CS314. Функциональное программирование

Лекция 23. Внутреннее устройство компилятора GHC

В. Н. Брагилевский

Направление «Фундаментальная информатика и информационные технологии»
Институт математики, механики и компьютерных наук имени И. И. Воровича
Южный федеральный университет

24 декабря 2016 г.

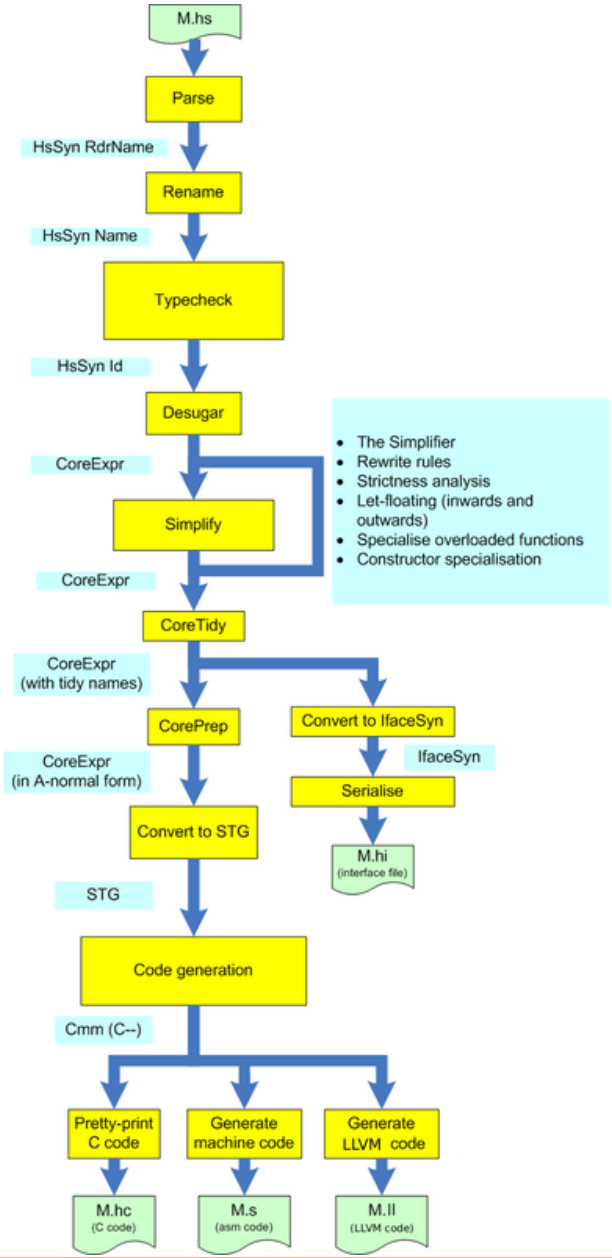
Компилятор GHC

- Собственно компилятор
  - Менеджер компиляции
  - Компилятор одного модуля (Hsc)
  - Драйвер
- Базовые библиотеки (base)
- Runtime System (RTS)
- Система сборки

