Оглавление

Новая парадигма

Внимательное изучение опыта написания и последующего распространения первых книг на русском языке о функциональном языке программирования Haskell [4, 5] подсказывает, что при работе над этими и другими подобными книгами упускается один очень важный момент. Более того, некоторые заинтересованные читатели в своих отзывах явно указывали на эту проблему, предлагая сменить сухой теоретический стиль изложения далёких от практического применения фундаментальных знаний на более приземлённое описание того, что и как можно сделать при помощи функционального программирования и функциональных языков. В книге [4] была произведена попытка сделать это, но, тем не менее, книга вышла несколько оторванной от реальности. Впрочем, такова, по всей видимости, судьба любого справочника.

Вместе с тем ещё в 1998 году один из апологетов функционального программирования Ф. Уодлер написал статью [22] о том, почему функциональные языки программирования не получают широкого распространения. В качестве одной из причин было указано крайне малое количество инструментальных средств, утилит, библиотек и прочих инструментов, позволяющих потенциальному разработчику сесть и начать работать на понравившемся ему функциональном языке программирования. Но вот на дворе XXI век, и к настоящему времени для языка Haskell выпущено огромное количество разнообразнейших инструментов, позволяющих создавать полноценные приложения для любой предметной области. Но почему же интерес к этому языку и его популярность всё ещё не могут быть сравнимы с популярностью таких языков, как Java или C++?

Один из возможных вариантов ответа заключается в том, что кроме замечательных инструментов необходимо создавать методическое обеспечение в должном объёме, которое позволит людям, не имеющим достаточного энтузиазма, на-

чать хотя бы интересоваться проблемой. Но уже упомянутые книги, предоставляющие фундаментальные знания, отпугивают новичков кажущейся на первый взгляд «суровой» математикой. А новичкам интересно найти в одном месте все инструменты, быстро установить их на свой персональный компьютер и написать свою первую программу «`Hello, world!`». Но в книгах и статьях новичков продолжают пугать быстрыми сортировками, оторванными от жизни факториалами и числами Фибоначчи.

Кому это нужно? Нет, несомненно, базовое образование и понимание глубинных основ науки о вычислениях крайне желательно для профессионального программиста. Но всё это может быть навёрстано позже, когда сформируется потребность понимания процессов, происходящих в недрах трансляторов функциональных языков. «Что такое ленивые вычисления?», «Почему ядро языка Haskell представляет собой типизированное λ-исчисление?», «Зачем программисту комбинаторная логика?», «Какие реальные применения в области программирования могут иметь монады?» и т. д. — все эти вопросы заинтересованный функциональным программированием специалист сможет задать себе и своим коллегам намного позже, когда проникнется сутью и духом функционального программирования. И тогда он найдёт всю необходимую теоретическую информацию, которой создано более чем достаточно.

Итак, в деле популяризации функционального программирования объявляется *новая парадигма*. Она заключается в предоставлении читателю конкретной прикладной информации о том, как применять функциональное программирование на практике для разработки программных систем любой сложности. Более того, в её рамках будут отсутствовать многомудрые повествования о морфизмах, теории категорий, β-редукции и прочих подобных вещах. И настоящая книга следует этой парадигме — она описывает существующий инструментарий, который уже сегодня можно взять и использовать на практике. Более того, на прилагаемом к книге CD записаны все программные средства, описываемые в книге, поэтому читатель сможет воспользоваться всеми новыми знаниями непосредственно в процессе чтения.

В добрый путь!

Краткое содержание книги

Данная книга разбита на пять глав, каждая из которых посвящена отдельному классу программных средств, предназначенных для работы с языком Haskell — разработки приложений на нём. Главы посвящены следующим классам инструментария:

1) трансляторам;
2) интегрированным средам разработки;
3) дополнительным утилитам;
4) важнейшим библиотекам языка;
5) справочным системам.

Выбор перечисленных классов программных средств обусловлен кибернетическим пониманием процесса преобразования идеи в разуме разработчика в создаваемое им программное средство, которое создаётся при помощи системы взаимосвязанных инструментов. Этот процесс схематично показан на рис. 1.

Каждая глава делится на разделы, каждый раздел отведён одному продукту одного из подклассов того класса программных средств, которому посвящена глава. Так, к примеру, в первой главе, посвящённой трансляторам, описывается по одному компилятору и интерпретатору. Так сделано сознательно — в книге представлен оптимальный (но не полный, поскольку охватить всё сразу невозможно), на взгляд автора, набор инструментов. В любом случае, в описании каждого конкретного программного средства в обязательном порядке указываются некоторые альтернативные решения, которые каждый читатель сможет найти в сети Интернет.

