

Etape Prevođenja Haskela do mašinskog jezika

Marija Mijailović, Miroslav Mišljenović
Nemanja Antić, Filip Lazić

Matematički fakultet, Beograd

21. Maj 2018.

- 1 Uvod
- 2 Uvod u Haskell i GHC
- 3 Frontend
- 4 Middle end
- 5 Backend

- Funkcionalna paradigma
 - Matematičke funkcije
 - Eliminacija bočnih efekta
 - Dokazivanje korektnosti programa
- Džon Bakus
 - Tjuringova nagrada za doprinos u razvoju *FORTTRAN-a*

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

Glazgov Haskell Kompajler (GHC)

Sastoji se iz tri faze:

- Frontend - pretvaranje u Core jezik
- Middle end - dodatne optimizacije
- Backend - STG mašina

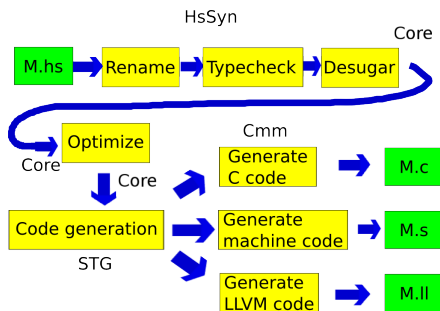


Figure 1: Razvojne etape pravljenja izvršnog kôda

- 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

Primer zamene *RdrNames* sa *Names*:

```
module K where f x = True
module N where import K
module M where import N(f) as Q f = (f, M.f, Q.f, \f -> f)
```


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`

- Pravi jednostavan tipizirani lambda račun
- Jednostavan lenji funkcionalni jezik Asembler funkcionalnog jezika

Sastoji od nekoliko elemenata:

- varijable
- literali
- let
- case
- lambda apstrakcije
- aplikacije

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*)

- Samoumetanje (eng. *inlining itself*)
 - SGNF (eng. *Weak Head Normal Form (WHNF)*)
 - Ne-SGNF (eng. *Non-Weak Head Normal Form (Non-WHNF)*)

let $f = \backslash x \rightarrow x * 3$ in $f(a + b) - c$

\Rightarrow *[inline f]*

let $f = \backslash x \rightarrow x * 3$ in $(\backslash x \rightarrow x * 3)(a + b) - c$

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

let $f = \lambda x. x * 3$ in $(\lambda x. x * 3)(a + b) - c$
 $\implies [deadf]$
 $(\lambda x. x * 3)(a + b) - c$

- β redukcija (eng. *beta reduction*)

$(\lambda x. x * 3)(a + b) - c \implies [beta] (let x = a + b in x * 3) - c$

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

- Tehnika za redukciju kôda
- 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
```

- Očigledno je sledeće pojednostavljenje, tj. vršimo transformaciju “poznatih-konstruktoru”

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

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 v0 = g(x) in  
let v1 = h(y) in  
f(v0,v1)
```

- CoreToStg

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

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*)
- C--
- LLVM

LLVM - razlozi implementacije

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

Table 1: Različita vremena generisanja izvršnih kôdova

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

Hvala na pažnji!

Pitanja?