Содержание

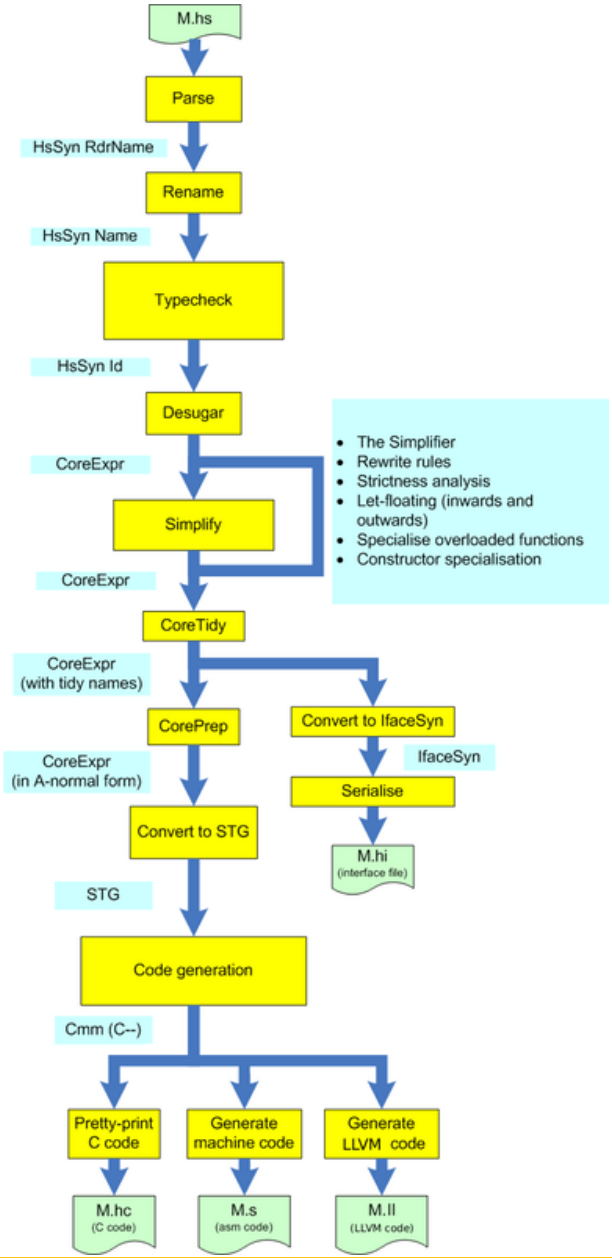
Содержание

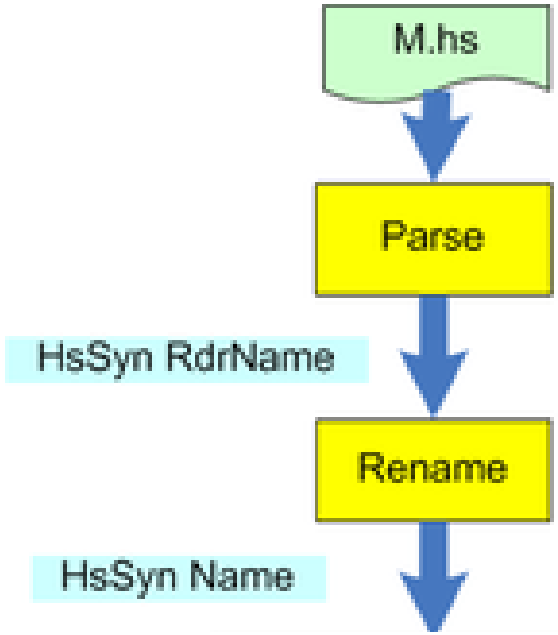
Общая схема



1. Разбор текста модуля (лексический и синтаксический анализ).
2. Разрешение имён.
3. Проверка типов.
4. Удаление синтаксического сахара и преобразование в Core.
5. Оптимизация.
6. Кодогенерация.

Компиляция модуля (1)





- HsSyn — дерево разбора (AST, Abstract Syntax Tree).
- RdrName, Name — имена с дополнительной информацией.

