CS314. Функциональное программирование

Лекции 21–22. Внутреннее устройство компилятора GHC

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

22 декабря 2017 г.

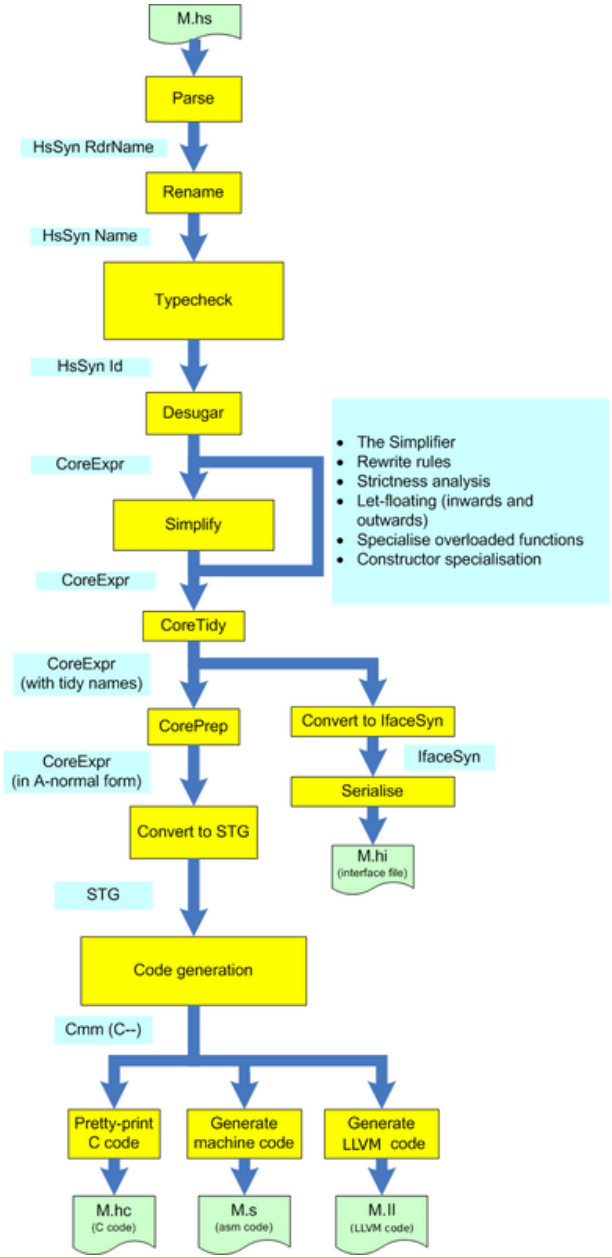
Компилятор GHC

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

Содержание

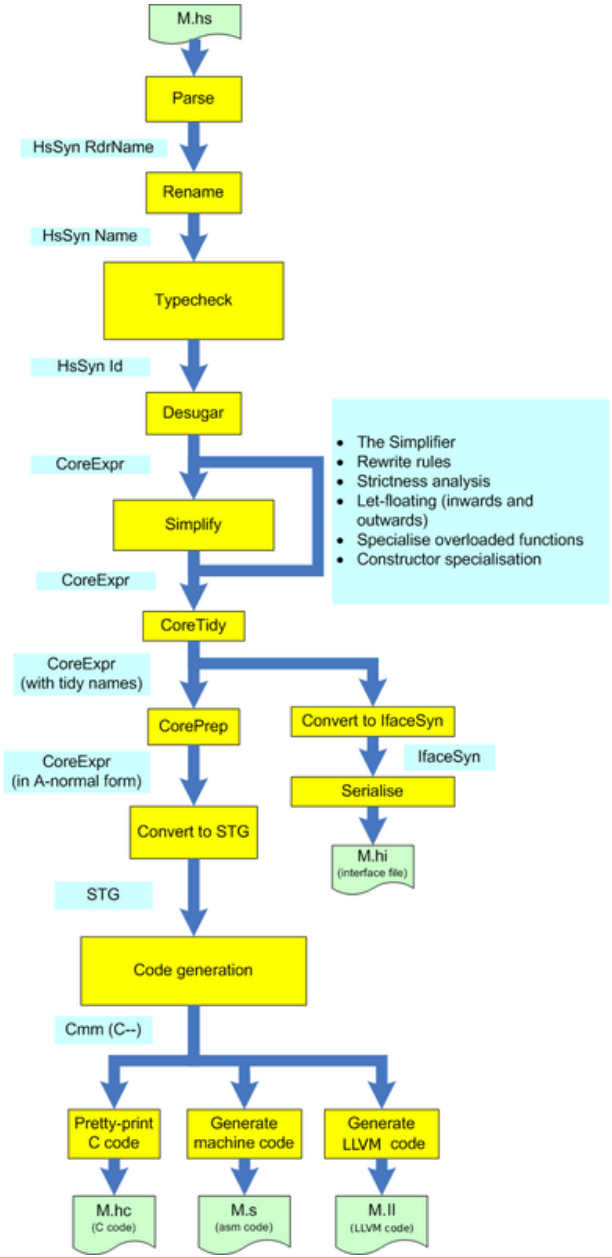
Содержание

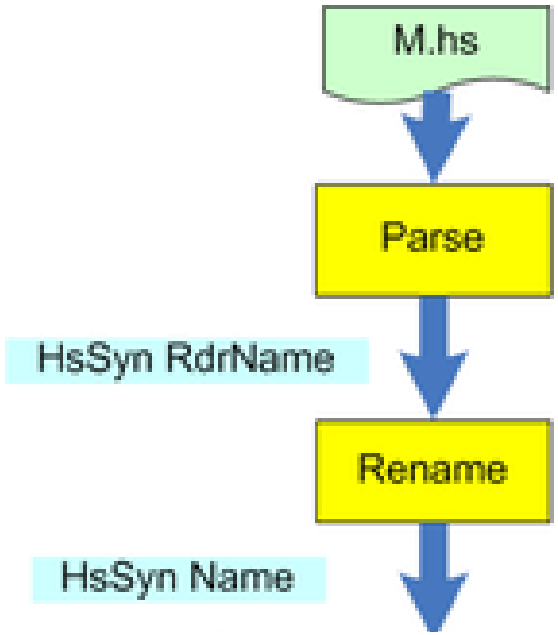
Общая схема



