Специализация машинного кода

Юрий Кравченко

руководитель Березун Даниил Андреевич

Лаборатория языковых инструментов

15 декабря 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 $spec^{\mathbf{ASM}}_{\mathrm{C}}$?

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

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

Дипломная работа

- 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
- Аппробирована первая проекция проекция футамуры для маленького языка

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)
```

А что сейчас?

- есть функции
- for, while, if
- нет структур данных
- один тип - int

```
1  fun foo (a, b)
2  begin
3      if (b == 0)
4      then
5          return 1
6      else
7          c := foo(a, b - 1);
8          return a * c
9      fi
10 end
11
12 a := 7;
13 b := 5;
14 return foo(a, b)
```

Межпроцедурность

```
1 некое состояние программы
2 rdx - динамический
3 mov rdx rcx
4 call foo
5 какое состояние?
```

```
1 int foo(int a) {
2     if (a > 0) {
3         return 1;
4     }
5     return 0;
6 }
```

```
1 int bar(int a, int b) {
2     if (b == 0) {
3         return 1;
4     }
5     return bar(a, b - 1) * a;
6 }
```

Первая проекция

$$[\![spec]\!]_L[int, source] = target$$

```
1 return x + 2
```

```
1  Start block 1
2  call 2
3  mov8b 12(2)  %rax
4  ret
5
6  Start block 2
7  call 3
8  mov89 %rax  %rdx
9  mov89 %rdx  12(2)
10 premov 0(2) %rax
11 ret
12
13 Start block 3
14 call 4
15 mov89 %rax  -80(0)
16 call 5
17 mov8b -80(0)  %rax
18 add01 2  %rax
19 ret
20
21 Start block 4
22 mov8b 108(2)  %rax
23 ret
24
25 Start block 5
26 premov 2 %rax
27 ret
```

Планы на будущее

Вторая проекция

$$[\![spec]\!]_L[spec, int] = comp$$

- Модифицировать специализатор
- Раздвоить специализатор
- Сравнить с компилятором и первой проекцией