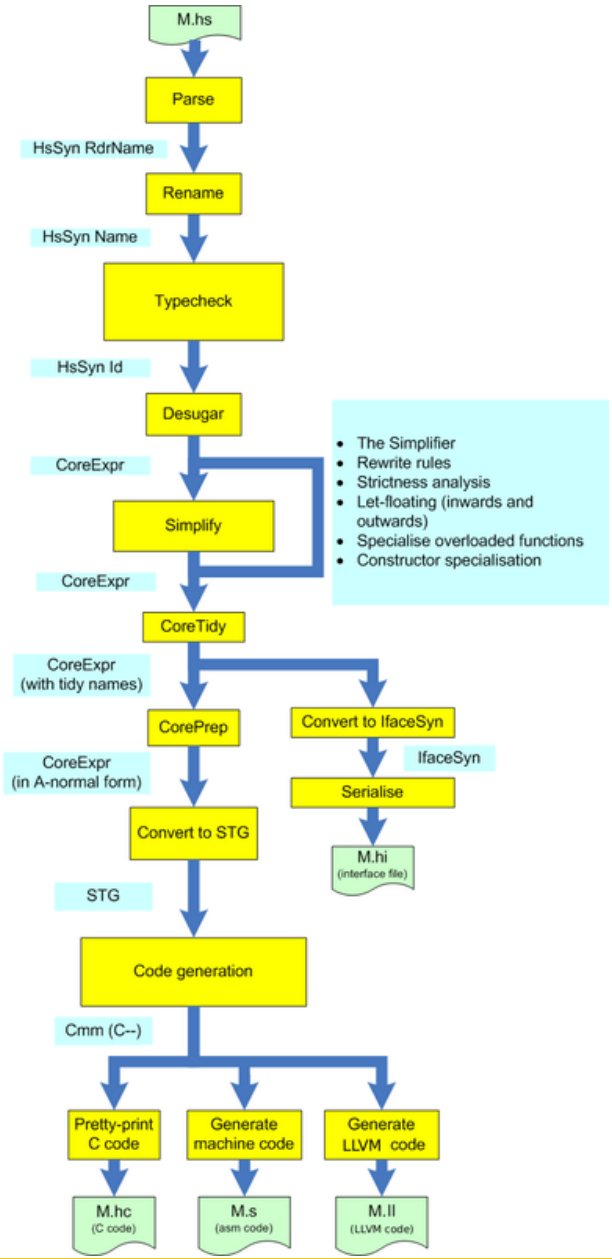
AST кода на Haskell (HsSyn) — модуль

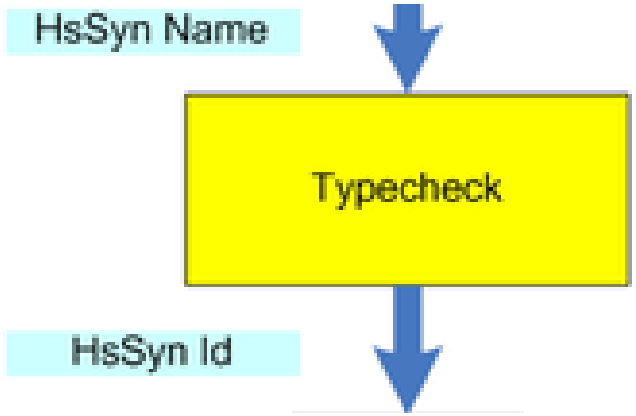
```
data HsModule name
  = HsModule {
      hsmodName :: Maybe (Located ModuleName),
      hsmodExports :: Maybe (Located [LIE name]),
      hsmodImports :: [LImportDecl name],
      hsmodDecls :: [LHsDecl name],
      -- ...
    }
```

AST кода на Haskell (HsSyn) — определения

```
type LHsDecl id = Located (HsDecl id)

data HsDecl id
  = TyClD       (TyClDecl id)       -- ^ Type or Class Declaration
  | InstD       (InstDecl  id)      -- ^ Instance declaration
  | DerivD      (DerivDecl id)      -- ^ Deriving declaration
  | ValD        (HsBind id)         -- ^ Value declaration
  | SigD        (Sig id)            -- ^ Signature declaration
  | DefD        (DefaultDecl id)    -- ^ 'default' declaration
  | ForD        (ForeignDecl id)    -- ^ Foreign declaration
  | WarningD    (WarnDecls id)      -- ^ Warning declaration
  | AnnD        (AnnDecl id)        -- ^ Annotation declaration
  | RuleD       (RuleDecls id)      -- ^ Rule declaration
  | VectD       (VectDecl id)       -- ^ Vectorise declaration
  | SpliceD     (SpliceDecl id)     -- ^ Splice declaration
                                    -- (Includes quasi-quotes)
  | DocD        (DocDecl)           -- ^ Documentation comment
  | RoleAnnotD  (RoleAnnotDecl id) -- ^ Role annotation
```

