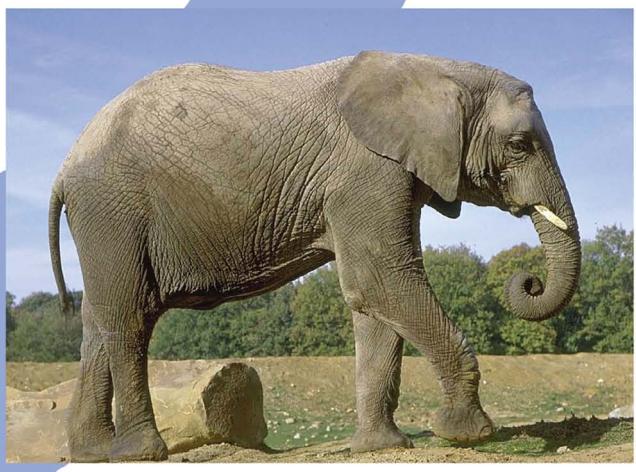
Практика работы на языке Haskell



Душкин Р. В.

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО

УДК 004.4
ББК 32.973.26-018.2
Д86

Душкин Р. В.
Д86 Практика работы на языке Haskell. – М.: ДМК Пресс, 2010. – 288 с., ил.
ISBN 978-5-94074-588-4

В книге рассматриваются прикладные аспекты работы на языке функционального программирования Haskell. Приводятся описания инструментальных средств пяти классов – трансляторов, интегрированных сред разработки, вспомогательных утилит, специализированных библиотек и справочно-архивных систем. Для каждого программного средства приводится краткое описание, его функциональность и примеры использования.

Книга станет хорошим подспорьем как для начинающих программистов, так и для профессионалов, использующих в своей практике функциональную парадигму программирования.

УДК 004.4
ББК 32.973.26-018.2

ISBN 978-5-94074-588-4

Оглавление

Глава 1

Трансляторы

В узком смысле под транслятором понимают программное средство, осуществляющее преобразование программы, написанной на одном языке (входном), в другой (выходной язык), часто являющийся последовательностью машинных команд. Трансляторы делятся на два обширных подкласса — интерпретаторы и компиляторы. Для языка Haskell разработано достаточное количество как первых, так и вторых. Далее описываются два наиболее развитых инструмента:

1) интерпретатор HUGS;
2) компилятор GHC (в поставке также имеется интерпретатор GHCi, который, однако, здесь не рассматривается).

Подробно о трансляторах и технологии их разработки рассказывается в книгах [1, 2]. Также одна глава книги [5] посвящена трансляторам и методам обработки формальных языков на языке программирования Haskell.

1.1 Интерпретатор HUGS

HUGS (точнее, HUGS 98) — это интерпретатор функционального языка Haskell, который сегодня стал стандартом де-факто для нестрогих функциональных языков программирования. Программное средство HUGS 98 представляет практически полное соответствие стандарту языка Haskell-98. Детальное описание программного средства на русском языке дано в методическом пособии [3].

1.1.1 Паспорт программного средства

Характеристика	Значение
Код	HUGS
Наименование	Haskell User's Gofer System
Последняя версия	Сентябрь 2006
Тип	Интерпретатор
Автор	Марк Джонс (Mark P. Jones)
Разработчик	Группа Йельского Университета (Yale Haskell Group)
Совместимость	MacOS, Unix, Windows
Язык	Английский
Размер	14 Мб
Обновления	Очень редко
Состояние	Стабильная поставка
Лицензия	BSD
Документация	Развитая
Web-сайт	`http://www.haskell.org/hugs/`
Аналоги	HBI, GHCi

1.1.2 Общее описание

Интерпретатор HUGS был реализован в 1995 году на основе достаточно старого инструментального средства для работы с функциональной парадигмой программирования Gofer (аббревиатура от англ. *GOod For Equational Reasoning*). Последняя являлась упрощённым вариантом языка Haskell и использовалась исключительно для обучения основам функционального программирования в Йельском Университете. В январе 1999 года интерпретатор HUGS стал практически полностью поддерживать формат языка Haskell-98 (за исключением нескольких незначительных особенностей).

Поскольку интерпретатор HUGS является достаточно простым и малым по размеру программным средством, он часто используется для обучения языку программирования Haskell, для быстрого прототипирования программ, а также для исследовательских целей, то есть в целом там, где нет необходимости в обес-

печении суровых требований по производительности и размеру исполняемого кода.