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.

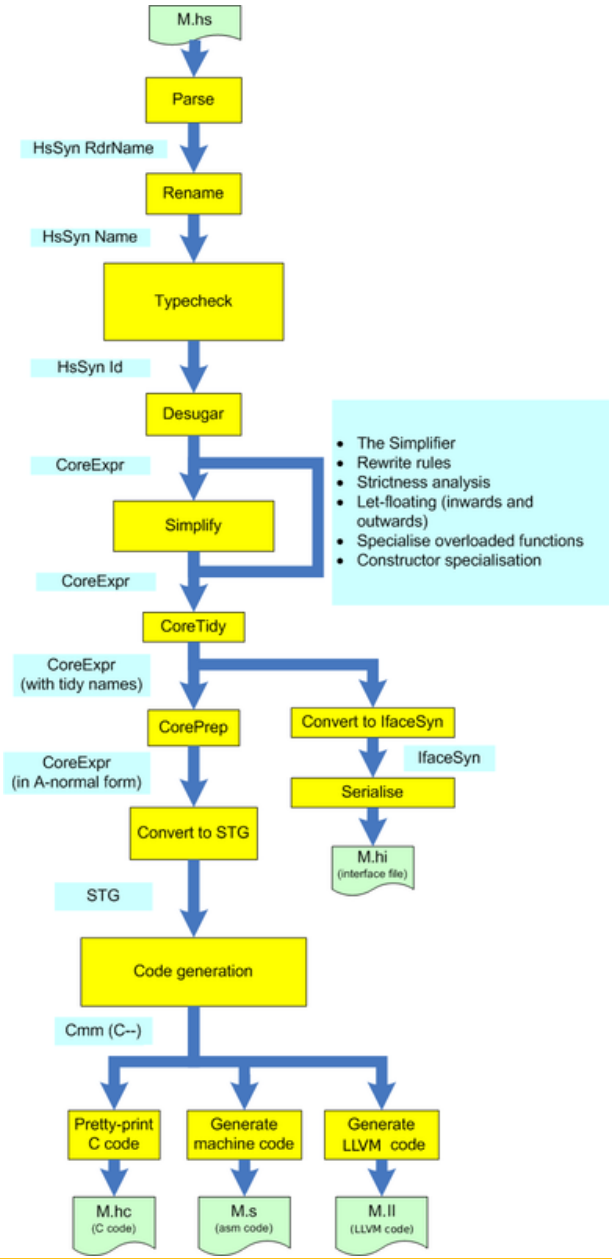
AST кода на Haskell (HsSyn) — модуль

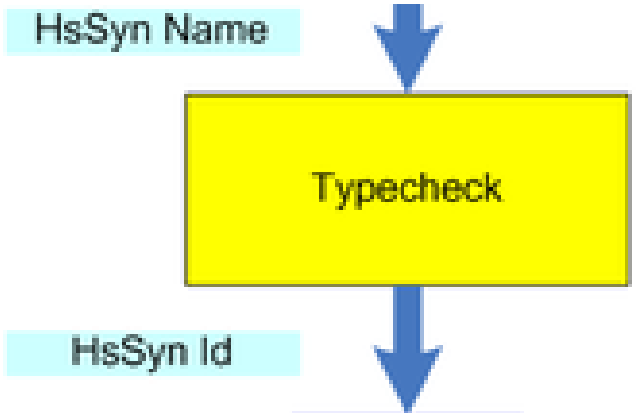
```
data HsModule name
  = HsModule {
      hsmodName :: Maybe (Located ModuleName),
      hsmodExports :: Maybe (Located [LIE name]),
      hsmodImports :: [LImportDecl name],
      hsmodDecls :: [LHsDecl name],
      -- ...
    }
```

