

Специализация машинного кода

Юрий Кравченко

руководитель Березун Даниил Андреевич

СПБАУ

30 мая 2018 г.

Специализация

Традиционное исполнение программы

$$\llbracket p \rrbracket_L[in_1, in_2, \dots] = out$$

Специализатор

Программу *spec* назовём специализатором, если

$$\begin{array}{lll} \llbracket spec \rrbracket_{L_2} [p, in_1] & = & p_{spec} \\ \llbracket p_{spec} \rrbracket_{L_1} [in_2, \dots] & = & out \end{array}$$

Цель специализации

source — программа на языке S

int — интерпретатор для языка S на языке L

Проекция Футамуры

[1973]

I $\llbracket spec \rrbracket_L [int, source] = target$

II $\llbracket spec \rrbracket_L [spec, int] = comp$

III $\llbracket spec \rrbracket_L [spec, spec] = cogen$

Вывод

$Interpreter \xrightarrow{spec} Compiler$

В чём подвох?

- ▶ Компилятор в язык реализации интерпретатора

$$\textit{Interpreter}_{\mathbf{L}}^S \xrightarrow{\textit{spec}_{\mathbf{L}}^{\mathbf{L}}} \textit{Compiler}_{\mathbf{L}}^{S \rightarrow \mathbf{L}}$$

Это основная проблема

- ▶ Апостериорный факт: реализовывать специализаторы сложно (~ как компиляторы)

Релевантные исследования

Все текущие исследования имеют одну из двух проблем

- ▶ Искусственный язык
- ▶ Нет возможности самоприменения

Partial Evaluation of Machine Code

[2015]

- ▶ Подмножество IA-32
- ▶ Использование сторонней закрытой библиотеки
- ▶ Написан на Java \Rightarrow нельзя самоприменить

Идея

- ▶ Специализатор для машинного кода

$$Interpreter_{\text{ASM}}^S \xrightarrow{\text{spec}_{\text{ASM}}^{\text{ASM}}} Compiler^{S \rightarrow \text{ASM}}$$

- ▶ Как получить $Interpreter_{\text{ASM}}^S$?

$$\llbracket gcc \rrbracket [Interpreter_C^S] = Interpreter_{\text{ASM}}^S$$

- ▶ Как получить $\text{spec}_{\text{ASM}}^{\text{ASM}}$?

$$\llbracket gcc \rrbracket [\text{spec}_C^{\text{ASM}}] = \text{spec}_{\text{ASM}}^{\text{ASM}}$$

- ▶ Как получить $\text{spec}_C^{\text{ASM}}$?

Цель и задачи

Цель

- ▶ Исследование возможностей специализатора машинного кода

Задачи

- ▶ Изучить существующие подходы и алгоритмы специализации для низкоуровневых языков программирования
- ▶ Разработать архитектуру специализатора с учётом рассмотренных подходов и особенностей языка специализации
- ▶ Добавление возможностей специализатора, необходимых для самоприменения
- ▶ Исследование возможностей полученного специализатора в том числе на кмп тесте и каком-то там ещё

Binding time analysis

динамический статический

```
1 int pow(int a, int b) {  
2     int res = 1;  
3     while (b > 0) {  
4         res = res * a;  
5         b = b - 1;  
6     }  
7     return a;  
8 }
```

\xrightarrow{BTA}

```
1 int pow(int a, int b) {  
2     int res = 1;  
3     while (b > 0) {  
4         res = res * a;  
5         b = b - 1;  
6     }  
7     return a;  
8 }
```


Основные проблемы

Проблема : ВТА делает регистры динамическими

```
1 //esi динамический
2 mov esi eax
3 //теперь eax динамический
4 mov 4 eax
5 //специализатор не знает значение eax
```

Решение : Online специализация

Проблема : Комплексные инструкции (push)

```
1 //esi динамический
2 push esi
3 //теперь esp динамический
4 push 4
5 pop eax
6 //специализатор не знает значение eax
```

Решение : ВТА разделяет инструкцию на простые

Основные проблемы

Проблема : Статические адреса (например стек) становятся константами

```
1 //%esi динамический
2 push %esi
3 //специализируется в
4 mov %esi (268123094)
5 //адрес может меняться между запусками %eax
```

Решение : Символьные вычисления

```
1 //%esi динамический
2 push %esi
3 //специализируется в
4 mov %esi -48(0)
```

