Similarity Metric Method for Binary Basic Blocks of Cross-Instruction Set Architecture

Xiaochuan Zhang
zhangxiaochuan@outlook.com

Artificial Intelligence Research Center, National Innovation Institute of Defense Technology, Beijing, China
• **Background**

Binary program similarity metric can be used in:

- **malware classification**
- **vulnerability detection**
- **authorship analysis**

The similarity between basic blocks is the basis.
Two step of basic block similarity metric

**Basic Block Embedding**

- `sub sp, sp, #72`
- `ldr r7, [r11, #12]`
- `ldr r8, [r11, #8]`
- `ldr r0, .LCPI0_0`

**Similarity Calculation**

- `movq %rdx, %r14`
- `movq %rsi, %r15`
- `movq %rdi, %rbx`
- `movabsq $.L0, %rdi`

- [0.24, 0.37, ..., 0.93]
- [0.56, 0.74, ..., 0.31]

**Similarity Score**

- [0, 1]
• **Background**

**Type of methods**

- **Basic block embedding**
  - manually
  - each dimension corresponds to a manually selected static feature [1-3]

- **Static word representation based methods** [4-7]

- **INNEREYE-BB, an RNN based method** [8]

---

INNEREYE-BB [1]

$$h_t = F(s_t, h_{t-1})$$

<table>
<thead>
<tr>
<th>Token type</th>
<th>x86</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic block label</td>
<td>77.68%</td>
<td>61.23%</td>
</tr>
<tr>
<td>function name</td>
<td>3.71%</td>
<td>12.58%</td>
</tr>
<tr>
<td>others</td>
<td>18.60%</td>
<td>26.19%</td>
</tr>
</tbody>
</table>

• Methodology & Implementation

Idealized Solution (based on PERFECT TRANSLATION assumption)
Practical Solution

Methodology & Implementation

- x86-encoder pre-training
- x86-encoder fine-tuning
- ARM-encoder training
x86-encoder pre-training

- data: x86-ARM basic block pairs
- NMT model: Transformer [1], other NMT models also work
- Optimization goal: minimize the translation loss

\[ L = - \sum_{k=1}^{m} \sum_{j=1}^{|V_{ARM}|} \hat{y}_{kj} \log(y_{kj}) \]

• Methodology & Implementation

ARM-encoder training & x86-encoder fine-tuning

- data: basic block triplets, \{anchor, positive, negative\}
- Optimization goal: minimize the margin-based triplet loss

\[
L = \max\{D(E_1, E_2) - D(E_1, E_3) + \gamma, 0\}
\]
Methodology & Implementation

Mixed negative sampling

- 33% Random Negatives
- 67% Hard Negatives

Hard Negatives:
Similar but not equivalent to anchor
Hard negative sampling: if anchor is a x86 basic block

anchor(x86)

rand_x86_1

rand_x86_2

......

rand_x86_n

pretrained x86-encoder

$E_{anchor}$

$E_1$

$D_1$

$E_2$

$D_2$

$E_n$

$D_n$

rand_x86_t

rand_ARM_t
Methodology & Implementation

Similarity Metric

\[ \text{Sim}(B_1, B_2) = \exp\left(-\frac{D(E_1, E_2)}{d}\right) \]
### Experiment & Result

#### Setup

- **prototype:** MIRROR
  
  [https://github.com/zhangxiaochuan/MIRROR](https://github.com/zhangxiaochuan/MIRROR)

- **Dataset:** MISA, **1,122,171** semantically equivalent x86-ARM basic block pairs
  
  [https://drive.google.com/file/d/1krJbsfu6EsLhF86QAUVxVRQjbkfWx7ZF/view](https://drive.google.com/file/d/1krJbsfu6EsLhF86QAUVxVRQjbkfWx7ZF/view)

<table>
<thead>
<tr>
<th>Project</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binutils</td>
<td>2.30</td>
<td>collection of binary tools</td>
</tr>
<tr>
<td>Coreutils</td>
<td>8.29</td>
<td>GNU core utilities</td>
</tr>
<tr>
<td>FFmpeg</td>
<td>n3.2.13</td>
<td>collection of multimedia process tools</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>1.1.1b</td>
<td>security protocols and cryptographic library</td>
</tr>
<tr>
<td>Redis</td>
<td>5.0.5</td>
<td>key-value database</td>
</tr>
</tbody>
</table>
## Experiment & Result

### Comparison with Baseline

<table>
<thead>
<tr>
<th>Model</th>
<th>x86-ARM</th>
<th>ARM-x86</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P@1  P@3  P@10</td>
<td>P@1  P@3  P@10</td>
</tr>
<tr>
<td>INNEREYE-BB</td>
<td>51.0% 66.6% 77.2%</td>
<td>32.8% 54.8% 79.5%</td>
</tr>
<tr>
<td>MIRROR (MISA\textsubscript{Triplet_Base})</td>
<td>64.0% 77.2% 85.7%</td>
<td>58.7% 73.8% 83.1%</td>
</tr>
<tr>
<td>MIRROR (MISA\textsubscript{Triplet_Large})</td>
<td>77.4% 88.7% 94.9%</td>
<td>74.2% 87.2% 94.1%</td>
</tr>
</tbody>
</table>

* Higher is better
## Experiment & Result

### Evaluation of negative sampling methods

<table>
<thead>
<tr>
<th>Negative Samples</th>
<th>x86-ARM</th>
<th>ARM-x86</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P@1</td>
<td>P@3</td>
</tr>
<tr>
<td>None</td>
<td>49.6%</td>
<td>56.2%</td>
</tr>
<tr>
<td>Random only</td>
<td>62.2%</td>
<td>79.2%</td>
</tr>
<tr>
<td>Hard only</td>
<td>60.0%</td>
<td>74.6%</td>
</tr>
<tr>
<td>Mixed (ours)</td>
<td>69.0%</td>
<td>83.8%</td>
</tr>
</tbody>
</table>

* Higher is better
Effectiveness of pre-training

The pre-training phase seems redundant?
# Experiment & Result

## Effectiveness of pre-training

<table>
<thead>
<tr>
<th>Setting</th>
<th>x86-ARM</th>
<th>ARM-x86</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P@1</td>
<td>P@3</td>
</tr>
<tr>
<td>Pre-train</td>
<td>Negative</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>Random</td>
<td>58.2%</td>
</tr>
<tr>
<td>True</td>
<td>Random</td>
<td>62.2%</td>
</tr>
<tr>
<td>False</td>
<td>Mixed</td>
<td>64.4%</td>
</tr>
<tr>
<td>True</td>
<td>Mixed</td>
<td>69.0%</td>
</tr>
</tbody>
</table>

* Higher is better
• Experiment & Result

Visualization

![Visualization](image-url)
Thanks!

zhangxiaochuan@outlook.com