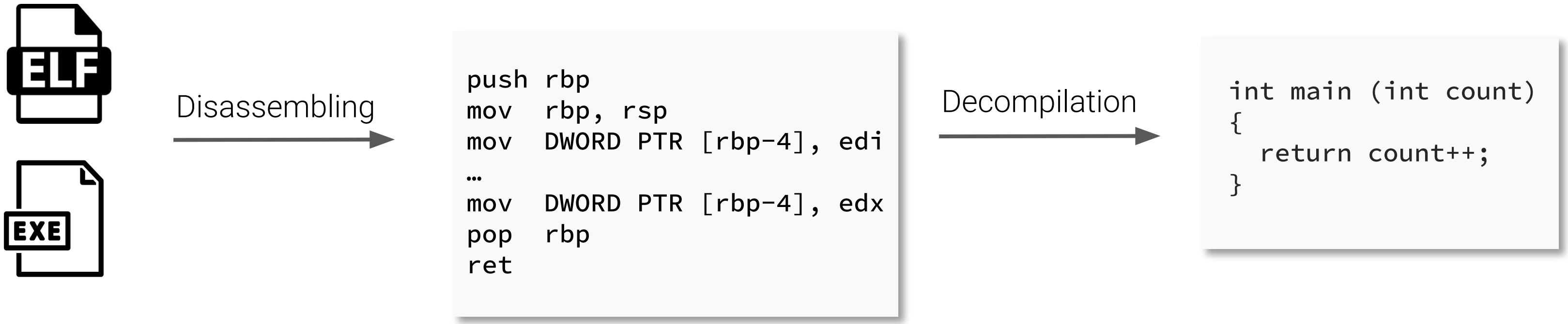
```
push rbp
mov  rbp, rsp
mov  DWORD PTR [rbp-4], edi
…
mov  DWORD PTR [rbp-4], edx
pop  rbp
ret
```

Loss of Information

The information is critical for understanding binaries

- Malware analysis
- Software debugging
- Software maintenance

Reverse Engineering



Disassembling

```
push rbp
mov  rbp, rsp
mov  DWORD PTR [rbp-4], edi
…
mov  DWORD PTR [rbp-4], edx
pop  rbp
ret
```

Decompilation

```
int main (int count)
{
  return count++;
}
```

Reverse Engineering

Loss of information:

- Variable types

The Variable Type Recovery Problem

```c
struct options {
  int flag;
  char name;
};

void func() {
  struct options opt;
  int tmp;
  opt.flag = 1;
  opt.name = 'a';
  tmp = opt.flag + 1;
}
```

```
Func:
     push  rbp
     mov   rbp, rsp
     mov   DWORD PTR [rbp-12], 1
     mov   BYTE PTR [rbp-8], 97
     mov   eax, DWORD PTR [rbp-12]
     add   eax, 1
     mov   DWORD PTR [rbp-4], eax
     pop   rbp
     ret
```

The Variable Type Recovery Problem

```
func:

        opt.flag: ?

        opt.name: ?

        tmp: ?
```

```
func:

        [rbp - 12]

        [rbp - 8]

        [rbp - 4]
```

Variable Type Recovery Attempts

1. Constraint-based methods

- Collect type constraints to encode data flow information:

```
add(x, 5) → x: numeric
```

- Solve constraints: writing rules

Variable Type Recovery Attempts

1. Constraint-based methods

- Collect type constraints to encode data flow information:

```
add(x, 5) → x: numeric
```

- Solve constraints: writing rules

2. ML-based methods

- Implicitly encode data flow information
- No need to write rules

Our Intuition

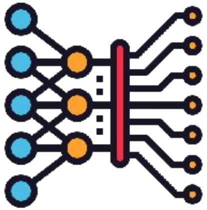
- Explicitly encode data flow information into graph
    - Variable access patterns
    - Variable usage

