Оглавление

От переводчиков 9

Что называют теорией языков программирования? 13

Благодарности 17

Глава 1. Термы и отношения 19
- 1.1. Индуктивные определения 19
  - 1.1.1. Теорема о неподвижной точке 19
  - 1.1.2. Индуктивные определения 22
  - 1.1.3. Структурная индукция 25
  - 1.1.4. Рефлексивно-транзитивное замыкание отношения 25
- 1.2. Языки 26
  - 1.2.1. Языки без переменных 26
  - 1.2.2. Переменные 26
  - 1.2.3. Многосортные языки 29
  - 1.2.4. Свободные и связанные переменные 29
  - 1.2.5. Подстановка 30
- 1.3. Три способа задания семантики языка 32
  - 1.3.1. Денотационная семантика 32
  - 1.3.2. Операционная семантика с большим шагом 32
  - 1.3.3. Операционная семантика с малым шагом 33
  - 1.3.4. Незавершающиеся вычисления 33

Глава 2. Язык PCF 35
- 2.1. Функциональный язык PCF 35
  - 2.1.1. Программы как функции 35
  - 2.1.2. Функции как объекты первого класса 35
  - 2.1.3. Функции с несколькими аргументами 36

2.1.4. Без присваиваний 36
2.1.5. Рекурсивные определения 36
2.1.6. Определения 37
2.1.7. Язык PCF 37
2.2. Операционная семантика с малым шагом 39
2.2.1. Правила 39
2.2.2. Числа 40
2.2.3. Эквивалентность (congruence) 42
2.2.4. Пример 42
2.2.5. Нередуцируемые замкнутые термы 43
2.2.6. Незавершающиеся вычисления 45
2.2.7. Слияние (confluence) 46
2.3. Стратегии редукции 47
2.3.1. Понятие стратегии 47
2.3.2. Слабая редукция 48
2.3.3. Вызов по имени 49
2.3.4. Вызов по значению 50
2.3.5. Немного лени не помешает 50
2.4. Операционная семантика с большим шагом 51
2.4.1. Вызов по имени 51
2.4.2. Вызов по значению 52
2.5. Вычисление PCF-программ 54

Глава 3. От вычисления к интерпретации 57
3.1. Вызов по имени 57
3.2. Вызов по значению 59
3.3. Оптимизация: индексы де Брауна 60
3.4. Построение функций с помощью неподвижных точек . 63
3.4.1. Первая версия: рекурсивные замыкания 63
3.4.2. Вторая версия: рациональные значения 65

Глава 4. Компиляция 69
4.1. Интерпретатор, написанный на языке без функций . . 70
4.2. От интерпретации к компиляции 71
4.3. Абстрактная машина для PCF 72
4.3.1. Окружение 72
4.3.2. Замыкания 72
4.3.3. Конструкции PCF 73
4.3.4. Использование индексов де Брауна 74
4.3.5. Операционная семантика с малым шагом 74
4.4. Компиляция PCF 75

Глава 5. PCF с типами 79
5.1. Типы . . . 80
5.1.1. PCF с типами . . . 80
5.1.2. Отношение типизации . . . 82
5.2. Отсутствие ошибок во время выполнения . . . 84
5.2.1. Использование операционной семантики с малым шагом . . . 84
5.2.2. Использование операционной семантики с большим шагом . . . 85
5.3. Денотационная семантика для PCF с типами . . . 86
5.3.1. Тривиальная семантика . . . 86
5.3.2. Завершаемость . . . 87
5.3.3. Отношение порядка Скотта . . . 89
5.3.4. Семантика неподвижной точки . . . 90

Глава 6. Вывод типов 95
6.1. Вывод мономорфных типов . . . 95
6.1.1. Присвоение типов нетипизированным термам . . . 95
6.1.2. Алгоритм Хиндли . . . 96
6.1.3. Алгоритм Хиндли с немедленным разрешением . . . 99
6.2. Полиморфизм . . . 101
6.2.1. PCF с полиморфными типами . . . 102
6.2.2. Алгоритм Дамаса—Милнера . . . 104