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
```

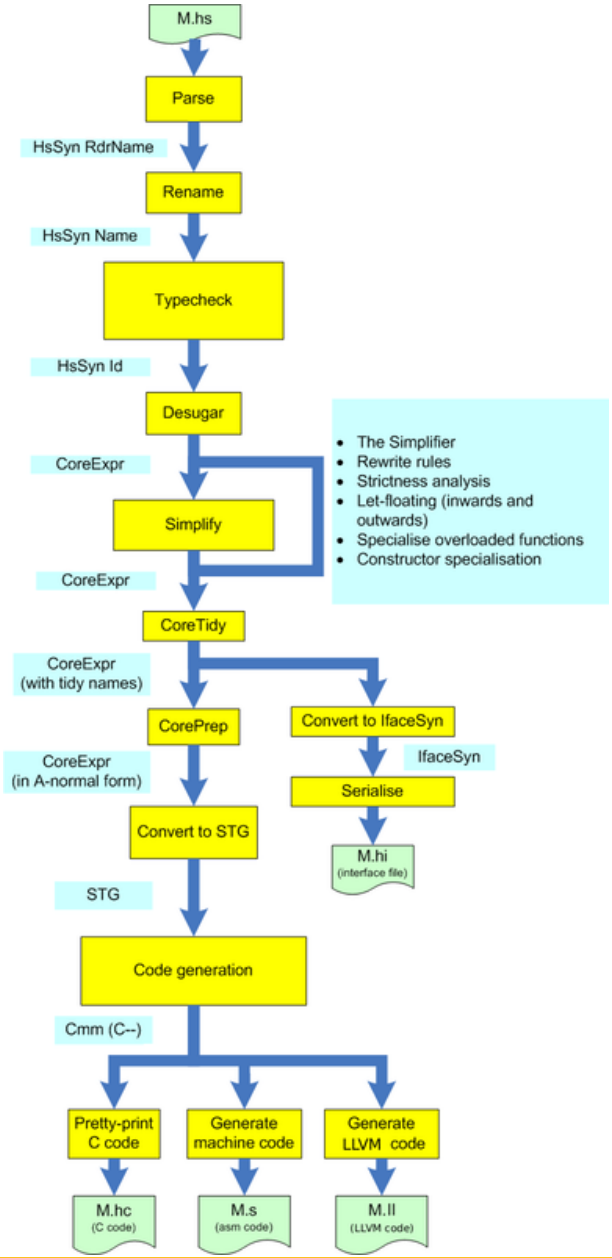
Компиляция модуля (2): проверка типов

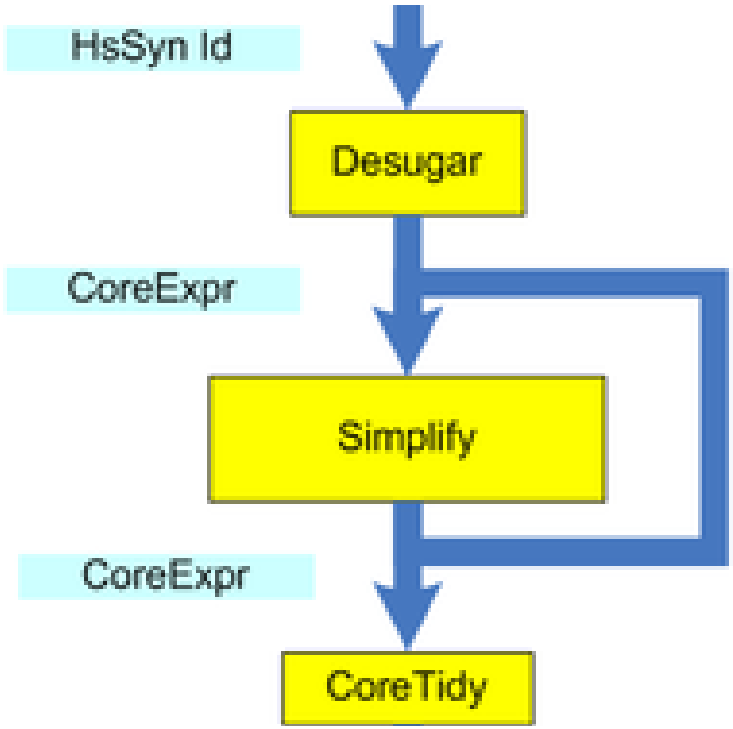




- Id — имя с информацией о типе (каждое выражение по результатам проверки типов имеет тип, указанный явно или выведенный).

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
	$\mid$	$\mathrm{let}\ \overline{x : \tau = e}\ \mathrm{in}\ u$	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))]
```