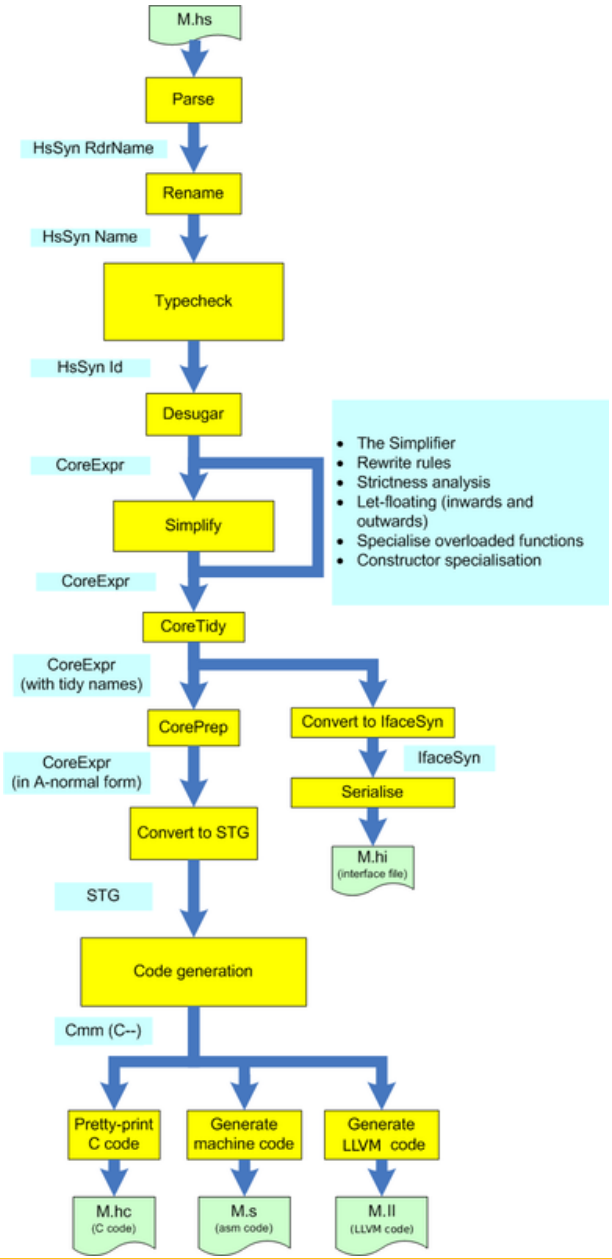
Компиляция модуля (2): проверка типов

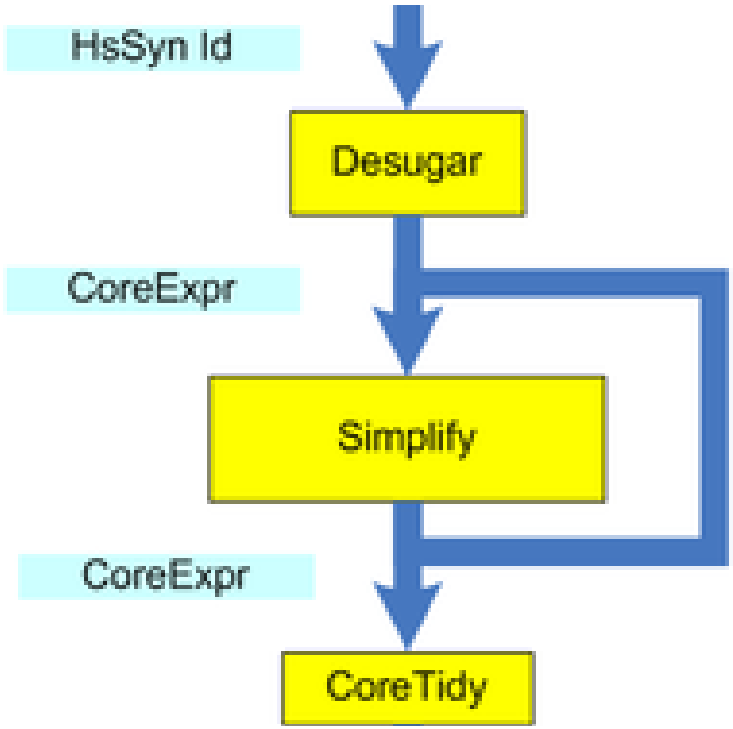




- Id — имя с информацией о типе (каждое выражение по результатам проверки типов имеет тип, указанный явно или выведенный).