Глава 7. Ссылки и присваивание 107
7.1. Расширение PCF . . . 108
7.2. Семантика PCF со ссылками . . . 109

Глава 8. Записи и объекты 117
8.1. Записи . . . 117
8.1.1. Помеченные поля . . . 117
8.1.2. Расширение PCF записями . . . 118
8.2. Объекты . . . 122
8.2.1. Методы и функциональные поля . . . 122
8.2.2. Что значит «Self»? . . . 123
8.2.3. Объекты и ссылки . . . 125

Послесловие 127

Библиография 131

Предметный указатель 132

От переводчиков

Теория языков программирования является важным разделом современной теоретической информатики, она активно развивается в работах западных специалистов, ей посвящаются большие конференции, статьи по этой тематике публикуются в ведущих научных журналах. К сожалению, эта теория долгое время оставалась на обочине советской, а затем российской науки, что, видимо, привело к дефициту русскоязычной литературы, в которой излагались бы её основы. Тем не менее, хотелось бы указать несколько исключений.

В первую очередь следует упомянуть сборник переводов под общим названием «Математическая логика в программировании» [1], изданный в 1991 году. В нём были опубликованы статьи таких известных учёных как Дана Скотт, Роджер Хиндли, Саймон Пейтон Джонс и других. Большой вклад в доведение теории языков программирования до российского читателя много позже внесла публикация перевода обширной научной монографии Джона Митчелла «Основания языков программирования» [2], выполненного под научной редакцией Н. Н. Непейводы. Наконец, благодаря труду энтузиастов был издан перевод книги Бенджамина Пирса «Типы в языках программирования» [3].

Настоящая книга представляет собой введение в теорию языков программирования, по содержанию и стилю изложения намного более доступное, чем любая из перечисленных книг. Авторам удалось вместить в очень небольшой объём самые необходимые сведения, удачно соединив вопросы операционной и денотационной семантики программ и теорию типов. В книге также нашлось место достаточному числу упражнений, что исключительно важно для начинающих. В результате данное издание можно рекомендовать для факультативов или семинаров на младших курсах университетов, а также для самостоятельного изучения. Упомянутые выше книги могли бы составить основу для более глубокого знакомства с темой на старших курсах, что обеспечило бы теории языков программирования более достойное место в университетских курсах, чем мы видим сейчас в нашей стране.

Несмотря на выход указанных изданий, в литературе по данному предмету остаётся несколько существенных пробелов, обозначим два из них. В теории языков программирования можно выделить два крупных раздела: теория типов и семантика языков программирования. Упомянутая выше книга Б. Пирса, а верней её перевод, даёт достаточно полное представление о теории типов. К сожалению, нам неизвестно перевода или оригинальной сравнимой по уровню работы на русском языке, посвящённой семантике языков программирования. Достойными кандидатами на эту роль видятся книги Карла Гюнтера или Глинна Винскеля, приведённые в авторском списке литературы в конце книги (пункты 3 и 14 соответственно), если бы они были изданы на русском языке.

Для глубокого изучения теории языков программирования совершенно необходимо знакомство с лямбда-исчислением, которое вполне можно считать тем самым «единственным универсальным» языком программирования, упомянутым авторами во введении к данной книге, но только для специалистов по теории языков программирования. Оно вводится в том или ином объёме в книгах Б. Пирса и Д. Митчелла, но (весьма разумно) обойдено стороной в настоящем издании. Тем не менее, полезно было бы иметь русскоязычное издание, посвящённое целиком этому предмету. Стоит отметить, что в издательстве «Мир» ещё в 1985 году выходил перевод энциклопедической книги выдающегося голландского учёного и специалиста в этой области Хенка Барендрегта «Ламбда-исчисление. Его синтаксис и семантика» [4]. Однако эта работа целиком посвящена бестиповому лямбда-исчислению, в то время как для современной теории языков программирования более важным представляется лямбда-исчисление с типами, великолепно описанное в доступной форме университетского учебника, например, Роджером Хиндли и Джонатаном Селдином [5]. Пример того, как более глубокое изучение вопросов типизации языков программирования часто проводится в рамках лямбда-исчисления, дают пятая и шестая главы настоящей книги, в которых приведены два различных подхода к типизации. Эти два подхода по существу соответствуют «типизации по Чёрчу» и «типизации по Карри», детально рассматриваемым в книге Р. Хиндли и Дж. Селдина.

