
# Recovering Types from Binaries

**Teodora Baluta** Shiqi Shen Alexandros Dimos

Advised by prof. Prateek Saxena
School of Computing, NUS
{teobaluta, shensq04, alexandros.dimos95}@gmail.com

# Decompilation

Load the Binary

Disassemble to ASM

....

**Recover Types**

High Level Code

```
loc_40055D:                             ; CODE XREF: bubblesort+D3↓j
                mov     [rbp+var_C], 0
                mov     [rbp+var_8], 1
                jmp     loc_400609
; ---------------------------------------------------------------------------

loc_400570:                             ; CODE XREF: bubblesort+C9↓j
                mov     eax, [rbp+var_8]
                cdqe
                shl     rax, 2
                lea     rdx, [rax-4]
                mov     rax, [rbp+var_18]
                add     rax, rdx
                mov     edx, [rax]
                mov     eax, [rbp+var_8]
                cdqe
```

```
2 {
3   int v2; // ST1C_4@4
4   __int64 result; // rax@6
5   signed int v4; // [sp+10h] [bp-Ch]@1
6   signed int i; // [sp+14h] [bp-8h]@2
7
8   v4 = 1;
9   while ( v4 )
10  {
11    v4 = 0;
12    for ( i = 1; ; ++i )
13    {
14      result = (unsigned int)i;
```

```
void bubblesort (int *a, int size) {
  int swapped = 1;
  while (swapped) {
    int i;
    swapped = 0;
    for (i = 1; i < size; i++) {
      if (a[i-1] > a[i] ) {
        int tmp;
        tmp = a[i-1];
        a[i-1] = a[i];
        a[i] = tmp;
        swapped = 1;
      }
    }
  }
}
```

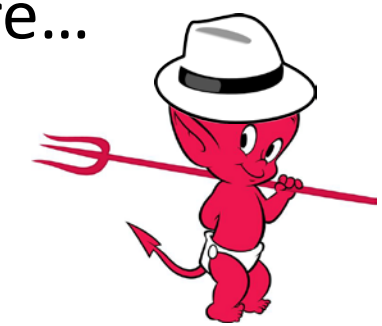
# Challenges

- Architecture and compiler dependent
  - X86, ARM, MIPS, PowerPC x gcc, clang
  - Compiler optimizations specific to the architecture
- Lack of higher-level semantics
  - During compiling any higher-level information is stripped
- Code obfuscation

# Why binary reverse engineering?

Do you trust the code you are running?

You shouldn't! Backdoors, malware…

```
ROM:0013DC50                LDR             R0, =aSCtUUnSSipSDip ; ">>> %s(ct=%u, un='%s',
ROM:0013DC54                LDR             R1, =aAuth_admin_int ; "auth_admin_internal"
ROM:0013DC58                BL              sub_5BBE74
ROM:0013DC5C
ROM:0013DC5C loc_13DC5C                     ; CODE XREF: auth_admin_internal+2C↑j
ROM:0013DC5F                ADD             R0, R5, #0x44
ROM:0013DC60                LDR             R1, =aSUnSU ; "<<< %s(un='%s') = %u"
ROM:0013DC64                BL              strcmp
ROM:0013DC68                CMP             R0, #0
ROM:0013DC6C                BNE             loc_13DC78
ROM:0013DC70                MOV             R0, #0xFFFFFFFD
ROM:0013DC74                LDMDB           R11, {R4-R8,R11,SP,PC}
ROM:0013DC78 ; ---------------------------------------------------------------------------
```

- Binary Reverse Engineering Applications
- Fuzzing for bug finding
- Binary hardening
- Commercial off-the-shelf (COTS) binary understanding

# WIP Contributions

- Can we recover types from binary code using **deep learning**?
- Build dataset for applying deep learning to binary reverse engineering
- Explore models of **explaining** *predicted* types