Несмотря на это интерпретатор HUGS сверх стандарта Haskell-98 поддерживает дополнительные возможности (расширения) языка, а именно:

1) HUGS поддерживает стандартизованные расширения (дополнения) к стандарту Haskell-98 — FFI (интерфейс взаимодействия со сторонними функциями) и структуризацию наименований модулей (использование пакетов модулей, обозначаемых при помощи точки (.)).

2) При помощи включения специальных параметров интерпретатора становится возможным использовать некоторые нестандартные расширения языка Haskell, ряд которых поддерживается и компилятором GHC, а другие специфичны именно для интерпретатора HUGS.

3) В поставке интерпретатора HUGS находится множество дополнительных к стандартным библиотек, которые также поддерживаются другими трансляторами языка (в первую очередь компилятором GHC).

Также в поставке интерпретатора имеется утилита `runhugs`, которая позволяет запускать на исполнение модуль на языке Haskell без входа в программную среду. Эта утилита запускает функцию `main` из заданного модуля и передаёт ей список заданных в командной строке аргументов. Такое использование интерпретатора часто бывает полезным для быстрой работы с исходными кодами в командной строке.

1.1.3 Функциональность и использование

Запуск интерпретатора HUGS выполняется либо с командной строки:

```
hugs {option} {file}
```

либо при помощи запуска программы из графической оболочки операционной системы. В последнем случае параметры запуска и загружаемые файлы могут быть определены как в настройках операционной системы (реестр, переменные окружения), так и в настройках самого интерпретатора (файл инициализации). Кроме того, настройки и список загруженных модулей можно изменять во время

работы с программным средством при помощи графического интерфейса пользователя или специальных команд интерпретатора, вводимых непосредственно в строку исполнения.

После успешного запуска интерпретатора на экран выводится приветственное сообщение примерно следующего вида:

```
--    --  --   --   ----    ---        ------------------------------------------------
||    ||  ||   ||  ||   ||  ||--        Hugs 98: Based on the Haskell 98 standard
||---||  ||--||  ||--||   --||         Copyright (c) 1994-2005
||---||           ---||                World Wide Web: http://haskell.org/hugs
||    ||                              Bugs: http://hackage.haskell.org/trac/hugs
||    ||  Version: Sep 2006           ------------------------------------------------

Hugs mode: Restart with command line option +98 for Haskell 98 mode

Type :? for help
Hugs>
```

Приглашение к вводу команды «`Hugs>`» свидетельствует о том, что текущий загруженный модуль является пустым модулем `Hugs`, который загружается по умолчанию в случаях, когда имена загружаемых модулей не заданы при загрузке программного средства. Если загрузить какой-либо модуль, то наименование последнего из загруженных видно в приглашении.

В командной строке интерпретатора возможно запускать правильные выражения языка Haskell для их исполнения, либо исполнять специальные команды интерпретатора HUGS, каждая из которых начинается с символа двоеточия (:).

Далее приводится перечень команд интерпретатора HUGS, которые можно использовать в его командной строке. При этом надо отметить, что команды можно запускать как при вводе полного их наименования, так и при вводе только первой буквы (например, `:q` вместо `:quit`).

Базовые команды

Запуск на исполнение выражения осуществляется простым вводом этого выражения после приглашения интерпретатора:

```
expression
```

Любое правильное выражение на языке Haskell будет непосредственно выполнено интерпретатором, а результат исполнения будет выведен на экран при помощи стандартной функции `show`. В случае неопределённости типа результата функция `show` будет использовать тип по молчанию, определённый при помощи ключевого слова **default** (если такого определения в загруженных модулях нет, то по умолчанию используется тип `Integer`). Несколько примеров:

```
Hugs> 2 + 3 * 4 - 5
9

Hugs> foldl (*) 1 [1..10]
3628800

Hugs> map snd [(1, 'a'), (2, 'b'), (3, 'c')]
"abc"
```

Если введённое выражение `expression` имеет тип `IO t` для некоторого типа `t`, то все результирующие действия ввода-вывода исполняются, а результат исполнения выражения обычно игнорируется. Например, запуск на исполнение выражения

```
do {print (25^3); putStr "Greetings!"}
```

выведет на экран результаты побочных эффектов функций `print` и `putStr`:

```
15625
Greetings!
```

Следующая команда `:t` запускается с одним параметром:

```
:type expression
```

Данная команда выводит на экран наиболее общий тип выражения `expression`, при этом само выражение не исполняется. Например:

```
Hugs> :t sequence
sequence :: Monad a => [a b] -> a [b]
```

Команда `:s` запускается с набором ключей для установки параметров интерпретатора HUGS, либо пустой:

```
:set {option}
```

Если команда запущена сама по себе, то результатом является вывод всех возможных параметров интерпретатора (режим подсказки), а также значения текущих установок. Перечень возможных параметров представлен в нижеследующей таблице.

