Etape prevođenja Haskela do mašinskog jezika

Miroslav Mišljenović, Marija Mijailović
Filip Lazić, Nemanja Antić

Matematički fakultet, Beograd

21. Maj 2018.

Sadržaj

Uvod

- Funkcionalna paradigma
  - Matematičke funkcije
  - Elimininacija bočnih efekata
  - Dokazivanje korektnosti programa

- Džon Bakus
  - Tjuringova nagrada za doprinos u razvoju *FORTRAN-a*

Haskel

- Funkcije višeg reda
- Ne-striktna semantika
- Statički polimorfizam
- Korisnički definisani algebarski tipovi
- Prepoznavanje šablona
- Rad sa listama

Glazgov Haskel Kompajler (GHC)

- Sastoji se iz tri faze:
  - Frontend - pretvaranje u Core jezik
  - Middle end - dodatne optimizacije
  - Backend - STG mašina

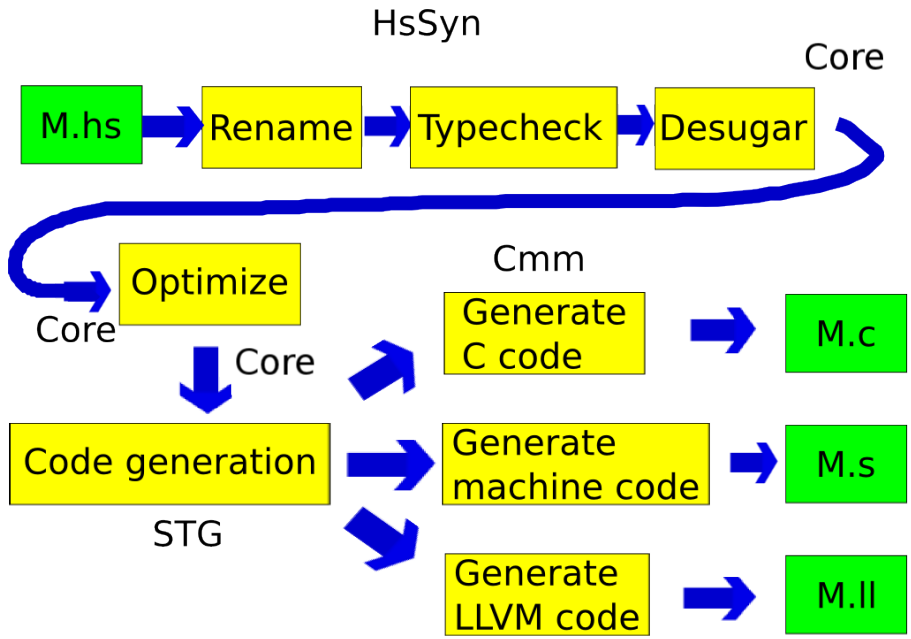

Figure 1: Razvojne etape pravljenja izvršnog kôda

Frontend

- Parsiranje (eng. *Parser*)
- Promena imena (eng. *Rename*)
- Provera tipa (eng. *Typecheck*)
- Prečišćavanje (eng. *Desugaring*)

Parsiranje i promena imena

- Parsiranje:
  - Šabloni (eng. *Pattern*)
  - Infiksni operatori
  - Poruke o greškama
- Promena imena:
  - Zamena *RdrNames* sa *Names*
  - Analiza zahteva za uzajmno rekurzivne grupe deklaracija
  - Veliki broj provera grešaka

Promena imena

Primer zamene *RdrNames* sa *Names*:

module K where $f\ x = \text{True}$
module N where import K
module M where import $N(f)$ as $Q\ f = (f, M.f, Q.f, \backslash f \to f)$

Rezultat preimenovanja modula M je:

$M.f = (M.f, M.f, K.f, \backslash f_{22} \to f_{22})$

Provera tipa i prečišćavanje

- Provera tipa:
  - Provera programa
  - Definisanje apstraktne sintakse
  - Interfejs typechecker-a
  - Funkcije tcRnModule i tcRnModuleTcRnM