- No need to manually write rules

TYGR

Model training

- Data flow analysis
- Graph embedding



TYGR

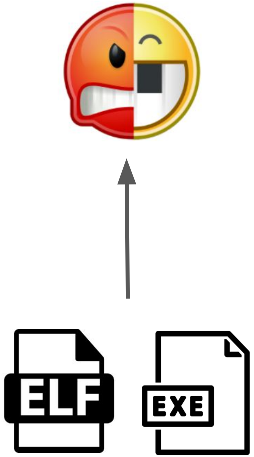
Model training

- Data flow analysis
- Graph embedding

Building a data set

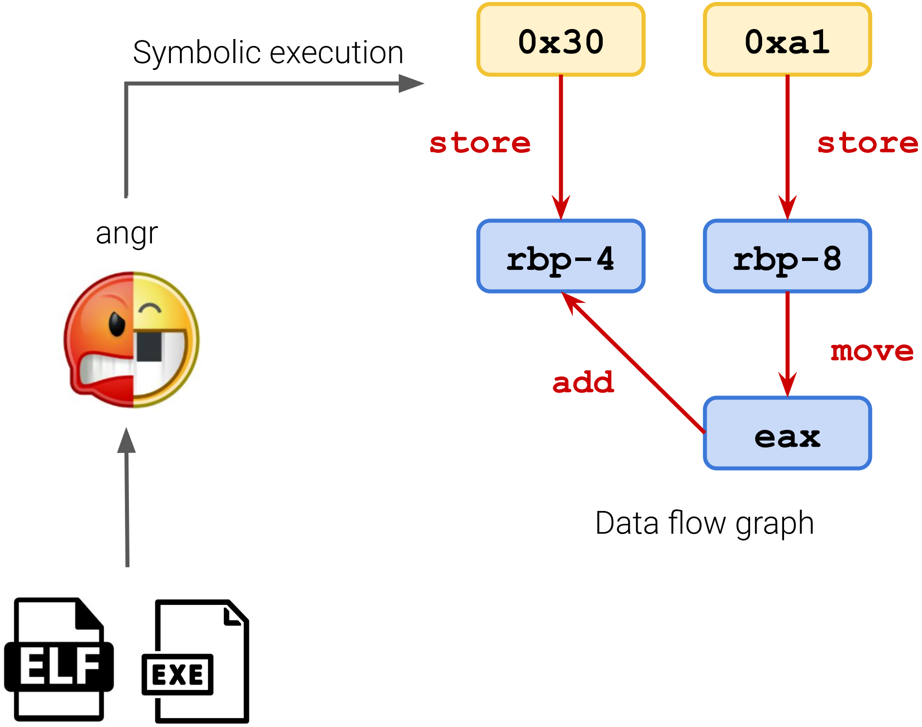
- Software source code
- x64, x86, AArch64, ARM, MIPS

