Специализация машинного кода

Юрий Кравченко

руководитель Березун Даниил Андреевич

СПбАУ

23 мая 2018 г.

Специализация

Традиционное исполнение программы

$$[\![p]\!]_L[in_1, in_2, \ldots] = out$$

Специализатор

Программу *spec* назовём специализатором, если

$$\begin{array}{lll} [\![spec]\!]_{L_2} & [p, in_1] & = & p_{spec} \\ [\![p_{spec}]\!]_{L_1} & [in_2, \ldots] & = & out \end{array}$$

Цель специализации

source — программа на языке S
int — интерпретатор для языка S на языке L

Проекции Футамуры [1973]

$$\begin{array}{lllll} I & [\![spec]\!]_L & [int, source] & = & target \\ II & [\![spec]\!]_L & [spec, int] & = & comp \\ III & [\![spec]\!]_L & [spec, spec] & = & cogen \end{array}$$

Вывод

$$Interpreter \xrightarrow{spec} Compiler$$

В чём подвох?

- Компилятор в язык реализации интерпретатора

$$Interpreter_{\mathbf{L}}^{S} \xrightarrow{spec_{\mathbf{L}}^{\mathbf{L}}} Compiler_{\mathbf{L}}^{S \to \mathbf{L}}$$

Это основная проблема

- Апостериорный факт: реализовывать специализаторы сложно ($\sim$ 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

$$Interpreter^{S}_{\mathbf{ASM}} \xrightarrow{spec^{\mathbf{ASM}}_{\mathbf{ASM}}} Compiler^{S \to \mathbf{ASM}}$$

- Как получить $Interpreter^{S}_{\mathbf{ASM}}$?

$$[\![gcc]\!][Interpreter^{S}_{C}] = Interpreter^{S}_{\mathbf{ASM}}$$

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

$$[\![gcc]\!][spec^{\mathbf{ASM}}_{C}] = spec^{\mathbf{ASM}}_{\mathbf{ASM}}$$

- Как получить $spec^{\mathbf{ASM}}_{C}$?

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 рассмотреных подходов и особенностей языка специалиации
- Добавление возможностей специализатора, необходимых для самоприменения
- Исследование возможностей полученного специазатора в том числе на кмп тесте и каком-то там ещё

Алгоритм

```
1  typedef struct _info {
2      char is_dynamic;
3      int mem;
4  } info;
5
6  struct _state {
7      long long regs[17];
8      info info_regs[17];
9      char flags[64];
10     info info_flags;
11     char** mem;
12     info** info_mem;
13     int mem_len;
14     int mem_mem_len[100];
15     struct _state* next;
16     long long hash;
17 };
```

Дополнительные возможности

- Расширенное подмножество инструкций: add, cmp, test, imul, jmp, jcc, mov, lea, pop, leave, push, ret, sub, call, ...
- Работа с выделением памятью
- Мультипроцедурная специализация
- Поддержка статических данных

KMP тест

```
1  int kmp(char* p, char* d, char* free1, char* free2)
                {
2      char* pp = p;
3      char* f = free1;
4      char* ff = free1;
5      char* neg = free2;
6      char* f0 = free1;
7      char* neg0 = free2;
8      while (1) {
9          if (p[0] == 0) {
10             return 1;
11         }
12         else if (f == f0) {
13             if (member3(p[0], neg, neg0)) {
14                 if (ff == f0) {
15                     p = pp;
16                     d++;
17                     ff = f0;
18                     neg = neg0;
19                     continue;
20                 }
21                 else {
22                     p = pp;
23                     ff++;
24                     f = ff;
25                     continue;
26                 }
27             }
28             else if (neg == neg0 && d[0] == 0) {
29                 return 0;
30             }
31             else if (p[0] == d[0]) {
32                 char* ptr = ff;
33                 while (ptr != f0) {
34                     ptr[ -1] = ptr[0];
35                     ptr++;
36                 }
37                 ptr[ -1] = p[0];
38                 ff--;
39                 p++;
40                 d++;
41                 neg = neg0;
42                 continue;
43             }
44             else if (ff == f0) {
45                 p = pp;
46                 d++;
47                 f = f0;
48                 ff = f0;
49                 neg = neg0;
50                 continue;
51             }
52             else {
53                 neg--;
54                 neg[0] = p[0];
55                 p = pp;
56                 ff++;
57                 f = ff;
58                 continue;
59             }
60         }
61         else if (p[0] == f[0]) {
62             p++;
63             f++;
64             continue;
65         }
66         else {
67             p = pp;
68             ff++;
69             f = ff;
70             continue;
71         }
72     }
73 }
```

$$[\![spec]\!]_{ASM}[kmp,"a"] \longrightarrow$$

```
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
```

Интерпретация

Результаты специализации интерпретатора

Результаты

- Выбрана и реализована модель специализатора, отвечающая требованиям задачи
- Добавлены необходимые для самоприменения возможности
- Проведено исследование возможностей специализатора

Конец

https://github.com/XJIE6/spec