Специализация машинного кода

Юрий Кравченко

руководитель Березун Даниил Андреевич

СПбАУ

18 июня 2018 г.

Специализация

Традиционное исполнение программы

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

Специализатор

Программу *spec* назовём специализатором, если

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

динамический статический

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

$$Interpreter \xrightarrow{spec} Compiler$$

Почему не используется

- 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)

Идея

- Специализатор для машинного кода

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

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

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

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

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

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

Цель и задачи

Цель

Исследование возможности и особенностей специализации машинного код

Задачи

- Изучить существующие подходы и алгоритмы специализации низкоуровневых языков программирования
- Исследовать особенности специализации машинного кода
- Предложить алгоритм специализации машинного кода
- Реализовать прототип специализатора для машинного кода на языке C, основанного на предложенном алгоритме специализации
- Произвести апробацию возможностей полученного специализатора

Существующие подходы

Partial evaluation and automatic program generation, Neil D. Jones, Carsten K. Gomard, Peter Sestoft, 1994.

- Специализатор для языка Flow Chart
- Flow Chart структурно похож на машинный код
- Специализатор является самоприменимым

Partial evaluation of machine code, Srinivasan Venkatesh, Reps Thomas, 2015.

- Специализатор для подмножества IA-32
- Использованы сложные техники
- Реализован на Java $\Rightarrow$ нельзя самоприменить

Binding time analisys(BTA)

Анализ времени исполнения классифицирует переменные/инструкции на статические и динамические

```
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 }
```

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 }
```

Особенности специализации машинного кода

Проблема : BTA 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 : BTA разделяет инструкцию на простые

Особенности машинного кода

Проблема : Специализация констант времени исполнения

```
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)
```

Алгоритм

```
 1 Input: program, program_input
 2 state ← generate_state(program, program_input)
 3 queue ← state
 4 while(!queue.empty()) {
 5     st ← queue.pop()
 6     while(!st.end()){
 7         cmd ← st.read_cmd()
 8         params ← st.read_params()
 9         if (BTA(params) == static) {
10             st ← st.eval(cmd, params)
11         }
12         else {
13             st ← st.dynamic_eval(cmd, params)
14             reduced ← st.reduce(cmd, params)
15             result ← result ++ reduced
16         }
17     }
18 }
19 Output: result
```

Реализация прототипа

- Реализован на языке C
- Выбран набор ключевых тестов и реализован фукционал для прохождения данных тестов
- Специализатор способен обрабатывать основные машинные инструкции: mov, lea, add, sub, imul, cmp, test, call, ret, push, pop, jmp, jcc
- Результатом работы специализатора является программа на языке, схожим с ассемблером. Язык позволяет наглядно представлять результаты специализации
- Генерация машинного кода реализована для узкого набора инструкций

KMP тест

- $[\![spec]\!]_{ASM}[kmp, "ababac"]$
- Для прохождения данного теста необходимо преобразовать наивный алгоритм поиска подстроки в строке в оптимальный (например, KMP).
- Искомая строка является статической
- Строка, в которой происходит поиск, является динамической
- Тест пройден

Специализация интерпретатора

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

$$[\![spec]\!]_{ASM}[interpreter, add] \longrightarrow$$

```
Start block -697046
mov89 %rsi -72(0)
call malloc
mov8b -72(0) %rax
mov8b 0(rax) %rax
mov89 %rax 0(2)
add83 -72(0) 4
mov8b -72(0) %rax
mov8b 0(rax) %rax
mov89 %rax 16(2)
add83 -72(0) 4
mov8b 0(2) %rax
mov89 %rax -100(0)
mov8b -100(0) %rax
mov89 %rax -28(0)
mov8b -28(0) %rax
mov89 %rax 32(2)
mov8b 16(2) %rax
mov89 %rax -100(0)
mov8b -100(0) %rax
mov89 %rax -28(0)
mov8b 32(2) %rcx
mov8b -28(0) %rdx
add01 %rcx %rdx
mov89 %rdx 32(2)
mov8b 32(2) %rax
mov89 %rax -100(0)
mov8b -100(0) %rax
ret
```

- Цель данного теста - оценить количество лишних инструкций
- Результат специализации отвечает требуемой функциональности. Лишние инструкции присутствуют

Итоги

- Исследованы различные подходы специализации низкоуровневых языков
- Исследованы особенности специализации машинного кода
- Предложен алгоритм специализации машинного кода
- Реализован прототип специализатора на языке C и произведена его апробация