- Prečišćavanje:
  - Prevod iz HsSyn tipa u GHC-ov međujezik CoreSyn

Core jezik

- Pravi jednostavan tipizirani lambda račun
- Jednostavan lenji funkcionalni jezik Asembler funkcionalnog jezika
- Sastoji od nekoliko elemenata:
  - varijable
  - literali
  - let
  - case
  - lambda apstrakcije
  - aplikacije

Middle end i umetanje

- Dodatna optimizacija:
  - umetanje kôda (eng. *inlining*)
    - ubrzanje od 20 – 40% kod funkcionalnih jezika
    - ubrzanje od 10 - 15% kod imperativnih jezika
  - eliminacija zajedničkih podizraza (eng. *common subexpression elimination*)

Umetanje

Samoumetanje (eng. *inlining itself*)

$$\text{let } \{f = \backslash x- > x * 3\} \text{ in } f(a + b) - c$$
$$\implies [inline\ f]$$
$$\text{let } \{f = \backslash x- > x * 3\} \text{ in } (\backslash x- > x * 3)(a + b) - c$$

Eliminacija mrtvog kôda (eng. *dead code elimination*)

$$\text{let } \{f = \backslash x- > x * 3\} \text{ in } (\backslash x- > x * 3)(a + b) - c$$
$$\implies [dead\ f]$$
$$(\backslash x- > x * 3)(a + b) - c$$

β redukcija (eng. *beta reduction*)

$$(\backslash x- > x * 3)(a + b) - c \implies [beta]\ (let\ \{x = a + b\}\ in\ x * 3) - c$$

Transformacije uslova

Razmotrimo, izraz:

```
if (x || y) then E1 else E2
```

Ovde je || je operator disjunkcije, definisan sa:

```
|| = \ a b → case a of {True → True; False → b}
```

Jednostavno izvršimo "case-case" transformacije:

```
case (case x of {True → True ; False → y}) of
True → E1
False → E2
```

Pridruživanje tačaka

Sada, primenjujući (novu) "case-case" transformaciju, dobijamo:

let e1 = E1 ; e2 = E2
in case x of
True → case True of {True → e1; False → e2}
False → case y of {True → e1; False → e2}

Dalje vršimo transformaciju "poznatih-konstruktora"

let e1 = E1
in case x of
True → e1
False → case y of {True → e1; False → E2}

Backend

- Prevođenje *CoreSyn* u *StgSyn (GHC's intermediate language)*, i to u dve faze:
  - CorePrep
    - A-normalna forma (eng. *A-normal form (ANF)*) - Sabry i Fellisen 1992. godine. Na primer:

f(g(x),h(y))
let v_0 = g(x) in
let v_1 = h(y) in
f(v_0, v_1)

  - CoreToStg

- STG program se pomoću generatora kôda pretvara u C--.

GHC kôd generator

- Ideja:
  - Problemi sa prevođenjem na C
  - Podrška za kôd generatore niskog nivoa
  - Razvoj jezika srednje-niskog nivoa
- Najznačajniji programi za generisanje izvršnog kôda:
  - NGC (eng. *Native Generated Code*)
  - LLVM (eng. *Low-Level Virtual Machine*)
  - C--

LLVM - razlozi implementacije

- Potreba za pravljenjem kompajlera visokih performansi za generisanje kôda
- GHC proizvodi Haskel programe koji se brzo izračunavaju
- Prilagođavanje svake nove verzije LLVM-a postojećoj aplikaciji GHC-a
- Uključuje celokupan lanac alata

GHC-6.13 (NCG)	GHC-6.13 (C)	GHC-6.13 (LLVM)	GCC-4.4.3
2.876s	0.576s	0.516s	0.335s

Table 1: Različita vremena generisanja izvršnih kôdova

Zaključak

- GHC kompajler je u stalnom razvoju:
  - strategije izvođenja
  - generacijskog garbage collector-a
  - prekoračenja heap memorije

Literatura

- Edgewall Software. The Glasgow Haskell, Current Status, 2018. on-line: https://ghc.haskell.org/trac/ghc
- Stanford University. Why is haskell difficult, 2018. on-line: http://www.scs.stanford.edu/11au-cs240h/notes/ghc.html
- Simon Marlow (editor). Haskell 2010 Language Report. Haskell community, April 2010

Kraj

Hvala na pažnji!

Pitanja?