Компилятор GHC языка Haskell: теория языков программирования в работе

Лекция 5. STG-машина и вычисление выражений Core

В. Н. Брагилевский

1 апреля 2018 г.

Computer Science клуб (Санкт-Петербург)

Институт математики, механики и компьютерных наук
имени И. И. Воровича, Южный федеральный университет (Ростов-на-Дону)

Что происходит с Core после оптимизации (CorePrep)

- Насыщение (saturation) конструкторов и примитивных операций аргументами:
  пусть g должно иметь два аргумента, тогда
  `f = /\a -> g (h 3)` =>
  `f = /\a -> \y -> g (h 3) y`!
- Конвертация в A-нормальную форму (аргументы всегда переменные, каждое подвыражение имеет имя):

```
f = /\a -> g (h 3) =>
f = /\a -> \y ->
let s = h 3 in g s y
```

- Для строгих аргументов можно использовать `case`:
  `f E => case E of x -> f x`
- Все unboxed-значения в `let` конвертируются в `case`
- λ-абстракции на значениях могут быть только в правой части равенства в `let`

Язык STG

STG — это функциональный язык, похожий на Core, но с отличиями:

- Он нетипизирован (но есть информация о типах представления).
- Он находится в A-нормальной форме.

STG-выражения

1. Атомы (литералы и переменные)
2. `let`-определения, где в правой части равенств допускаются функции в виде явных лямбд, ненасыщенные применения, насыщенные конструкторы, задумки для всех остальных видов выражений
3. Насыщенные применения примитивных функций
4. Потенциально ненасыщенные применения
5. case-выражения, в ветках которых выражения STG

Пример кода Core

```
f a b = a + b
main = print $ f 2 3
```

```
main :: IO ()
[GblId, Str=DmdType]
main =
 print
  @ Integer
  GHC.Show.$fShowInteger
  (+ @ Integer
       GHC.Num.$fNumInteger
       2 3)
```

Фрагмент кода STG

```
sat_s10R :: GHC.Integer.Type.Integer
[LclId, Str=DmdType] =
    \u srt:SRT:[rp0 :-> GHC.Num.$fNumInteger] []
        let {
          sat_s10Q [Occ=Once] ::
                        GHC.Integer.Type.Integer
          [LclId, Str=DmdType] =
          NO_CCS GHC.Integer.Type.S#! [3#]; } in
        let {
          sat_s10P [Occ=Once] ::
                        GHC.Integer.Type.Integer
          [LclId, Str=DmdType] =
              NO_CCS GHC.Integer.Type.S#! [2#];
            } in  GHC.Num.+ GHC.Num.$fNumInteger
                            sat_s10P sat_s10Q;
```

```
Main.main :: GHC.Types.IO ()
[GblId, Str=DmdType] =
  \u srt:SRT:[0B :-> System.IO.print,
                rLs :-> GHC.Show.$fShowInteger,
                s10R :-> sat_s10R] []
  System.IO.print GHC.Show.$fShowInteger sat_s10R;
```