Каждый параметр может быть двух видов:

1) Флаг, который либо устанавливается при помощи знака (+) перед ним, либо сбрасывается при помощи знака (−) перед ним соответственно. В таблице ниже все флаги описаны так, как будто бы запущены с символом (+). Все флаги помечены соответствующим образом в графе «Флаг».

2) Параметры со значениями, которые выглядят в общем виде как `−Pstr`. Знак (−) перед символов параметра может быть заменён на символ (+) — это не имеет значения. Строка `str` устанавливается в качестве значения соответствующего параметра.

Параметр	**Флаг**	**Пояснение**
Параметры языка		
98	+	Режим поддержки стандарта Haskell-98 без расширений. По умолчанию флаг взведён, и этот параметр может быть изменён только с командной строки, но не из интерпретатора HUGS в рабочем режиме. По умолчанию параметр включён.
`−cnum`		Устанавливает значение ограничения режима отсечки в заданное число `num` (по умолчанию 40, и этого значения достаточно практически для любых случаев использования HUGS).
`o`	+	(Строчная буква.) Разрешает пересечение экземпляров классов. Работает только в расширенном режиме (параметр −98). По умолчанию параметр отключён.
`O`	+	(Заглавная буква.) Разрешает пересечение экземпляров классов даже в случаях, если это небезопасно. Работает только в расширенном режиме (параметр −98). По умолчанию параметр отключён.

`H`	+	Поддержка строк специального вида, которые записываются в двойных апострофах «‘‘» и «’’». Обычно такие строки являются очень длинными, а потому могут включать в себя символы перевода строки, но не символ (`$`). Последний используется для включения в строку значения параметра функции, в которой используется такая строка. По умолчанию параметр отключён.
Параметры загрузки модулей		
`l`	+	Включение использования по умолчанию литературного кода (изначально эта опция выключена). Файлы с расширением `.hs` всегда понимаются интерпретатором в качестве обычных кодов, в то время как файлы с расширением `.lhs` понимаются как «литературные» коды. Использование этого параметра меняет поведение интерпретатора HUGS в отношении всех прочих файлов.
`.`	+	Распечатывает точки (.) во время процесса загрузки модулей. По умолчанию параметр отключён.
`q`	+	Не печатает ничего во время загрузки модулей. По умолчанию эта опция включена.
`w`	+	Заставляет интерпретатор всегда показывать информацию о том, какие модули загружены. По умолчанию опция выключена.
`-Fcmd`		Запускает внешнюю команду `cmd` для того, чтобы произвести препроцессорную обработку загружаемых модулей перед тем, как загрузить их в интерпретатор. Само собой, с этой командой интерпретатор HUGS уже считывает информацию из стандартного потока вывода запущенного препроцессора. Эта возможность полезна для использования каких-либо расширений языка Haskell, обрабатываемых препроцессором.

−Pstr		Устанавливает набор каталогов для поиска исходных файлов в значение str. В этой строке все каталоги разделяются при помощи двоеточия (:) (в отличие от операционных систем, где обычно используется точка с запятой (;)).
−Sstr		Устанавливает перечень возможных расширений файлов, в которых интерпретатор HUGS осуществляет поиск модулей. По умолчанию этот набор состоит из расширений .hs и .lhs, так что интерпретатор ищет модули в файлах с этими расширениями в каталогах, перечисленных при помощи предыдущего параметра. Опять же, для разделения значений используется двоеточие (:).
Определение редактора исходных файлов		
−Estr		Устанавливает команду для вызова внешнего редактора для редактирования исходных файлов, загруженных в интерпретатор HUGS. В строке str можно использовать специальные символы подстановки: %d заменяется на номер строки исходного файла, на которой произошла последняя ошибка; %s заменяется на имя файла модуля; %f заменяется на полный путь к файлу модуля; %% заменяется на %. Эти символы подстановки можно использовать для управления поведением внешнего редактора.
Параметры вычисления и отображения результатов		
−pstr		Устанавливает строку str в качестве приветствия интерпретатора к вводу значений. Последовательность %s в строке str заменяется на имя текущего модуля.
−rstr		Использует значение str в качестве строки для вызова последней введённой команды. По умолчанию для этих целей используется команда ($$).
k	+	Заставляет интерпретатор HUGS выводить полную информацию об ошибках сортов типов. По умолчанию флаг выключен.