Основные проблемы

Проблема : Не всегда можно сделать lifting

```
1 lea -72(0) %ebx
2 add 10 %ebx
3 //%edx динамический
4 mov %ebx 4(%edx)
5 //после специализации получим
6 mov -62(0) 4(%ebx)
7 //запись из памяти в память запрещена
```

Решение : Генерация дополнительных инструкций

```
1 lea -72(0) %ebx
2 add 10 %ebx
3 //%edx динамический
4 mov %ebx 4(%edx)
5 //после специализации получим
6 lea -62(0) %ebx // сгенерированная инструкция
7 mov %ebx 4(%ebx)
```

KMP test

```

1  int kmp(char* p, char* d, char* frow1, char* frow2)
2  {
3      char* pp = p;
4      char* f = frow1;
5      char* ff = frow1;
6      char* rmg = frow2;
7      char* rmq = frow2;
8      while (1) {
9          if (p[0] == 0) {
10             return 1;
11          }
12          else if (f == f0) {
13             if (number3(p[0], rmq, rmq0)) {
14                 if (ff == f0) {
15                     p = pp;
16                     d++;
17                     ff = f0;
18                     rmq = rmq0;
19                     continue;
20                 }
21                 else {
22                     p = pp;
23                     ff++;
24                     f = ff;
25                     continue;
26                 }
27             }
28             else if (rmq == rmq0 && d[0] == 0) {
29                 return 0;
30             }
31             else if (p[0] == d[0]) {
32                 char* ptr = ff;
33                 while (ptr != f0) {
34                     ptr[0] = ptr[0];
35                     ptr++;
36                 }
37                 ptr[0] = p[0];
38                 f++;
39                 d++;
40                 rmq = rmq0;
41                 continue;
42             }
43             else if (ff == f0) {
44                 p = pp;
45                 d++;
46                 f = f0;
47                 ff = f0;
48                 rmq = rmq0;
49                 continue;
50             }
51             else {
52                 rmq[0] = p[0];
53                 p = pp;
54                 ff++;
55                 f = ff;
56                 continue;
57             }
58             }
59             else if (p[0] == f[0]) {
60                 p++;
61                 ff++;
62                 continue;
63             }
64             else {
65                 p = pp;
66                 ff++;
67                 f = ff;
68                 continue;
69             }
70             }
71             }
72             }
73             }

```

[[spec]]_{ASM}[kmp,"a"]

```

1  Start block -633763
2  mov89 %rsi -88(0)
3  mov8b -88(0) %rax
4  movb6 0(rax) %rax
5  test %al %al
6  cjump 0x85 to 565830
7  premov 0 , %rax
8  ret
9
10 Start block 565830
11 mov8b -88(0) %rax
12 movb6 0(rax) %rax
13 cmp39 97 %rax
14 cjump 0x85 to 925494
15 add83 -88(0) 1
16 premov 1 , %rax
17 ret
18
19 Start block 925494
20 add83 -88(0) 1
21 mov8b -88(0) %rax
22 movb6 0(rax) %rax
23 test %al %al
24 cjump 0x85 to 565830
25 premov 0 , %rax
26 ret

```

Специализация интерпретатора

```
int foo(int a, int b){  
    int c;  
    c = a;  
    c += b;  
    return c;  
}
```

$\llbracket spec \rrbracket_{ASM} [interpreter, foo]$ \rightarrow

```
1 Start block -697046  
2 mov89 %rsi -72(0)  
3 call malloc  
4 mov8b -72(0) %rax  
5 mov8b 0(rax) %rax  
6 mov89 %rax 0(2)  
7 add83 -72(0) 4  
8 mov8b -72(0) %rax  
9 mov8b 0(rax) %rax  
10 mov89 %rax 16(2)  
11 add83 -72(0) 4  
12 mov8b 0(2) %rax  
13 mov89 %rax -100(0)  
14 mov8b -100(0) %rax  
15 mov89 %rax -28(0)  
16 mov8b -28(0) %rax  
17 mov89 %rax 32(2)  
18 mov8b 16(2) %rax  
19 mov89 %rax -100(0)  
20 mov8b -100(0) %rax  
21 mov89 %rax -28(0)  
22 mov8b 32(2) %rcx  
23 mov8b -28(0) %rdx  
24 add01 %rdx %rcx  
25 mov89 %rdx 32(2)  
26 mov8b 32(2) %rax  
27 mov89 %rax -100(0)  
28 mov8b -100(0) %rax  
29 ret
```

Конец

<https://github.com/XJIE6/spec>