Компиляция модуля (3): Core и его оптимизация





- Удаление синтаксического сахара — преобразование во внутренний язык (Core).
- Оптимизация.

Внутренний язык Core

Синтаксис: выражения и паттерны

t, e, u	$::=$	x	Переменные
	$\mid$	K	Конструктор данных
	$\mid$	k	Литералы
	$\mid$	$\lambda x : \sigma . e \mid e\ u$	Абстракция и применение (значения)
	$\mid$	$\Lambda a : \eta . e \mid e\ \varphi$	Абстракция и применение (типы)
	$\mid$	$\mathrm{let}\ \overline{x : \tau = e}\ \mathrm{in}\ u$	Локальное связывание
	$\mid$	$\mathrm{case}\ e\ \mathrm{of}\ \overline{p \to u}$	Выбор варианта
	$\mid$	$e \rhd \gamma$	Преобразование типа значения (cast)
	$\mid$	$\lfloor \gamma \rfloor$	Приведение типа (coercion)
p	$::=$	$K\ \overline{c : \eta}\ \overline{x : \tau}$	Паттерны

- Греческие буквы — типы и сорта
- Надчёркивание — списки подтермов
- System FC

Алгебраический тип данных для Core

```
type CoreExpr = Expr Var

data Expr b
  = Var   Id
  | Lit   Literal
  | App   (Expr b) (Arg b)
  | Lam   b (Expr b)
  | Let   (Bind b) (Expr b)
  | Case  (Expr b) b Type [Alt b]
  | Cast  (Expr b) Coercion
  | Tick  (Tickish Id) (Expr b)
  | Type  Type
  | Coercion Coercion
  deriving (Data, Typeable)

type Arg b = Expr b
type Alt b = (AltCon, [b], Expr b)
data AltCon = DataAlt DataCon | LitAlt  Literal | DEFAULT
data Bind b = NonRec b (Expr b) | Rec [(b, (Expr b))]
```

Оптимизация Core

- Упрощение (simplifier).
- Правила переписывания термов.
- Анализ строгости.
- Специализация перегруженных функций.
- Специализация конструкторов.
- ...