Содержание

Содержание

Синтаксис

`t`	::=	*термы:*
	`x`	*переменная*
	`λx.t`	*абстракция*
	`t t`	*применение*
`v`	::=	*значения:*
	`λx.t`	*значение-абстракция*

Вычисление (операционная семантика)

$$\boxed{t \to t'}$$

$$\frac{t_1 \to t_1'}{t_1\ t_2 \to t_1'\ t_2} \quad \text{(E-App1)}$$

$$\frac{t_2 \to t_2'}{v_1\ t_2 \to v_1\ t_2'} \quad \text{(E-App2)}$$

$$(\lambda x.t_{12})\ v_2 \to [x \mapsto v_2]t_{12} \quad \text{(E-AppAbs)}$$

Возможные расширения

- Логические константы и числа, операции над ними
- Пары и списки

Альтернатива

- Кодирование этих понятий на языке λ-исчисления
- Добавление новых синтаксических конструкций

Содержание

Синтаксис

t	::=	*термы:*
	x	*переменная*
	λ x :T .t	*абстракция*
	t t	*применение*
v	::=	*значения:*
	λ x :T .t	*значение-абстракция*
T	::=	*типы:*
	T $\rightarrow$ T	*тип функций*
Γ	::=	*контексты:*
	$\emptyset$	*пустой контекст*
	Γ,x:T	*связывание термовой переменной*

Вычисление $\boxed{t \to t'}$

$$\frac{t_1 \to t_1'}{t_1\ t_2 \to t_1'\ t_2} \quad \text{(E-App1)}$$

$$\frac{t_2 \to t_2'}{v_1\ t_2 \to v_1\ t_2'} \quad \text{(E-App2)}$$

$$(\lambda\ x\ :T_{11}\ .t_{12})\ v_2 \to [x \mapsto v_2]t_{12} \quad \text{(E-AppAbs)}$$

Типизация

$$\boxed{\Gamma \vdash t : T}$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \text{(T-Var)}$$

$$\frac{\Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1 . t_2 : T_1 \to T_2} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-App)}$$

Расширения

- Базовые типы (Bool, Nat,. . .)
- Константы и операции для базовых типов (true, false, if/then/else, 0, succ, pred, iszero)
- Правила вычисления и правила типизации

Новые правила типизации

$$\boxed{\Gamma \vdash \mathtt{t} : \mathtt{T}}$$

$$\vdash \mathtt{true} : \mathtt{Bool} \qquad \text{(T-True)}$$

$$\vdash \mathtt{false} : \mathtt{Bool} \qquad \text{(T-False)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Bool} \quad \Gamma \vdash \mathtt{t}_2 : \mathtt{T} \quad \Gamma \vdash \mathtt{t}_3 : \mathtt{T}}{\Gamma \vdash \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 : \mathtt{T}} \qquad \text{(T-If)}$$

$$\vdash \texttt{0} : \texttt{Nat} \qquad \text{(T-Zero)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{Nat}}{\Gamma \vdash \texttt{succ t}_1 : \texttt{Nat}} \qquad \text{(T-Succ)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{Nat}}{\Gamma \vdash \texttt{pred t}_1 : \texttt{Nat}} \qquad \text{(T-Pred)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{Nat}}{\Gamma \vdash \texttt{iszero t}_1 : \texttt{Bool}} \qquad \text{(T-IsZero)}$$

Деревья вывода типов: пример

$$\dfrac{\dfrac{\dfrac{\texttt{x:Bool} \in \texttt{x:Bool}}{\texttt{x:Bool} \vdash \texttt{x:Bool}}\ \text{T-Var}}{\vdash \lambda\texttt{x:Bool.x} \ :\ \texttt{Bool} \to \texttt{Bool}}\ \text{T-Abs} \qquad \dfrac{}{\vdash \texttt{true} \ :\ \texttt{Bool}}\ \text{T-True}}{\vdash (\lambda\texttt{x:Bool.x})\ \texttt{true} \ :\ \texttt{Bool}}\ \text{T-App}$$

Необходимые свойства отношения типизации

Безопасность = продвижение + сохранение

- Продвижение (progress): правильно типизированный терм является либо значением, либо может проделать следующий шаг вычисления.
- Сохранение (preservation): если правильно типизированный терм делает шаг вычисления, то получающийся терм также правильно типизирован.