# Approaches

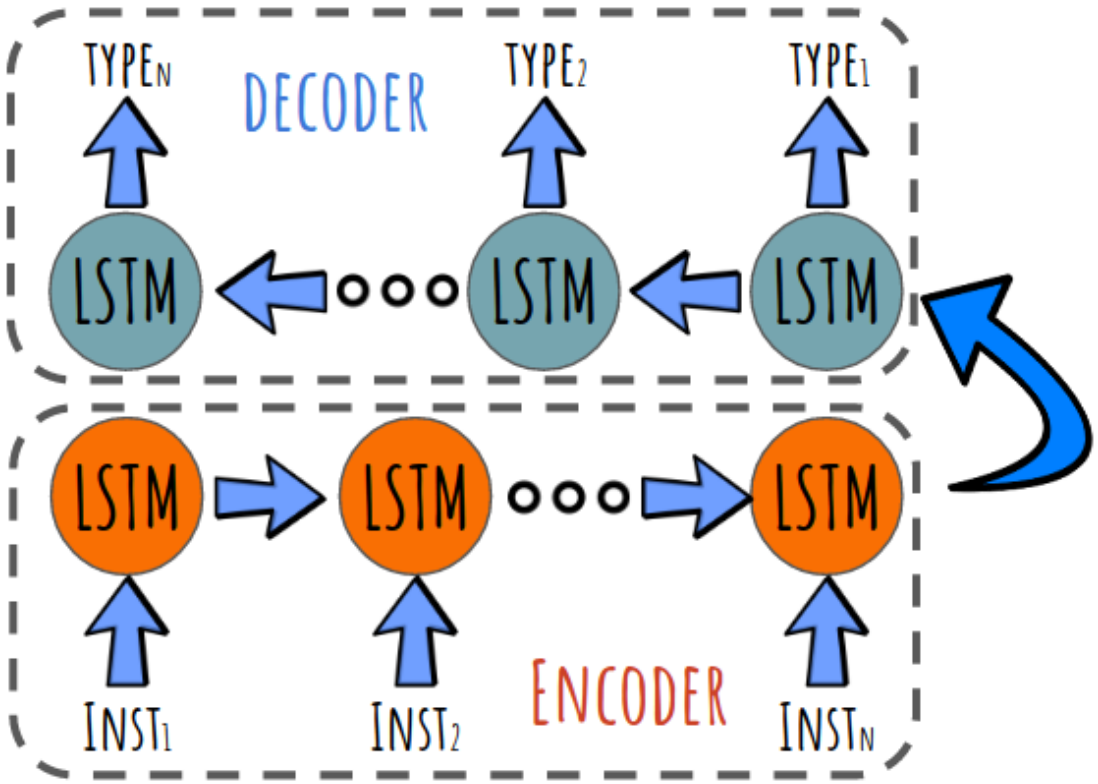
- Heuristics-based model **Pattern-based**
  - Used by one of the most popular commercial tools, Hex-rays
- Compiler model **Principled approach**
  - Type-based first proposed by Mycroft [Mycroft, LNCS'99]
  - Constraint-based type inference (TIE [Lee et al., NDSS'11], Retypd [Noonan et al., PLDI'16])
- Statistical model
  - Statistical language model to determine virtual function calls targets in C++ binaries [Katz et al., POPL'16]

# How wrong?

Type Recovery using Hex-rays for binaries in SPEC2006 (IDAPro 7, gcc 5.4)