`T`	+	Заставляет в определённых случаях непосредственно использовать тип по умолчанию (определённый при помощи ключевого слова **default**) при вычислении типа выражения при помощи команды `:t`. По умолчанию флаг отключён.
`Q`	+	При выводе каких-либо идентификаторов заставляет интерпретатор HUGS распечатывать квалифицированные имена. По умолчанию опция выключена.
`t`	+	Заставляет интерпретатор выводить на экран тип вычисленного значения. По умолчанию опция выключена.
`u`	+	Использует метод `show` для вывода результатов вычисления на экран. По умолчанию опция включена. Если флаг взведён, то в некоторых случаях интерпретатор HUGS не сможет вывести результат вычисления выражения на экран (например, для большинства функций, поскольку для функциональных типов не определён экземпляр класса `Show`). Если флаг установлен, то результирующее значение выводится в любом случае (иной раз в виде внутренних переменных интерпретатора).
`I`	+	Заставляет интерпретатор HUGS выводить на экран побочные эффекты функций ввода-вывода. По молчанию опция выключена.
Параметры использования ресурсов		
`−hnum`		Устанавливает максимальный размер кучи в памяти, который используется интерпретатором. По умолчанию значение `num` равно 250 кб.
`s`	+	После каждого вычисления любого выражения распечатывает статистику по выполненным вычислениям — количество выполненных элементарных редукция и число задействованных ячеек памяти. По умолчанию опция выключена.

`g`	+	Распечатывает количество ячеек памяти, освобождённое после каждого запуска сборщика мусора. По умолчанию параметр выключен.
`R`	+	Включает оптимизацию, которая немного ускоряет процесс вычислений. По умолчанию параметр включён. Отключение оптимизации может потребоваться при использовании отладчика.

Следующая команда запускает на исполнение главную функцию модуля `main` с заданными параметрами:

`:main {argument}`

Эта команда запускает передаёт в функцию `main` перечень аргументов, которые можно получить при помощи стандартной функции `getArgs`. Команда полезна для тестирования программ в интерпретаторе.

Наконец, команда:

`:quit`

закрывает интерпретатор HUGS и осуществляет выход в операционную систему.

Команды для загрузки и редактирования модулей

Из командной строки интерпретатора HUGS можно загружать модули и вызывать внешнюю программу для их редактирования (с последующей перезагрузкой в интерпретатор).

Команда

`:load [`**module**`|file]`

загружает в память интерпретатора заданный модуль (который может быть обозначен путём указания в параметре имени файла). При этом из памяти интерпретатора выгружаются все ранее загруженные модули, кроме стандартного модуля `Prelude` и вновь загружаемого модуля. Последний загруженный модуль становится текущим модулем. Если не задать имени модуля, то команда загружает текущий модуль.

Когда при помощи этой команды интерпретатору HUGS даётся задание загрузить некоторый модуль `M`, интерпретатор ищет файл `dir/M.hs` или `dir/M.lhs`, где

каталог `dir` входит в множество путей для поиска модулей. Это множество может быть изменено при помощи параметра `−P` команды `:set`. Также и стандартные расширения файлов для поиска модулей могут быть изменены при помощи параметра `−S` той же команды.

В структурные именах модулей (таких, как `A.B.M`) все точки заменяются на обратные слэши (/), а в начало пути, естественно, подставляется компонент `dir`.

В противовес рассмотренной команде команда

`:also` [**module**|`file`]

догружает в память интерпретатора заданный модуль, не выгружая оттуда уже имеющиеся в ней модули. Соответственно, последний загруженный модуль становится текущим.

В этом же ряду стоит команда

`:reload`

которая перегружает текущий модуль, оставляя в памяти интерпретатора только его и стандартный модуль `Prelude`.