Другие свойства отношения типизации

- Леммы об инверсии (обращение правил типизации)
- Единственность типа
- Канонические формы (пример: `succ (succ (succ 0))`)
- Сохранение типов при подстановке
- Перестановка и ослабление в контекстах
- Нормализация (завершаемость)
- Явная и неявная типизация

Стирание типов перед вычислением

Типы не участвуют в вычислениях, поэтому после проверки их можно удалить.

$$
\begin{array}{lcl}
\mathit{erase}\,(\mathtt{x}) & = & \mathtt{x} \\
\mathit{erase}\,(\lambda \mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2) & = & \lambda \mathtt{x}.\ \ \mathit{erase}\,(\mathtt{t}_2) \\
\mathit{erase}\,(\mathtt{t}_1\ \ \mathtt{t}_2) & = & \mathit{erase}\,(\mathtt{t}_1)\ \mathit{erase}\,(\mathtt{t}_2)
\end{array}
$$

Вычисление должно коммутировать со стиранием.

Проблема дублирования кода в STLC

```
doubleNat  = λf:Nat -> Nat. λx:Nat. f (f x)
doubleBool = λf:Bool -> Bool. λx:Bool. f (f x)
doubleFun  = λf:(Nat -> Nat) -> (Nat -> Nat).
                λx:Nat -> Nat. f (f x)
```

Содержание

Примеры полиморфных функций

id : ∀X. X −> X
id = ΛX. λx:X. x
idNat = **id** [Nat]

double : ∀X. (X −> X) −> X −> X
double = ΛX. λf:X −> X. λa:X. f (f a)
doubleNat = double [Nat]

quadruple = ΛX. double [X −> X] (double [X])

- Параметрический полиморфизм

Синтаксис (расширяет STLC)

t	::=	*термы:*
	x	*переменная*
	λx:T.t	*абстракция*
	t t	*применение*
	ΛX.t	*абстракция типа*
	t [T]	*применение типа*
v	::=	*значения:*
	λx:T.t	*значение-абстракция*
	ΛX.t	*значение-абстракция типа*

$\mathtt{T}$	$::=$	*типы:*
	$\mathtt{X}$	*типовая переменная*
	$\mathtt{T} \to \mathtt{T}$	*тип функций*
	$\forall \mathtt{X}.\mathtt{T}$	*универсальный тип*
Γ	$::=$	*контексты:*
	$\emptyset$	*пустой контекст*
	$\Gamma, \mathtt{x} : \mathtt{T}$	*связывание термовой переменной*
	$\Gamma, \mathtt{X}$	*связывание типовой переменной*

Вычисление $\boxed{t \to t'}$

$$\frac{t_1 \to t_1'}{t_1\ t_2 \to t_1'\ t_2} \quad \text{(E-App1)}$$

$$\frac{t_2 \to t_2'}{v_1\ t_2 \to v_1\ t_2'} \quad \text{(E-App2)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \to [x \mapsto v_2]t_{12} \quad \text{(E-AppAbs)}$$

$$\frac{t_1 \to t_1'}{t_1\ [T_2] \to t_1'\ [T_2]} \quad \text{(E-TApp)}$$

$$(\Lambda X.t_{12})\ [T_2] \to [X \mapsto T_2]t_{12} \quad \text{(E-TAppTAbs)}$$

Типизация

$\boxed{\Gamma \vdash t : T}$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \text{(T-Var)}$$

$$\frac{\Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1 . t_2 : T_1 \to T_2} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-App)}$$

$$\frac{\Gamma, X \vdash t_2 : T_2}{\Gamma \vdash \Lambda X . t_2 : \forall X . T_2} \quad \text{(T-TAbs)}$$

$$\frac{\Gamma \vdash t_1 : \forall X . T_{12}}{\Gamma \vdash t_1\ [T_2] : [X \mapsto T_2]T_{12}} \quad \text{(T-TApp)}$$

Свойства System F

- Продвижение и сохранение.
- Нормализация
- Для определения незавершающихся рекурсивных функций необходима дополнительная конструкция `fix`.
- Возможность стирания типов.
- (!!!) Реконструкция типов после стирания — неразрешимая задача.
- Варианты System F с ограничениями: пренексный полиморфизм (типовые переменные имеют областью значения типы без кванторов), полиморфизм ранга 2.
- Импредикативность (переменная под квантором может иметь значением сам определяемый объект).

Содержание

Функции на типах и виды

$\mathsf{Id} = \Lambda X.\ X$

$\mathsf{Id} = \Lambda X::*.\ X$

$\mathsf{Pair} = \Lambda A::*.\ \Lambda B::*.\forall X.(A \to B \to X) \to X$

$\mathsf{Pair} :: * \to * \to *$

$\mathsf{PairNB} = \mathsf{Pair}\ [\mathsf{Nat}]\ [\mathbf{Bool}]$

System F_ω

- Виды: $*$, $* \to *$, $* \to * \to *$, ...
- Отношение видообразования (присвоение видов) $\Gamma \vdash T :: K$
- Отношение эквивалентности определений типов ($S \equiv T$), так как синтаксически различные типы могут означать одно и то же, например:
  Id Nat -> Id **Bool** $\equiv$ Nat -> **Bool**
- Параллельная редукция — направленный вариант экивалентности определений типов.