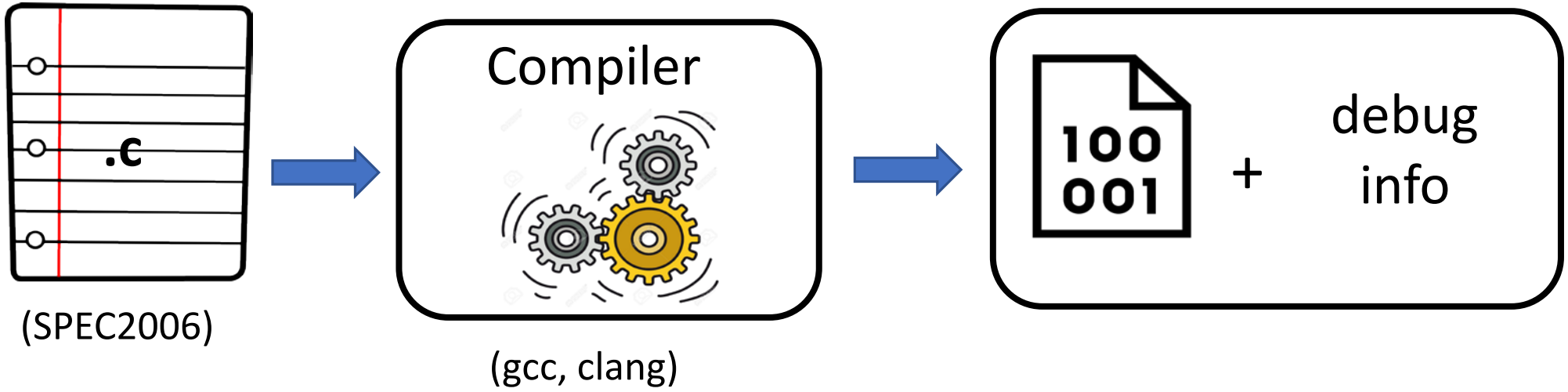
| Types | Mismatch | Mistaken Types |
|---|---|---|
| int | 24.89% | uint: 93.68% |
| | | char ptr: 1.88% |
| | | int ptr: 1.45% |
| struct ptr | 97.96% | int: 73.33% |
| | | int ptr: 8.57% |
| | | uint16 ptr: 6.53% |
| uint | 8.88% | int: 81.12% |
| | | char ptr: 8.75% |
| | | int ptr: 2.33% |
| char ptr | 43.13% | int: 44.75% |
| | | uint: 16.68% |
| | | void ptr ptr: 15.35% |
| union ptr | 100% | int: 87.11% |
| | | int ptr: 7.11% |
| | | uint: 2.92% |