TYGR: Pipeline



angr

TYGR: Pipeline

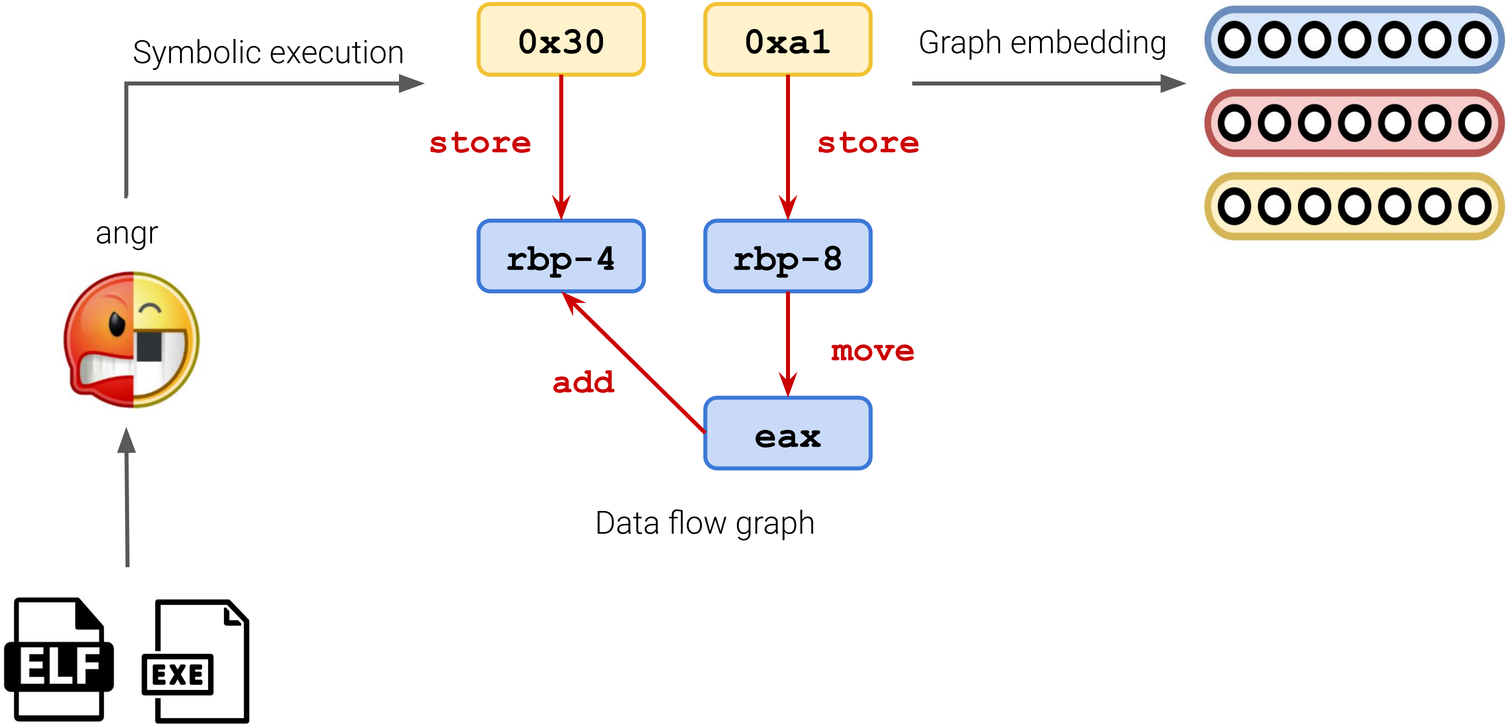


Symbolic execution

angr

Data flow graph

TYGR: Pipeline



Symbolic execution

angr

`0x30`

`0xa1`

`store`

`store`

`rbp-4`

`rbp-8`

`move`

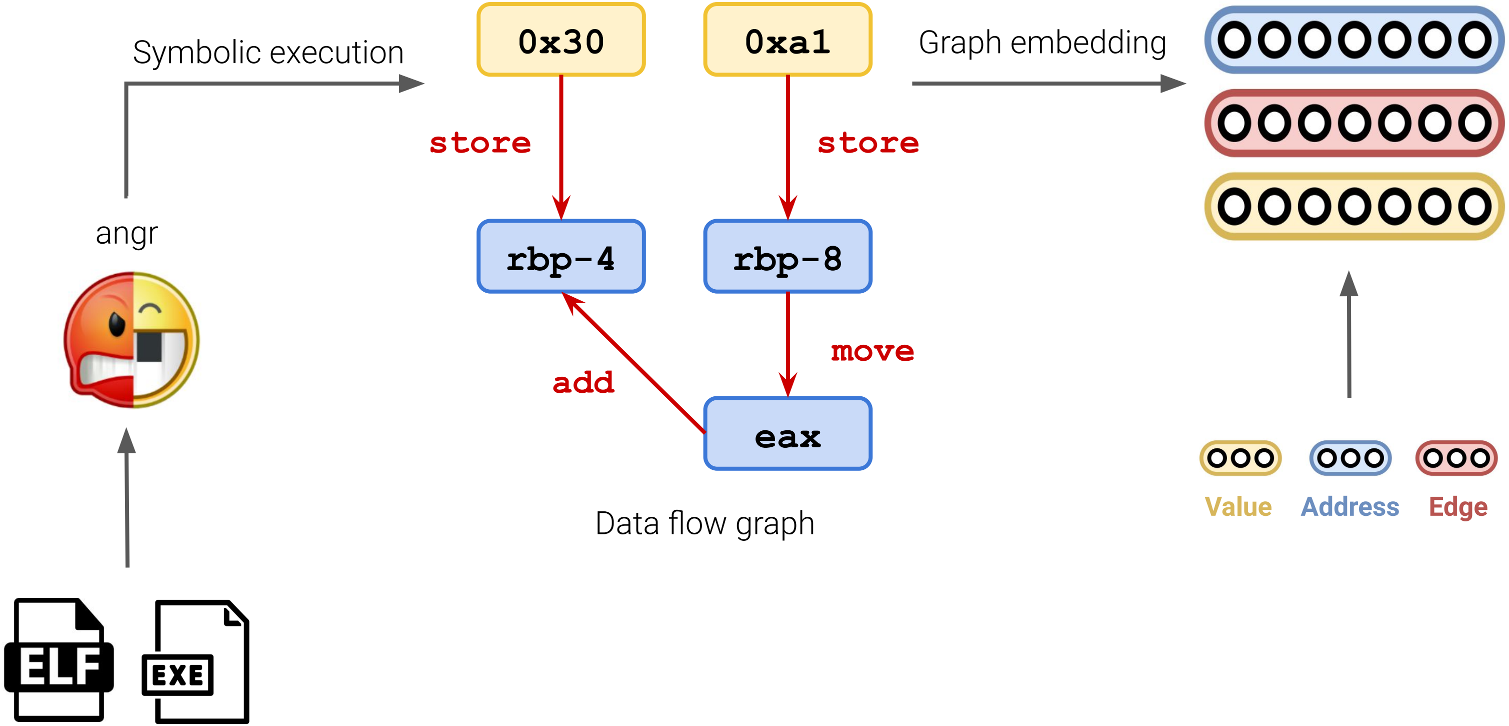
`add`

`eax`

Data flow graph

Graph embedding

TYGR: Pipeline



Symbolic execution

angr

`0x30`

`0xa1`

`store`

`store`

`rbp-4`

`rbp-8`

`move`

`add`

`eax`

Data flow graph

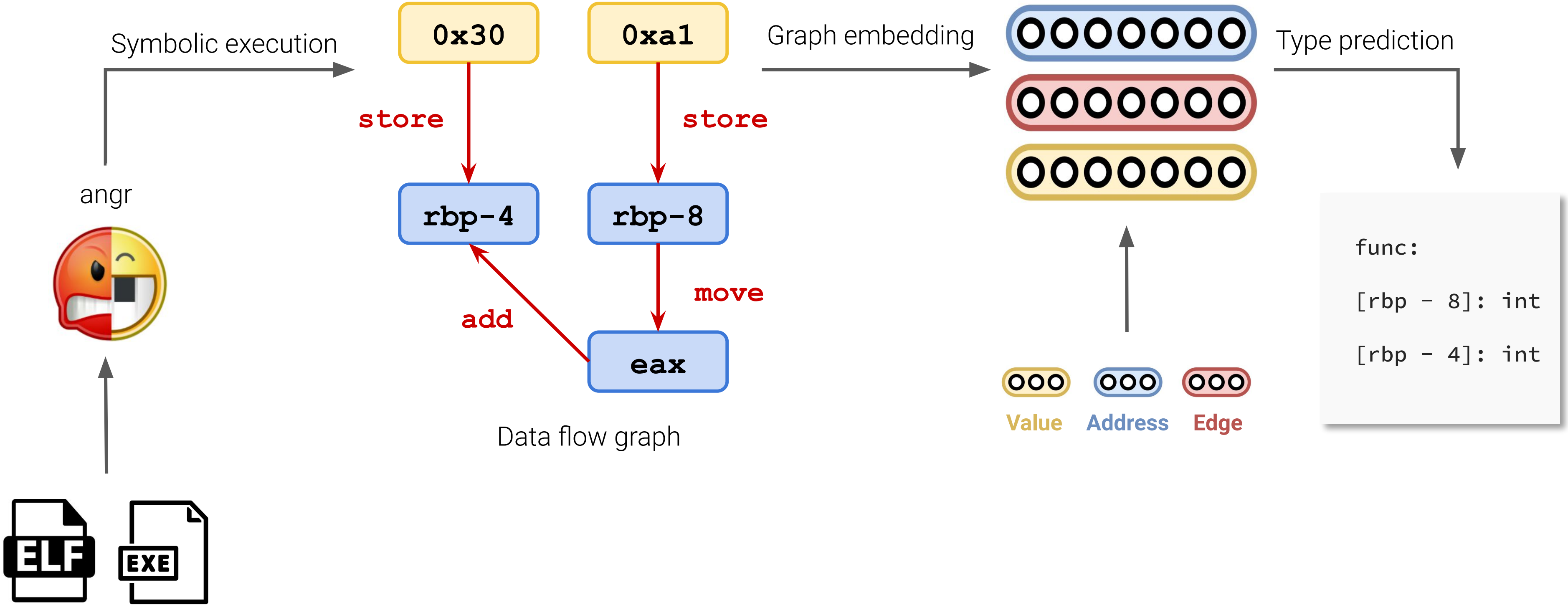
Graph embedding

Value

Address

Edge

TYGR: Pipeline



Symbolic execution

angr

Data flow graph

Graph embedding

Value

Address

Edge

Type prediction

```
func:
[rbp - 8]: int
[rbp - 4]: int
```

Challenge: Type Recovery for `struct`

```c
struct options {
  int flag;
  char name;
};

void func() {
  struct options opt;
  int tmp;
  opt.flag = 1;
  opt.name = 'a';
  tmp = opt.flag + 1;