Ещё один компонент общего образования в информатике, который используется в большинстве книг по теории языков программирования, это функциональное программирование. Прекрасный обзор переводной и оригинальной литературы по этому предмету содержится в статье [6]. Заметим лишь, что в данной книге, как и в издании труда Б. Пирса, для иллюстрации материала используется язык программирования ML, относительно непопулярный в российском академическом сообществе. Нетрудно

выяснить, что в большинстве университетских курсов в России для обсуждения функционального программирования предлагается старейший язык Лисп, которому заметно больше повезло с переводными изданиями в нашей стране (а также их тиражами). Западные университеты довольно давно стали отказываться от Лиспа и его диалектов в пользу ML, а в последнее время популярность завоёвывает более современный язык функционального программирования Haskell, отголоски этого процесса можно видеть и в некоторых отечественных университетах. По всей видимости, если в России и произойдёт смещение интересов университетских преподавателей от Лиспа, то центром притяжения для них станет скорее более актуальный язык Haskell, ML же останется для многих своеобразной «потерянной главой». Этот факт, тем не менее, вряд ли сможет помешать знакомству с теорией языков программирования, потому что в соответствующих книгах обычно используются самые базовые, интуитивно понятные возможности этого языка. Стоит лишь иметь в виду более общее соображение: некоторый опыт работы с функциональными языками весьма полезен для знакомства с темой, которой посвящена настоящая книга.

Малое число переводов неизбежно приводит к терминологическим трудностям, и мы хотели бы сделать несколько замечаний по этому поводу. Отметим, что термин *confluency* мы переводим как *свойство слияния* (в отличие от *конфлюэнтности* и *сходимости*, как в переводах книг Б. Пирса и Д. Митчелла соответственно). Следуя переводу книги Митчелла и избегая при этом дословного перевода использованного в книге термина *standardisation*, свойство гарантированной достижимости нормальной формы терма, при условии её существования, редукцией с вызовом по имени мы называем *полнотой*. Сложный для перевода и обычно либо транслитерируемый, либо переводимый как *отложенный вызов* термин *thunk* у нас переведён как *задумка*, тогда как *редекс*, прочно закрепившийся в русскоязычной литературе ещё с перевода книги Х. Барендрегта, остался транслитерированным. Читателю также будет полезно иметь в виду, что авторы считают *ноль* натуральным числом.

В. Н. Брагилевский и А. М. Пеленицын,
факультет математики, механики и компьютерных наук
Южного федерального университета

Что называют теорией языков программирования?

Человечество пока ещё довольно далеко от создания единственного универсального языка программирования. Новые языки появляются почти каждый день, к уже существующим добавляются новые возможности. Улучшения в языках программирования служат созданию более надёжных программ, сокращают время разработки и упрощают сопровождение. Улучшения необходимы и для удовлетворения новым требованиям, таким как разработка параллельных, распределённых или мобильных приложений.

Определение языка программирования начинается с описания его синтаксиса. Что предпочесть, `x := 1` или `x = 1`? Нужно ли ставить скобки после `if` или нет? И вообще, какую последовательность символов можно считать программой? Для ответа на подобные вопросы имеется полезный инструмент: формальная грамматика. Используя грамматику, можно точно описать синтаксис языка программирования, что, в свою очередь, помогает создавать программы проверки синтаксической корректности других программ.

Однако даже строгое определение синтаксически корректной программы не позволяет предсказать, что именно случится после её запуска. Определяя язык программирования, необходимо также описать его семантику, то есть ожидаемое поведение программы во время исполнения. Два языка могут иметь одинаковый синтаксис, но различную семантику.