Примеры правил

$$\frac{\Gamma, \mathtt{X}::\mathtt{K}_1 \vdash \mathtt{T}_2::\mathtt{K}_2}{\Gamma \vdash \Lambda \mathtt{X}::\mathtt{K}_1 \,.\, \mathtt{T}_2 \;::\; \mathtt{K}_1 \to \mathtt{K}_2} \quad (\text{K-Abs})$$

$$\frac{\Gamma, \mathtt{X}::\mathtt{K}_1 \vdash \mathtt{T}_2::*}{\Gamma \vdash \forall \mathtt{X}::\mathtt{K}_1 \,.\, \mathtt{T}_2 \;::\; *} \quad (\text{K-All})$$

$$\frac{\mathtt{S}_2 \equiv \mathtt{T}_2}{\forall \mathtt{X}::\mathtt{K}_1 \,.\, \mathtt{S}_2 \equiv \forall \mathtt{X}::\mathtt{K}_1 \,.\, \mathtt{T}_2} \quad (\text{Q-All})$$

Изменения в правилах типизации (фрагмент)

$$\frac{\Gamma, \mathtt{X} :: \mathtt{K}_1 \vdash \mathtt{t}_2 \ : \ \mathtt{T}_2}{\Gamma \vdash \lambda \mathtt{X} :: \mathtt{K}_1 . \mathtt{t}_2 \ : \ \forall \mathtt{X} :: \mathtt{K}_1 . \mathtt{T}_2} \quad \text{(T-TAbs)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 \ : \ \forall \mathtt{X} :: \mathtt{K}_{11} . \mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{T}_2 \ :: \ \mathtt{K}_{11}}{\Gamma \vdash \mathtt{t}_1 \ [\mathtt{T}_2] \ : \ [\mathtt{X} \mapsto \mathtt{T}_2]\mathtt{T}_{12}} \quad \text{(T-TApp)}$$

$$\frac{\Gamma \vdash \mathtt{t} \ : \ \mathtt{S} \qquad \mathtt{S} \equiv \mathtt{T} \qquad \Gamma \vdash \mathtt{T} \ :: \ *}{\Gamma \vdash \mathtt{t} \ : \ \mathtt{T}} \quad \text{(T-Eq)}$$

На пути к GHC Core

- Бестиповое λ-исчисление
- Простое типизированное λ-исчисление
- System F
- System F_ω

For many years, GHC's intermediate language was essentially:

- *System Fw, plus*
- *algebraic data types (including existentials)*

But that is inadequate to describe GADTs and associated types. So in 2006 we extended GHC to support System FC, which adds

- *equality constraints and coercions*

`https://ghc.haskell.org/trac/ghc/wiki/Commentary/Compiler/FC`

System FC (синтаксис)

t, e, u	$::=$	x	Переменные
	$\mid$	K	Конструкторы 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

Содержание

Выражения

$e,\ u$	$::=$		Expressions, *coreSyn/CoreSyn.lhs*:`Expr`
	$\mid$	n	`Var`: Variable
	$\mid$	lit	`Lit`: Literal
	$\mid$	$e_1\ e_2$	`App`: Application
	$\mid$	$\lambda n.e$	`Lam`: Abstraction
	$\mid$	**let** *binding* **in** e	`Let`: Variable binding
	$\mid$	**case** e **as** n **return** τ **of** $\overline{alt_i}^{\,i}$	`Case`: Pattern match
	$\mid$	$e \triangleright \gamma$	`Cast`: Cast
	$\mid$	$e_{\{tick\}}$	`Tick`: Internal note
	$\mid$	τ	`Type`: Type
	$\mid$	γ	`Coercion`: Coercion

Типы

$$
\begin{array}{lll}
\tau,\ \kappa,\ \sigma,\ \phi & ::= & \\
 & \mid & n \\
 & \mid & \tau_1\ \tau_2 \\
 & \mid & T\ \overline{\tau_i}^{\,i} \\
 & \mid & \tau_1 \rightarrow \tau_2 \\
 & \mid & \forall n.\tau \\
 & \mid & \mathsf{lit} \\
 & \mid & \tau \triangleright \gamma \\
 & \mid & \gamma
\end{array}
$$

GHC Core

- Правила типизации
- Правила видообразования
- Преобразование свидетелей приводимости
- Роли типов
- Инструменты проверки корректности