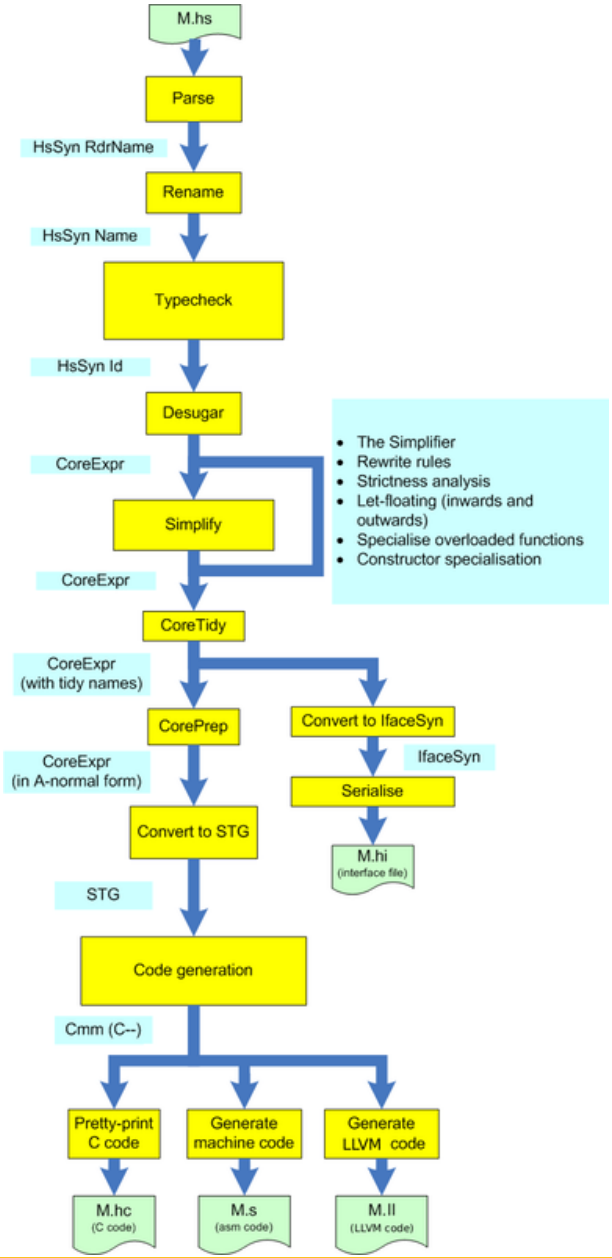
Пример кода Core

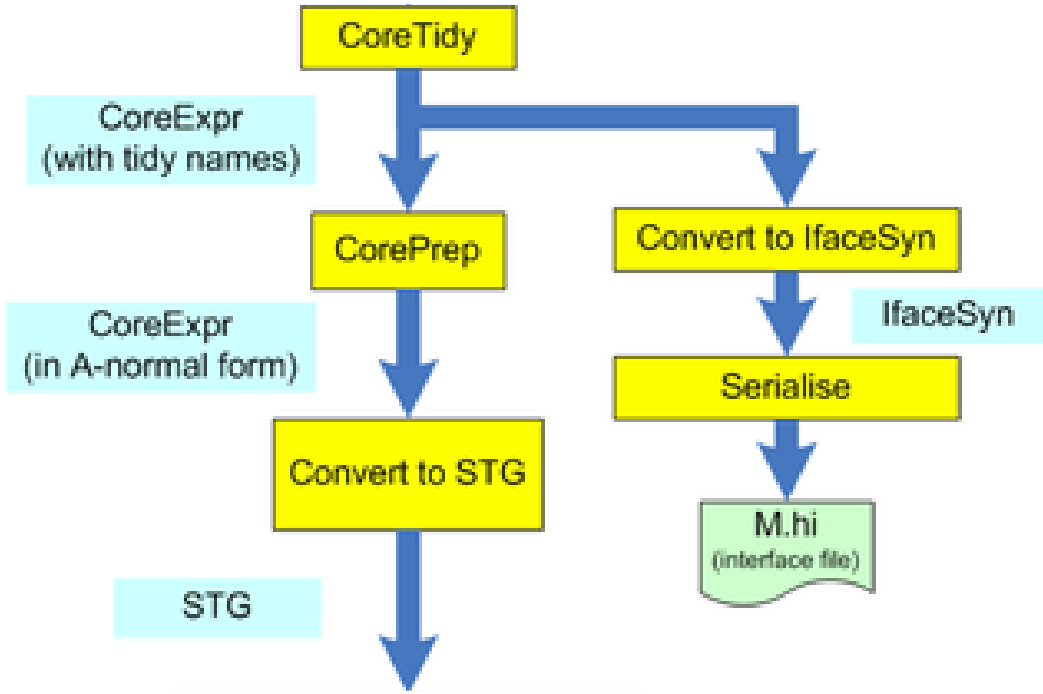
```
f a b = a + b

main = print $ f 2 3
```

```
$ ghc -c ex.hs -ddump-simpl
```