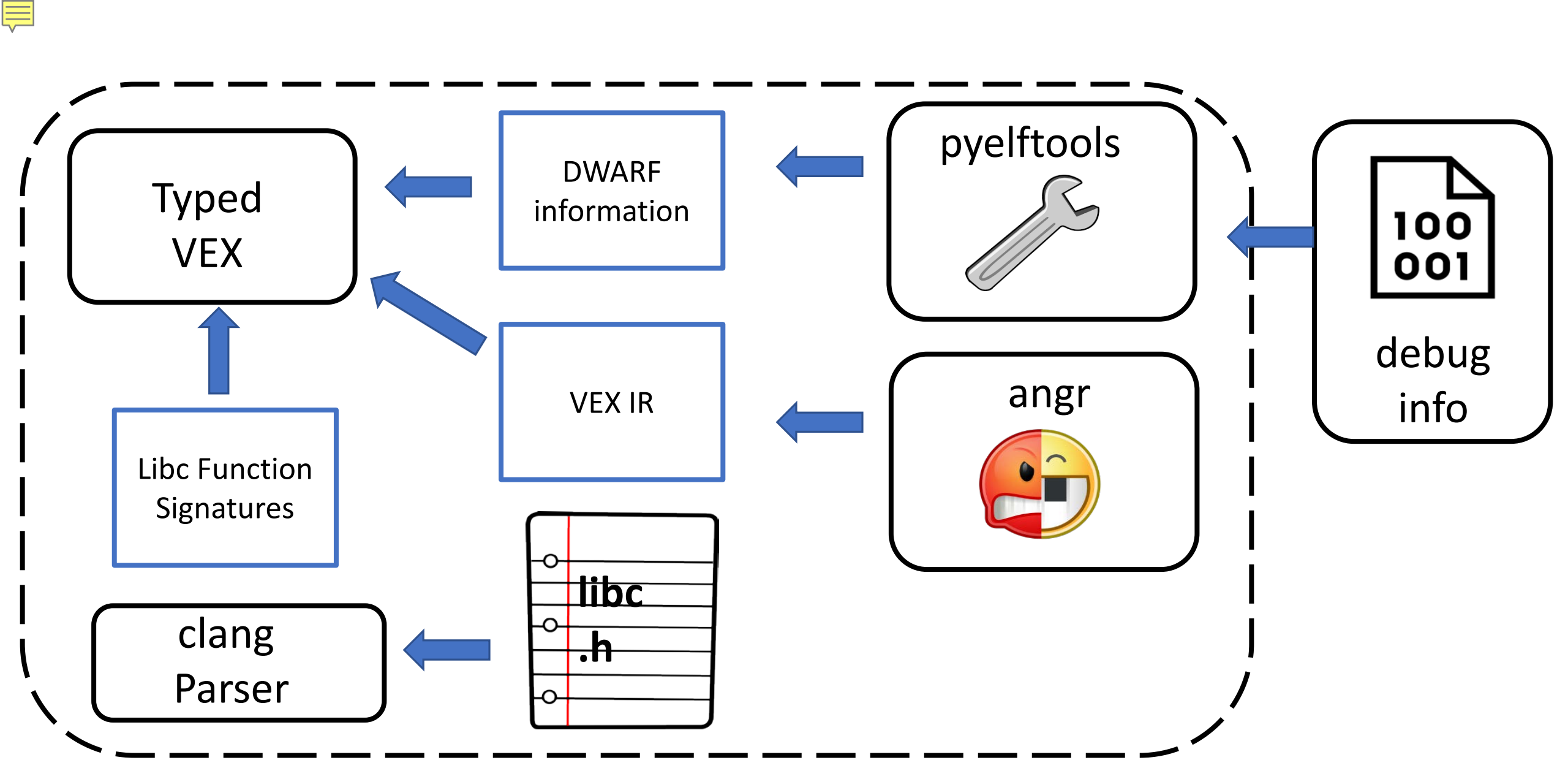
# Type Recovery using Deep Learning



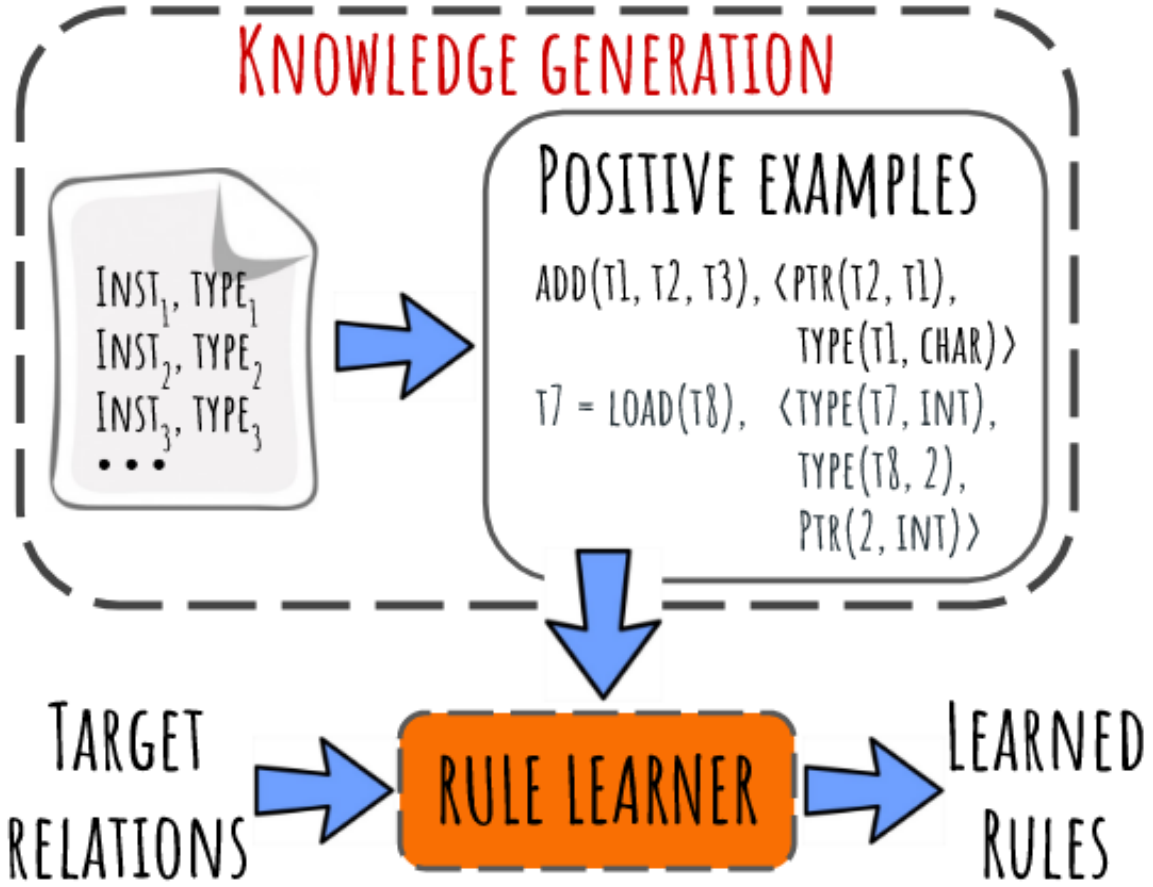

- Estimate types using a seq2seq model
- Handles variable-length input and has a notion of *memory*
- Input
  - a sequence of instructions
- Output
  - a sequence of types

# Dataset Creation




Typed
VEX
DWARF
information
pyelftools
100
001
debug
info
VEX IR
angr
Libc Function
Signatures
libc
.h
clang
Parser

# Type Rules from Predicted Types




- From the predicted types, construct a knowledge base (examples of relations)
- Learn the typing rules in first-order logic form
- Implemented using First Order Inductive Learner (FOIL) algorithm

# FOIL in a nutshell


Instructions

t7 = load(t8)
t9 = load(t10)
t11 = load(t12)
t13 = load(t14)

Predicted types
t7 – int
t8 – int*
t9 – struct
t10 – struct*
t11 – char
t12, t13 – char*
t14 – char**

Encoding
t7 – 7
int - 0
t8 – 8
….

Facts in first-order logic

# FOIL in a nutshell


- Use **load**, **ptr** and **type** relations to learn the **type** formula

***type** Pos*: [(7, 0), (8, 2), (9, 1), (10, 3), (11, 0), (12, 2), (13, 5), (14, 6)]
***type** Neg*: *Universe - Pos*

set of predicates: **load**, **ptr**
**+ examples load, ptr**

Target Relation: type

FOIL learning algorithm

Set of rules for **type**

# FOIL in a nutshell


- To learn a set of rules do a specific-to-general search
  - Each new rule covers a subset of positive examples (more general)
  - Delete all positive examples covered by the new rule
- To learn a rule, do a general-to-specific search
  - Initially, there are no preconditions
  - Each new literal adds to the number of negative examples (more specific)

# Preliminary Results

- For small examples learned rules are correctly formed