Содержание

Оптимизация Core

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

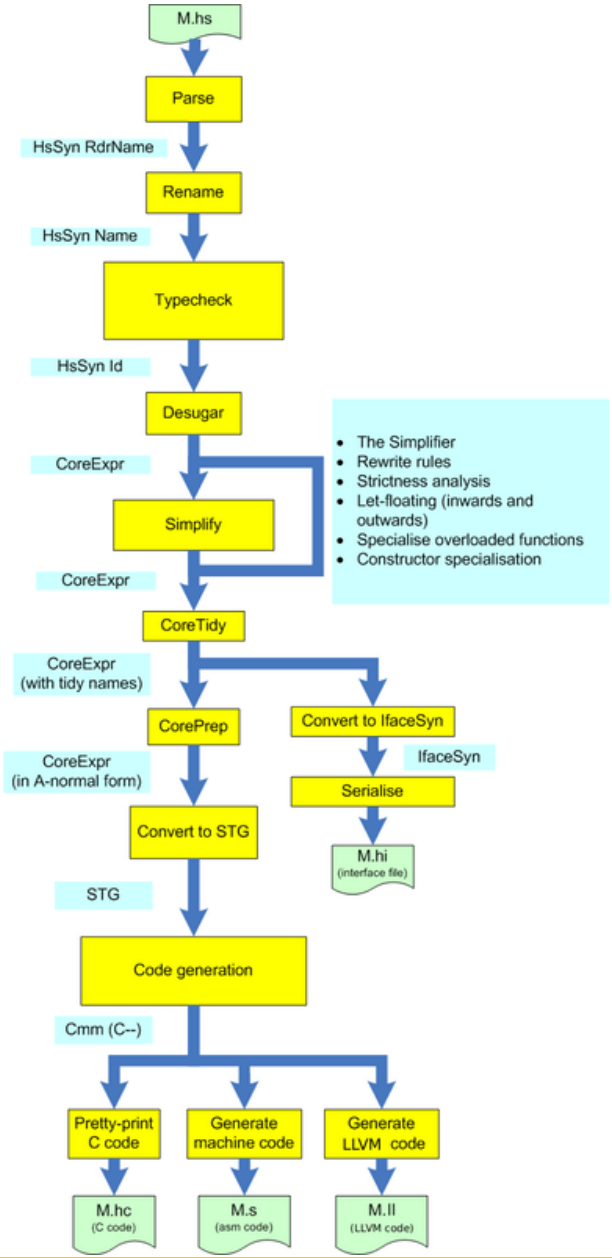
Пример кода Core

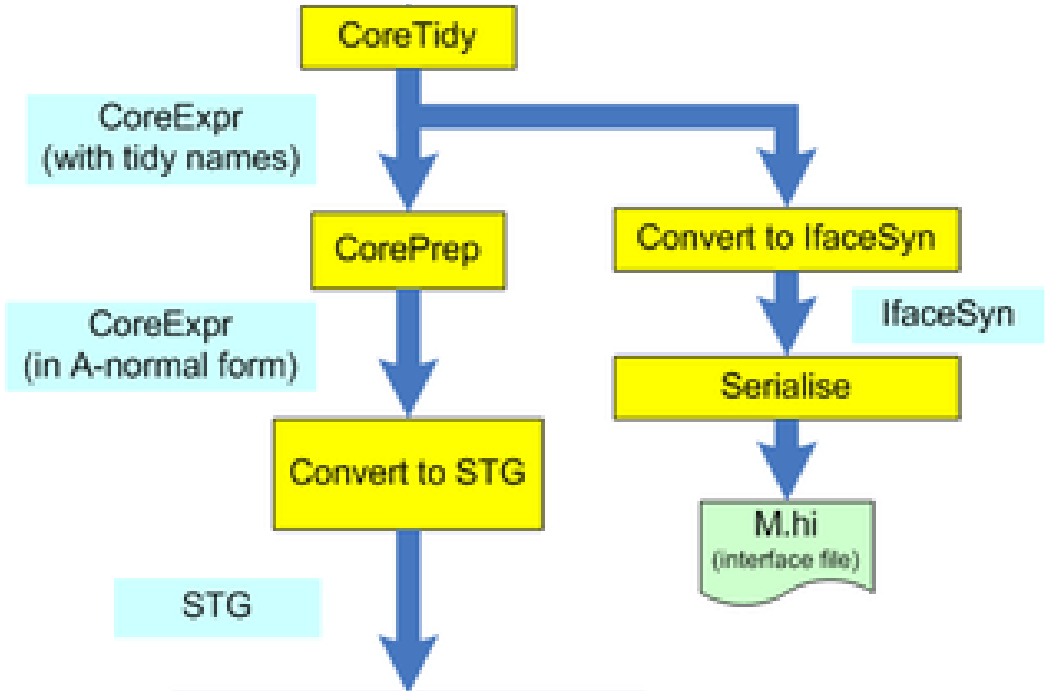
```
f a b = a + b

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