Важной является команда

`:edit` [`file`]

которая запускает внешний редактор и загружает в него указанный файл для редактирования. Если имя файла не задано, интерпретатор HUGS загружает во внешний редактор файл, в котором содержится текущий модуль. Необходимо отметить, что команду для запуска внешнего редактора можно поменять при помощи параметра `−E` команды `:set`.

Также полезной является команда

`:find name`

загружающая в редактор модуль из состава текущих загруженных модулей, в котором имеется определение идентификатора `name`.

Команды для получения информации

Не менее важными при работе с интерпретатором HUGS являются справочные команды, несущие больше вспомогательную роль. Так, к примеру, простая команда

`:?`

выводит в консоль интерпретатора краткую информацию об описанных здесь командах (естественно, на английском языке).

Команда

`:names [pattern]`

выводит на экран перечень всех идентификаторов программных сущностей, определённых в текущих загруженных модулях, которые удовлетворяют заданной модели `pattern`. В модели можно использовать символ (∗), который обозначает любую последовательность символов; символ (?), обозначающий любой одиночный символ; [`chars`] — любой из перечисленных в квадратных скобках символ; `char−char` — любой из интервала символов от первого до второго; а также `\char`, обозначающий просто символ `char`.

Если же эту команду выполнить без указания модели идентификатора, то она перечислит все идентификаторы, имеющиеся в текущих модулях.

В противовес этой команда

`:info [name]`

описывает заданную по идентификатору `name` программную сущность. Можно просить интерпретатор описать квалифицированный идентификатор для тех программных сущностей, которые находятся в загруженных в память модулях, но не в текущем. Если никакого идентификатора не задано, команда выводит на экран перечень всех загруженных файлов.

Также и команда

`:browse [all][`**module**`]`

выводит на экран все экспортируемые идентификаторы из указанного модуля **module**. Если указана директива `all`, то на экран выводятся все идентификаторы (не только экспортируемые). Если имя модуля не указано, то, как обычно, используется текущий модуль.

И, наконец, команда

`:version`

выводит на экран информацию о текущей версии интерпретатора HUGS.

Прочие команды

Остаётся рассмотреть ещё три команды, которые невозможно классифицировать каким-либо определённым образом.

`:![command]`

Данная команда выходит в операционную систему, не выгружая из памяти интерпретатор HUGS, и выполняет команду `command`. Если имя команды не указано, то просто осуществляется выход в командную консоль операционной системы.

`:cd dir`

Команда для смены текущего рабочего каталога интерпретатора HUGS на `dir`. Если каталог `dir` начинается с символов `~/`, то тильда (`~`) заменяется на домашний каталог пользователя.

И последняя команда

`:gc`

запускает сборщик мусора.

1.1.4 Другие способы запуска интерпретатора HUGS

В поставке интерпретатора HUGS имеется дополнительная утилита `runhugs`, которую можно использовать для автономного запуска программ на языке Haskell, минуя их загрузку в интерпретатор (само собой, что программы всё равно интерпретируются). Если модуль на языке Haskell является исполняемым, то есть в нём имеется функция `main`, то запуск утилиты

`runhugs {options} file {arguments}`

вызовет на исполнение непосредственно в консоли операционной системы функцию `main` из модуля в файле `file`, при этом функции `main` будут переданы в качестве аргументов все последующие параметры `{arguments}` (они могут быть извлечены в функции `main` при помощи стандартной функции `getArgs`), а сам интерпретатор HUGS будет работать так, как будто бы его настроили при помощи параметров `options`.

1.1.5 Часто задаваемые вопросы

Ниже перечислены ответы на наиболее часто задаваемые вопросы относительно интерпретатора HUGS.

1) *Как правильно называть интерпретатор HUGS?*

   Разработчики используют наименование «HUGS» для обозначения всего класса интерпретаторов, созданных ими. Для конкретизации определённой версии спецификации языка Haskell, которая поддерживается интерпретатором, этот номер версии называется непосредственно после общего наименования: «HUGS 1.3» или «HUGS 98». Для обозначения конкретной версии самого интерпретатора HUGS используется дата выпуска стабильной версии.

2) *Какова связь между программными средствами HUGS и Gofer?*

   Интерпретатор HUGS основан на программном средстве Gofer и изначально был разработан на базе версии 2.30b. При этом интерпретатор HUGS реализует большое количество существенных дополнений, и обеспечивает практически полную совместимость со стандартом Haskell-98.