Приведём пример того, что неформально понимают под семантикой. Вычисление значения функции обычно объясняют следующим образом. «*Для вычисления результата* V *выражения, записанного в форме* $f\ e_1\ \ldots\ e_n$, *где символ* f *обозначает функцию, определяемую выражением* $f\ x_1\ \ldots\ x_n$ = e', *необходимо: во-первых, вычислить значения* $W_1, \ldots, W_n$ *аргументов* $e_1, \ldots, e_n$, *затем связать эти значения с переменными* $x_1, \ldots, x_n$, *и, наконец, вычислить выражение* e'. *Полученное в итоге значение* V *и есть результат вычисления*».

Такое объяснение семантики, выраженное естественным языком (русским, в данном случае), конечно, даст нам представление о том, что произойдёт во время исполнения программы, но насколько точным это представление окажется? Рассмотрим, к примеру, программу:

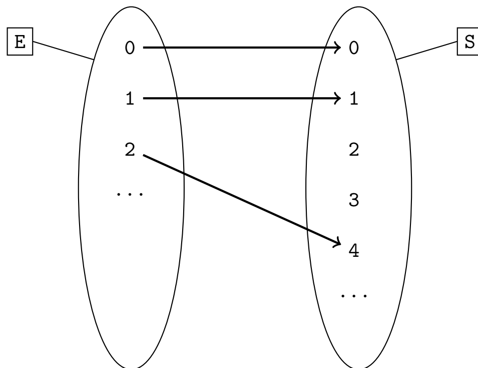
```
f x y = x
g z = (n = n + z; n)
n = 0; print (f (g 2) (g 7))
```

В зависимости от способа интерпретации данного выше объяснения можно установить, что результатом программы будет либо значение 2, либо значение 9. Причина в том, что объяснение на естественном языке не уточняет, следует ли вычислять `g 2` до или после `g 7`, тогда как в рассматриваемой программе порядок вычисления важен. В объяснении следовало бы сказать: «аргументы e_1, ..., e_n вычисляются, *начиная с* e_1», или наоборот, «*начиная с* e_n».

Два программиста, прочитав неоднозначное объяснение, могут понять его по-разному. Хуже того, разработчики компиляторов языка могут выбрать различные соглашения. Тогда одна и та же программа будет давать разные результаты в зависимости от используемого компилятора.

Хорошо известно, что естественные языки слишком неточны для описания синтаксиса языка программирования, вместо них следует использовать формальные языки. В случае семантики ситуация аналогична, для её описания также необходим некий формальный язык.

Так что же такое семантика программы? Возьмём, к примеру, программу `p`, которая запрашивает целое число, вычисляет его квадрат и отображает полученный результат. Чтобы описать поведение этой программы, нам понадобится ввести отношение `R` между входными значениями и соответствующими им результатами.



Теперь можно сказать, что семантикой этой программы является отношение `R` между элементами множества входных значений `E` и элементами множества выходных значений `S`, то есть некоторое подмножество декартова произведения `E` × `S`.

Следовательно, семантика программы является бинарным отношением. В свою очередь, семантика языка программирования это тернарное отношение: «программа `p` с входным значением `e` возвращает выходное значение `s`». Обозначим это отношение следующим образом: `p, e` $\hookrightarrow$ `s`. Программа `p` и вход `e` доступны до начала выполнения программы. Зачастую эти два элемента объединяются в *терм* `p e`, и семантика языка приписывает значение этому терму. В этом случае семантика языка оказывается бинарным отношением `t` $\hookrightarrow$ `s`.

Теперь для выражения семантики языка программирования нам требуется язык, позволяющий описывать отношения.

Если семантика программы является функциональным отношением, то есть для каждого входного значения существует не более одного выходного, будем говорить, что программа является *детерминированной*.

Примерами недетерминированных программ являются компьютерные игры, поскольку для того, чтобы игра доставляла удовольствие, необходим элемент случайности. Язык называется детерминированным, если все программы, которые можно написать на этом языке, детерминированы, или, что то же самое, если его семантика является функциональным отношением. В этом случае семантику можно определить не описанием отношений, а с помощью некоторого языка описания функций.