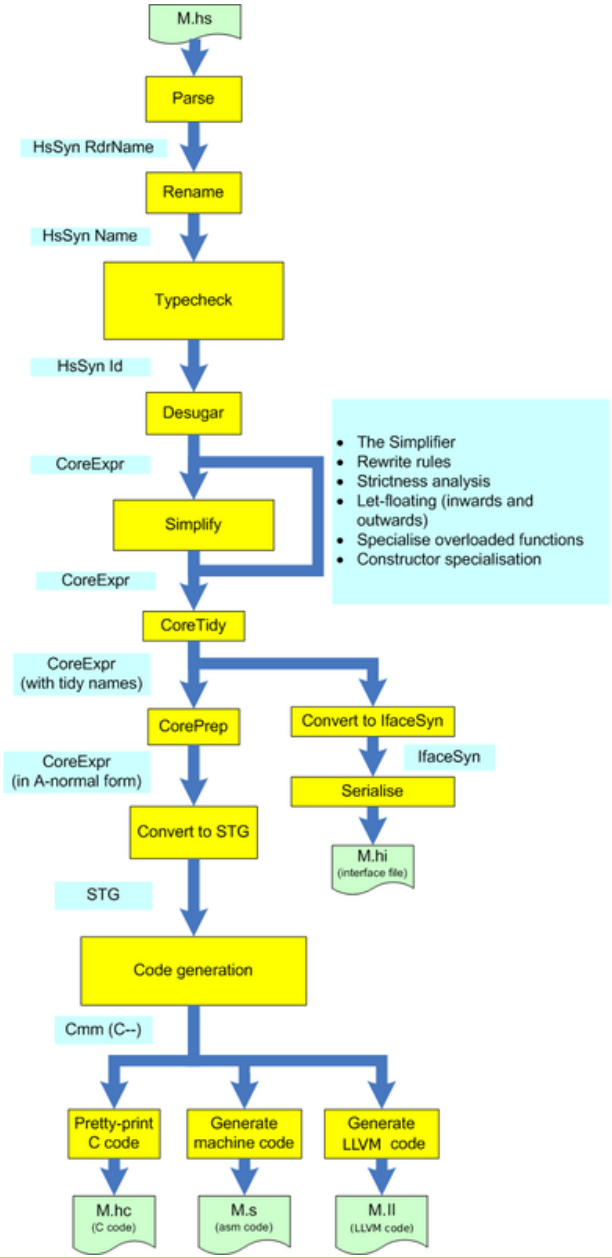


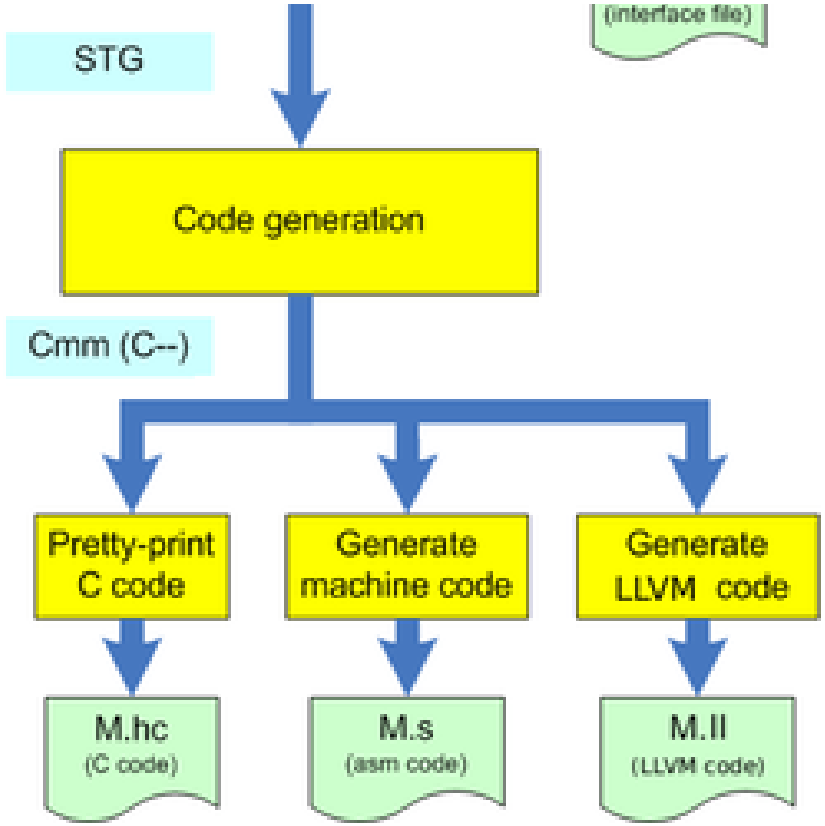
- Построение нормальной формы.
- Генерация интерфейсного файла (для межмодульной оптимизации).
- STG — Spineless Tagless G-machine

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.

Спасибо!