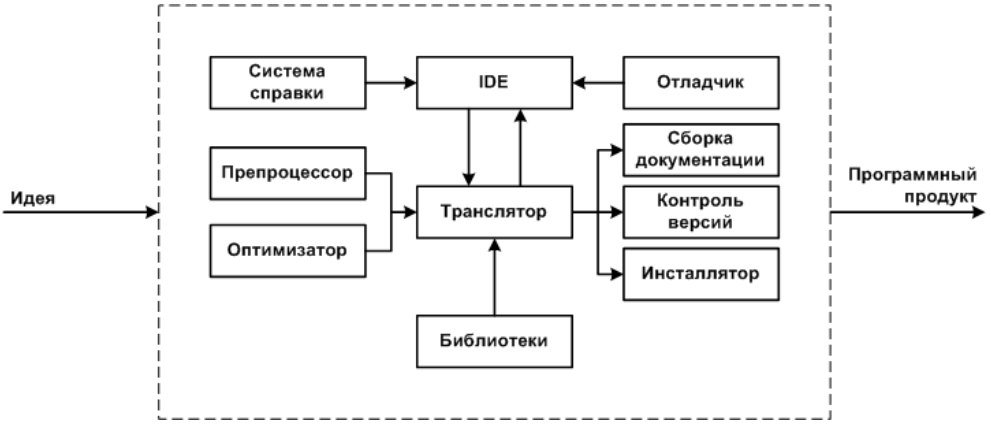

Рис. 1. Кибернетическая схема преобразования идеи в законченное программное средство

Каждое описываемое программное средство рассматривается в следующем ключе. В самом начале идёт так называемый «паспорт программного средства» — перечень значений ключевых характеристик. Этот перечень можно использовать для быстрого ознакомления с основным назначением программного средства, а также для сравнения их друг с другом. На стр. 12 показан пример такого паспорта для интерпретатора HUGS. Все таблицы паспортов унифицированы, другое дело, что в некоторых случаях сравнение программных средств не имеет смысла (пример — компилятор и библиотека для него).

Смысл граф паспорта в большинстве случаев понятен из их названия. В графе «Последняя версия» приводится номер выпуска соответствующего инструментального средства на весну — лето 2009 года. Графа «Язык» показывает язык графического интерфейса пользователя и (или) документации. Графа «Размер» содержит указание на размер инсталляционного пакета под операционную систему Windows. Наконец, в графе «Аналоги» представлены в большинстве случаев лишь некоторые аналоги, которые показались автору вполне достойными конкурентами рассматриваемому средству.

В целях единообразия представления исходных кодов здесь используется определённое форматирование текста, которое специальным образом выделяет структурные элементы функций и других программных сущностей. В отличие от предыдущих публикаций автора при вёрстке этой книги использовался пакет

LaTeX «listings» со специально включённой поддержкой языка Haskell, а потому структурные элементы оформляются в стиле этого пакета.

Таким образом, обычные идентификаторы из программ на языке Haskell при упоминании в тексте книги записываются при помощи моноширинного шрифта: `foldr`, `last`, `Functor`, `fst` и т. д. Ключевые слова, в свою очередь, выделяются полужирным начертанием обычного шрифта: **let**, **class**, **module**. Знаки операций и специальные символы при записи внутри текста ограничиваются круглыми скобками: (//), ($) и т. д., а сами скобки при необходимости выделения в тексте записываются в кавычках: «(», «)».

Отдельные определения программных сущностей оформляются программными блоками с акцентированием ключевых слов, причём сам текст исходных кодов записывается моноширинным шрифтом с небольшой разрядкой:

```
class Pretty a where
  pretty     :: a -> Doc
  prettyList :: [a] -> Doc

  prettyList = list . map pretty
```

К книге прикладывается компакт-диск, на который записаны все описываемые в книге инструменты. Само собой, те из них, которые находятся в активной стадии разработки, могут быть выпущены в новой версии, поэтому читателю рекомендуется внимательно смотреть в паспорте программного средства в графу «частота обновления», на основании записи в которой станет ясно, имеет ли смысл проверять наличие новой версии соответствующего инструмента на его официальном сайте. К имеющимся на компакт-диске инструментам по возможности прикладывается и документация на них.

Для описания представленных в книге программных продуктов, библиотек и инструментов использовались материалы и документация, прилагаемые к соответствующим программам, а также дополнительные статьи и книги, перечисленные в списке литературы.

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