3) *Кто отвечает за разработку и поддержку интерпретатора HUGS?*

   Наибольший объём кода при реализации интерпретатора HUGS, равно как и первую версию оного, написал Марк Джонс (Mark P. Jones). Тем не менее, сегодня реализация HUGS обязана своим существованием многим программистам и группам разработчиков.

4) *Как сообщить о найденной ошибке?*

   О найденной ошибке можно сообщить при помощи электронной почты `hugs-bugs@haskell.org`. Перед отправкой письма желательно просмотреть перечень известных ошибок и отклонений от стандарта (имеется на официальном сайте интерпретатора, а также в документации).

5) *Как из интерпретатора перейти к определениям программных сущностей в исходных кодах?*

   К сожалению, интерпретатор HUGS предоставляет функциональность только для запуска выражений на вычисления. Для редактирования исходных кодов модулей необходимо пользоваться внешними редакторами

(которые, кстати, могут быть вызваны из рабочей среды интерпретатора при помощи команды `:e`).

6) *Как можно загрузить несколько модулей в одно и то же время?*

Опять же к сожалению, в HUGS имеется возможность загрузить только один модуль для работы с ним, но в этому модуле, естественно, можно использовать неограниченное количество включений других модулей (ключевое слово **import**).

7) *Почему некоторые неэкспортируемые идентификаторы доступны из командной строки интерпретатора?*

Для оптимизации своей работы интерпретатор HUGS использует таблицу идентификаторов, в которую включаются все идентификаторы из модулей, экспортируемых текущим модулем.

8) *Что случилось с функциями `isAlpha`, `toInt` и другими?*

В версии HUGS 98 стандартный модуль `Prelude` приведён в соответствие со стандартом. Раньше в него были включены некоторые дополнительные функции и другие программные сущности. Теперь они перенесены в другие стандартные модули (например, функция `isAlpha` теперь определена в модуле `Char`).

9) *Интерпретатор HUGS поддерживает стандарт Haskell-98 полностью?*

Не совсем. Например, не поддерживаются взаимно рекурсивные модули; имеются и некоторые мелкие несоответствия стандарту (в документации они описаны полностью).

10) *Можно ли при помощи интерпретатора HUGS создать исполняемый файл?*

Поставочный пакет содержит только интерпретатор и специальную утилиту, которая позволяет запускать процесс интерпретации модуля из командной строки (см. подраздел 1.1.4). Других возможностей обработки исходных кодов нет.

1.1.6 Окончательные замечания

Различных способов использования интерпретатора HUGS, его функциональных возможностей и других аспектов достаточно много — можно было бы написать отдельный томик. Детальная документация, правда на английском языке, включена в поставку интерпретатора на прилагаемом компакт-диске. В числе не расмотренных в этом разделе тем:

1) Использование для настройки интерпретатора системных переменных.
2) Добавление в стандартную поставку новых модулей.
3) Компиляция модулей, использующих FFI.
4) Графический интерфейс для операционной системы Windows.
5) Отличия от стандарта Haskell-98.
6) Расширения языка Haskell, поддерживаемые интерпретатором HUGS.
7) Специфичные расширения языка Haskell, реализованные только в интерпретаторе HUGS.
8) История выпуска версий интерпретатора.

1.2 Компилятор GHC

Компилятор GHC (полное наименование незамысловато — *Glasgow Haskell Compiler*, компилятор языка Haskell из Глазго) является свободным программным средством, предоставляющим программисту возможности по компиляции и интерпретации исходных кодов, написанных на языке Haskell. Можно сказать, что сегодня этот компилятор стал практически единственным мощным программным средством для компиляции, поскольку все его «конкуренты» (аналоги) не предоставляют всей функциональности и всех возможностей, которые даёт пользователю GHC.