  - type($X_1$, $X_2$):-ptr($X_2$, $X_3$), load($X_4$, $X_1$), type($X_4$, $X_3$).
  - type($X_1$, $X_2$):-ptr($X_3$, $X_2$), load($X_1$, $X_4$), type($X_4$, $X_3$).
- Limitations
  - Using **information gain** is not a good measure
  - Requires richer and fewer examples (not scalable)

# Conclusions

- Developed dataset to work on binaries using deep learning
- Explored FOIL as a type rule learning system and discovered the limitations
- Future work
  - Recover types using a deep learning model
  - Adapt FOIL to probabilistic reasoning, scale it [Qiang Zeng et al., VLDB'14] or use a differentiable first-order learner [Fan Yang et al., arXiv:1702.08367]

Thank you!

# References


[Mycroft, LNCS 1999] Mycroft, Alan. "Type-based decompilation." *Lecture notes in computer science* (1999): 208-223.

[Lee et. al, NDSS 2011] Lee, JongHyup, Thanassis Avgerinos, and David Brumley. "TIE: Principled reverse engineering of types in binary programs." In Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS'11), San Diego, CA, February 2011.

[3] Noonan, Matt, Alexey Loginov, and David Cok. "Polymorphic type inference for machine code." In *PLDI*, 2016.

[4] Katz, Omer, Ran El-Yaniv, and Eran Yahav. "Estimating types in binaries using predictive modeling." In *POPL*, 2016.

[5] Quinlan, J. Ross. "Learning logical definitions from relations." *Machine learning* 5, no. 3 (1990)

[6] Zeng, Qiang, Jignesh M. Patel, and David Page. "Quickfoil: Scalable inductive logic programming." *Proceedings of the VLDB Endowment* 8, no. 3 (2014)

[7] Yang, Fan, Zhilin Yang, and William W. Cohen. "Differentiable Learning of Logical Rules for Knowledge Base Completion." *arXiv preprint arXiv:1702.08367* (2017)