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

Компиляция модуля (4): подготовка к кодогенерации



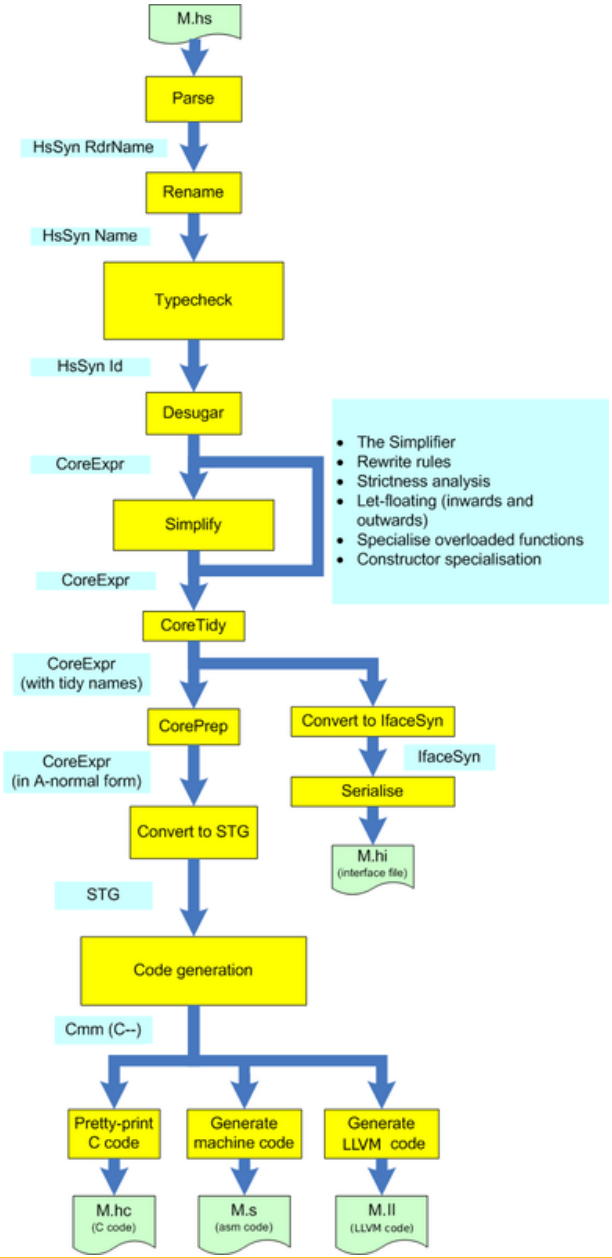


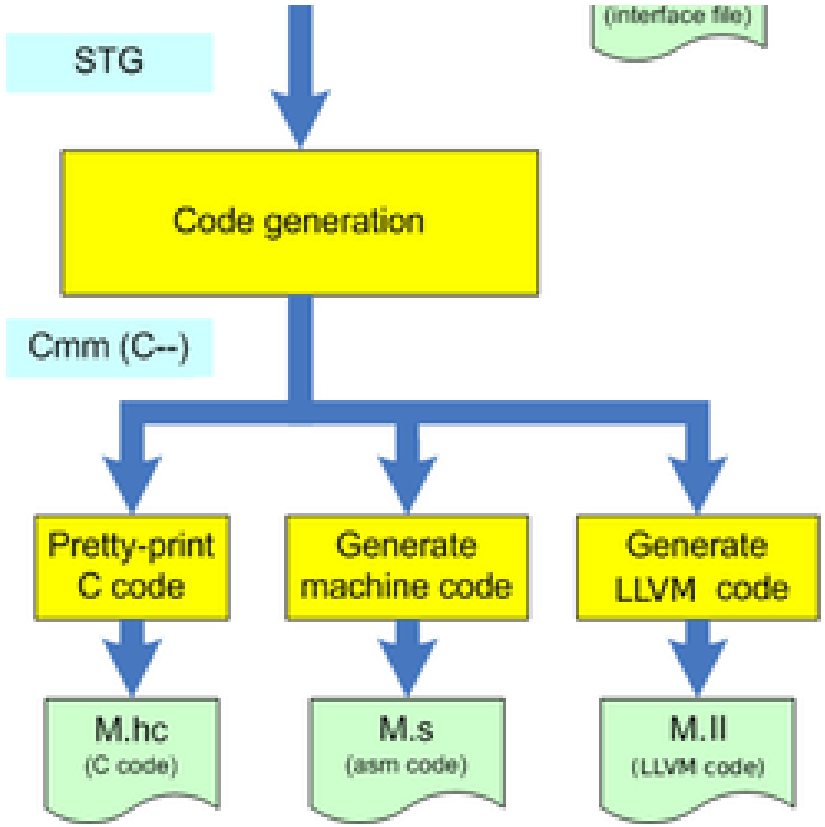
- Построение нормальной формы.
- Генерация интерфейсного файла (для межмодульной оптимизации).
- STG — Spineless Tagless G-machine