}
```

Challenge: Type Recovery for `struct`

```
func:
    opt.flag: ?
    opt.name: ?
    tmp: ?
```

`struct` Type Recovery by Size

```
func:

    opt.flag: struct<4, 1>

    opt.name: struct<4, 1>

    tmp: Primitive_4
```

Problems:

1. Types with same size
2. Struct with same shape

`struct` Type Recovery using ML

```
func:

    opt.flag: struct<int, char>

    opt.name: struct<int, char>

    tmp: int
```

Problem:

Only if `struct<int, char>` is in the dictionary

TYGR

```
func:
    opt.flag: struct
    opt.name: struct
    tmp: int
```

Step 1: Basic type recovery

TYGR

```
func:

    opt.flag: int

    opt.name: char
```

Step 2: Member type recovery

TYGR

```
func:
    opt.flag: struct_int
    opt.name: struct_char
    tmp: int
```

Step 1: Basic type recovery

Step 2: Member type recovery

Dataset: TYDA

- Compiled C binary executables from Gentoo and Debian repositories
- Five architectures: x64, x86, AArch64, ARM32, and MIPS
- Four compiler optimizations O0 through O3
- 327K binaries and 130M functions in TYDA

Evaluation: Baselines

	Overall Accuracy	Struct Accuracy
TYGR	**74.5%**	**40.6%**
DIRTY	55.8%	34.1%
OSPREY	71.8%	29.5%

Accuracy results on GNU coreutils `O0` executables

Evaluation: Accuracy per-type (x64 O0)

Type	Occurrence (%)	Accuracy (%)
struct*	25.6	91.1
i32	23.1	90.9
char*	16.3	74.7
bool	1.7	83.9
…	…	…
i16*	0.1	55.0

Case Study: Bool

Type	Occurrence (%)	Accuracy (%)
char*	16.3	74.7
bool	**1.7**	**83.9**

Conclusion

- TYGR is a system that uses a novel graph-based representation of data-flow information for type inference
- We construct a new dataset for x64, x86, AArch64, ARM32, and MIPS architectures

Thank you

https://github.com/sefcom/TYGR

Chang Zhu
Chang.Zhu@asu.edu

Ziyang Li
liby99@seas.upenn.edu