1.2.1 Паспорт программного средства

Характеристика	Значение
Код	GHC
Наименование	Glasgow Haskell Compiler
Последняя версия	6.10.2
Тип	Компилятор
Автор	Кевин Хаммонд (Kevin Hammond)
Разработчик	Университет Глазго (The University of Glasgow)
Совместимость	MacOS, Solaris, Unix, Windows
Язык	Английский
Размер	36 Мб
Обновления	Довольно частые
Состояние	Постоянная доработка
Лицензия	BSD
Документация	Развитая
Web-сайт	`http://www.haskell.org/ghc/`
Аналоги	HBC, Helium, JHC, NHC, YHC

1.2.2 Общее описание

По своей сути программное средство GHC состоит из двух компонентов — интерпретатора и компилятора, что позволяет программистам двояко использовать его. На деле для запуска того или иного режима используются просто разные параметры, а программный модуль для запуска существует один. Более того, оба режима достаточно тесно интегрированы друг с другом. Например, после компиляции объектные файлы можно загружать в режиме интерпретации, и всё будет работать. А сам режим интерпретации на самом деле предварительно компилирует загружаемый модуль, а потому интерпретация происходит намного быстрее.

В компиляторе GHC реализовано несколько расширений языка Haskell, которые уже далеко отошли от стандарта Haskell 98. Само собой разумеется, что стандарт языка поддерживается компилятором, но дополнительно он предоставляет

такие уже практически ставшие стандартом де-факто расширения, как параллельные вычисления, использование исключений, многопараметрические определения классов, локально-универсальная и экзистенциальная квантификация типов (мощное расширение системы типизации Хиндли-Милнера), функциональные зависимости типов и многие другие. Все такие расширения детально описаны в документации на компилятор GHC.

Также в составе компилятора GHC имеется мощный оптимизатор кода, который позволяет создавать быстрые программы. По умолчанию компилятор и так пытается генерировать код, который исполняется как можно быстрее, но при включении оптимизации получаемые исполняемые файлы могут обнаруживать удивительное быстродействие.

Наконец, в поставке компилятора GHC имеется широчайший набор библиотек, намного превосходящий набор, определённый стандартом языка. Все библиотеки также подробно документированы (по крайней мере, для каждой имеется автоматически сгенерированная при помощи утилиты Haddock документация). К числу таких библиотек относятся, к примеру, следующие: расширенный пакет библиотек для монадических вычислений, просто огромный пакет для работы с многочисленными стандартными и не очень структурами данных, расширенная библиотека для работы с FFI, несколько библиотек для работы с графикой и графическими интерфейсами пользователя, библиотека для сетевого взаимодействия, библиотека для работы со звуком, пакет библиотек для работы с операционной системой на низком уровне, библиотека для создания синтаксических анализаторов и другой обработки текста, а также много других мелких библиотек.

1.2.3 Функциональность и использование

Базовая функциональность и свойства компилятора GHC могут быть описаны следующим:

1) Компилятор GHC полностью поддерживает стандарт языка Haskell-98, а также предоставляет программисту широчайший набор расширений.

2) Это программное средство особенно хорошо поддерживает расширения для использования метода конкурентных и параллельных вычислений, включая использование такого мощного механизма, как программная реализация транзакционной памяти (STM).

3) Компилятор GHC генерирует быстрый исполняемый код, особенно для конкурентных вычислений.

4) Компилятор GHC может использоваться на различных операционных системах, включая семейства Windows и Unix. Также могут быть задействованы различные аппаратные платформы.

5) В компилятор встроены мощные средства оптимизации, включающие в себя возможности оптимизации кода между модулями.

6) Компилятор GHC либо компилирует исходные коды на языке Haskell при помощи предварительной трансляции на язык C с последующей компиляцией посредством компилятора GCC, либо для некоторых платформ непосредственно транслирует исходный код на языке Haskell в машинные коды. Также есть возможность компилировать исходные коды в байт-код, имеется возможность совместного использования скомпилированных программ и программ в байт-кодах.

7) Компилятор GHC предоставляет механизмы профилирования как по времени, так и по занимаемому объёму памяти.

8) Наконец, в поставке компилятора GHC имеется широчайший набор библиотек для решения разнообразнейших задач.

Итак, как уже говорилось, функциональность компилятора GHC можно разделить на два типа — компиляция и интерпретация. Сам по себе компилятор может работать в режиме интерпретатора, для этого его необходимо запустить со специальным параметром командной строки (`--interactive`). Работа в режиме интерпретации ничем не отличается от работы интерпретатора HUGS, даже используются те же команды для управления программным средством. Единственное существенное различие — возможность «на лету» определять новые функции при помощи ключевого слова **let**, используя его непосредственно в командной строке интерпретатора:

```
> let x = (2^4 * 3) * 31
> x
1488
```