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;
```

Компиляция модуля (5): кодогенерация





- Cmm — низкоуровневый императивный язык с явным стеком.

Фрагмент кода Cmm

```
I64[(old + 24)] = stg_bh_upd_frame_info;
I64[(old + 16)] = _c10Y::I64;
I64[Hp - 24] = GHC.Integer.Type.S#_con_info;
I64[Hp - 16] = 3;
_c111::P64 = Hp - 23;
I64[Hp - 8] = GHC.Integer.Type.S#_con_info;
I64[Hp] = 2;
_c112::P64 = Hp - 7;
R2 = GHC.Num.$fNumInteger_closure;
I64[(old + 48)] = stg_ap_pp_info;
P64[(old + 40)] = _c112::P64;
P64[(old + 32)] = _c111::P64;
call GHC.Num.+_info(R2) args: 48, res: 0, upd: 24;
```

Кодогенерация и RTS

Результаты кодогенерации

- Байт-код для интерпретации.
- Нативный код (на языке ассемблера).
- C-код.
- LLVM IR (промежуточный язык LLVM).

Функции RTS

- Управление памятью (в том числе сборка мусора).
- Управление потоками (потоки ОС и легковесные потоки приложения).
- Реализация примитивных операций.
- Подключение динамических библиотек и реализация FFI.
- Интерпретация байт-кода.

Содержание

Способы расширения функционала

- Определяемые пользователем правила переписывания термов.
- Плагины к компилятору.
- Компилятор как библиотека (GHC API).

Правила переписывания термов

```
{-# RULES "fold/build"
    forall k z (g::forall b. (a->b->b) -> b -> b) .
       foldr k z (build g) = g k z
 #-}
```

- Применение семантически корректных преобразований.

Плагины к компилятору

- Плагин — это один проход на этапе оптимизации — функция из Core в Core в отдельном файле.
- Подключение флагом компилятора или директивой в исходном коде.
- Компилятор динамически подключает преобразование и выполняет его.
- Аннотации плагинов — способ указания места применения преобразования.

Компилятор как библиотека (GHC API)

- Модульность позволяет подменять отдельные этапы компиляции.
- Каждый этап — функция, которую может вызвать пользователь в собственной программе.
- Компилятор целиком или частично может встраиваться в программу пользователя.

Литература

- S. Marlow, S. Peyton Jones. The Glasgow Haskell Compiler. (Сборник The Architecture of Open Source Applications, vol. 2). `http://www.aosabook.org/en/ghc.html`

Репозитории с лабораторными работами

- Содержимое репозиториев может быть удалено в любой момент, начиная с 01.02.2017.
- Прошу не публиковать и никаким образом не распространять задания и их решения.

Добор баллов

- Первый добор в день второй консультации (16 января), 14:00, а. 202:
  - Возможна сдача (досдача) проекта.
  - Переписывание контрольных работ 1 и 2.
- Последующие доборы:
  - Задания на добор формируются индивидуально в зависимости от набранных в семестре баллов (по темам с наименьшим количеством набранных баллов).
  - Задания на каждый следующий добор формируются с учётом результатов предыдущего.
  - Добор в феврале по материалам первого и второго модулей.
  - Начиная с марта на добор будут предлагаться задания по темам первого модуля.

Экзамен

- Экзамен проводится в письменной форме, продолжительность — 1,5 часа, начало в 08:30.
- В билете 25 вопросов, предполагающих короткие ответы (от одного слова до двух–трёх предложений).
- Ответ на каждый вопрос оценивается одним или двумя баллами.
- Экзамен считается сданным при наборе не менее чем 22 баллов.
- Всего за экзамен можно получить до 40 баллов.
- На странице курса опубликованы программа курса и демо-вариант экзамена.

Спасибо!