“Techniques for automatically reasoning about and improving software quality”

Claire Le Goues,
Cylab Partner’s Conference, 2019
DIRE: A Neural Approach to Decompiled Identifier Naming

Claire Le Goues,
Cylab Partner’s Conference, 2019
DIRE: A Neural Approach to Decompiled Identifier Naming

Jeremy Lacomis, Pengcheng Yin, Ed Schwartz, Miltos Allamanis, Claire Le Goues, Graham Neubig, Bogdan Vasilescu
Reverse Engineering: understanding program behavior without access to its source code.

Computer cables snake across the floor. Cryptic flowcharts are scrawled across various whiteboards adorning the walls. A life-size Batman doll stands in the hall. This office might seem no different than any other geeky workplace, but in fact it’s the front line of a war—a cyberwar, where most battles play out not in remote jungles or deserts but in suburban office parks like this one. As a senior researcher for Kaspersky Lab, a leading computer security firm...
Reverse engineering is very difficult.
A better choice: Decompiler
A better choice: Decompiler
A better choice: Decompiler
A better choice: Decompiler

```c
v8 = v7 + 1;
switch ( __ROR1__(*v6 - 99, 1) )
{
    case 0:
        if ( v6[1] != 111 )
            goto LABEL_46;
        if ( v6[2] != 110 )
            goto LABEL_46;
        if ( v6[3] != 118 )
            goto LABEL_46;
        v9 = v6[4];
        if ( v9 )
        {
            if ( v9 != 61 )
                goto LABEL_46;
        }
```
Compilation Loses Information

Comments:

/* This is the functionality
 * you're looking for! */

Loop Constructs:

while (x < 100) {...}

for (i = 0; i < 100; ++i) {...}

Variable Names:

int width, length;
double volume;
char *user_id;

User-Defined Types:

typedef struct {
    int x;
    int y;
} point;
Comments:

/* This is the functionality
 * you're looking for! */

Loop Constructs:

while (x < 100) {...}
for (i = 0; i < 100; ++i) {...}

Variable Names:

int width, length;
double volume;
char *user_id;

User-Defined Types:

typedef struct {
    int x;
    int y;
} point;
Names are helpful for readability

```c
v8 = v7 + 1;
switch ( __ROR1__(*v6 - 99, 1) )
{
    case 0:
        if ( v6[1] != 111 )
            goto LABEL_46;
        if ( v6[2] != 110 )
            goto LABEL_46;
        if ( v6[3] != 118 )
            goto LABEL_46;
        v9 = v6[4];
        if ( v9 )
        {
            if ( v9 != 61 )
                goto LABEL_46;
        }
```
Names are helpful for readability

```c
+= 1;
switch (__ROR1__ (*v6 - 99, 1)) {
    case 0:
        if (v6.1 != 111)
            goto LABEL_46;
        if (v6.2 != 110)
            goto LABEL_46;
        if (v6.3 != 118)
            goto LABEL_46;
        if (v4]
            if (v6 != 61)
                goto LABEL_46;
```
Decompilers don't Recover Code!

State of the practice in variable name recovery ca. 2018
**Goal:** automatically recover meaningful variable names to support reverse engineering.
Natural languages are complex

Tiger, Tiger
burning bright
In the forests
of the night..
..but natural utterances are simple & repetitive

TIGER!! RUN!!!
English, தமிழ், German

Can be rich, powerful, expressive

..but “in nature” is mostly simple, repetitive, boring

Statistical models

Google, ASSISTANT, Cortana
The “naturalness of software” thesis

Programming languages are complex...

...but natural programs are simple & repetitive.

and this, too, can be exploited.

[Hindle et al, 2011]
Plan: adapt natural language translation to rename variables in decompiled code.

• The noisy channel translation model:

\[ p(e) \quad \text{language model} \]
\[ p(f \mid e) \quad \text{channel model} \]

Goal: recover \( e \)

\[ f \quad \text{distorted message} \]

• Dire:

Binary \( \rightarrow \) Decompiler \( \rightarrow \) Token Stream \( \rightarrow \) DIRE \( \rightarrow \) Variable Names

Carnegie Mellon University
School of Computer Science
Part 1: Training.
Difficulty: Decompilation Changes Structure

Original Source
```c
#include <stdio.h>
int main() {
    int cur = 0;
    while (cur <= 9) {
        printf("%d\n", cur);
        ++cur;
    }
    return 0;
}
```

Decompiled Code
```c
#include <stdio.h>
int main() {
    int v1 = 0;
    int v2;
    for (v2 = 0; v2 < 10; ++v2)
        printf("%d\n", v2);
    return v1;
}
```
Difficulty: Decompilation Changes Structure

- Different line count.
Difficulty: Decompilation Changes Structure

Original Source
```c
#include <stdio.h>
int main() {
    int cur = 0;
    while (cur <= 9) {
        printf("%d\n", cur);
        ++cur;
    }
    return 0;
}
```

Decompiled Code
```c
#include <stdio.h>
int main() {
    int v1 = 0;
    int v2;
    for (v2 = 0; v2 < 10; ++v2) {
        printf("%d\n", v2);
    }
    return v1;
}
```

- Different line count.
- Different numbers of variables.
Difficulty: Decompilation Changes Structure

- Different line count.
- Different numbers of variables.
- Different types of loops.
Part 1: Training data

- Key insight: instruction offsets and operations on variables do not change between decompilation with and without debug information.
Part 1: Training data

GitHub

gcc -g
Part 2: Model.
char * mystrocopy(char *VAR1, char *VAR2){
char *result;
if (VAR1 && VAR2)
  result = strcopy(VAR1, VAR2);
else
  result = OLL;
return result;
}
Dire has high exact-match accuracy, and each component contributes uniquely

<table>
<thead>
<tr>
<th></th>
<th>DIRE Acc.</th>
<th>DIRE CER</th>
<th>Lexical Enc. Acc.</th>
<th>Lexical Enc. CER</th>
<th>Structural Enc. Acc.</th>
<th>Structural Enc. CER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall</td>
<td>74.3</td>
<td>28.3</td>
<td>72.9</td>
<td>28.5</td>
<td>64.6</td>
<td>37.5</td>
</tr>
<tr>
<td>Body in Train</td>
<td>85.5</td>
<td>16.1</td>
<td>84.3</td>
<td>16.3</td>
<td>75.7</td>
<td>25.5</td>
</tr>
<tr>
<td>Body not in Train</td>
<td>35.3</td>
<td>67.2</td>
<td>33.5</td>
<td>67.7</td>
<td>26.3</td>
<td>76.1</td>
</tr>
</tbody>
</table>
This work...

Comments:

/* This is the functionality
 * you're looking for! */

Loop Constructs:

while (x < 100) {...}
for (i = 0; i < 100; ++i) {...}

Variable Names:

int width, length;
double volume;
char *user_id;

User-Defined Types:

typedef struct {
    int x;
    int y;
} point;
Ongoing work...

Comments:

/* This is the functionality * you're looking for! */

Loop Constructs:

while (x < 100) {...}
for (i = 0; i < 100; ++i) {...}

Variable Names:

int width, length;
double volume;
char *user_id;

User-Defined Types:

typedef struct {
    int x;
    int y;
} point;

type point;