УДК 004.42 + 519.681.2 + 519.682.1

О логических и алгебраических основаниях классификации формальной семантики программ

Шилов Н.В. (Институт систем информатики СО РАН)

Существует определённый разрыв в уровне математической подготовки программистов-теоретиков и программистов-практиков: первые сильны в науке (абстрактной алгебре и математической логике), а вторые — в искусстве (разработке программных систем). В статье представлен достаточно простой подход к алгебраическим и логическим основаниям формальной семантики программ, ориентированный на инженеров-программистов с элементарными знаниями в абстрактной алгебре и математической логике. Для этого в статье объясняются основы *операционной*, *денотационной*, *аксиоматической* семантики, а также семантики *второго порядка* на примере "эзотерического" языка, синтаксически похожего на язык программирования.

Ключевые слова: Формальная семантика программ, операционная семантика, денотационная семантика, аксиоматическая семантика, семантика второго порядка, эзотерические языки программирования

1. Введение

Мир очень изменился за последние 30 лет. Век информационных технологий сменил век достижений технологий обычных, всюду мобильная связь, компьютеры и Интернет. Но мы обращаемся с "умной" техникой, как с волшебным реквизитом, забывая, что этот "ум" — программы, написанные людьми. Как происходят вычисления в конкретном компьютере? (Ведь в вычислительных машинах нет действительных чисел, даже и рациональных почти нет; есть, к примеру, типы INTEGER и REAL.) И ведь ошибки могут быть не только в вычислениях. Могут ошибаться специалисты, создающие математическую модель процесса или сооружения. Могут быть ошибки в алгоритме решения задачи. Могут быть ошибки в программной реализации алгоритма. Используя выросшие, как на дрожжах, информационные технологии, мы делаем выводы о реальности — а насколько эти выводы верны?

Аварии в энергетике и на транспорте, обрушения зданий и сооружений — похоже, XXI век становится веком техногенных катастроф. И во многом потому, что человечество слиш-

ком небрежно относится к тому аппарату, с помощью которого проектирует свою мощную технику и управляет ею. Программными ошибками были вызваны серьёзные аварии на заводах, катастрофы с самолётами, гибель больных при лечении. Ошибки в программном обеспечении систем управления работой электростанций всех типов, трубопроводов, энергосистем, водоочистных сооружений приводили (и не раз ещё приведут) к очень серьёзным последствиям [1, 2].

Одно из направлений повышение надёжности программного обеспечения — разработка и практическое применение математически точных методов описания "смысла" программ (или, как об этом говорят специалисты, "формальной семантики"). Минуло уже почти полвека с момента публикации Робертом В. Флойдом статьи *Assigning Meanings to Programs* [7], в которой была сделана первая (и удачная) попытка разработать такую семантику. Прошло почти 20 лет после публикации Дэвидом А. Шмидтом призыва к академической общественности [12] сделать формальную семантику программ понятной и доступной широкому кругу инженеров-программистов. За эти годы в академических кругах было разработано множество вариантов формальной семантики, но сложилась такая ситуация с использованием этих формализмов в практике программной инженерии, которую Д.А. Шмидт охарактеризовал словами[1]: "Значительно больше экспертов-теоретиков, чем практикующих программистов".

А между тем академическая активность в попытках обобщить, сделать доступной и полезной формальную семантику, представить свой "взгляд с высоты" нарастает, в неё включаются всё новые учёные.

Например, Петер Д. Мозес в 2001 г. пропагандировал свой подход к многообразию формальных семантик и их использованию [9]. В опубликованной аннотации доклада он написал[2]: "Работа даёт обзор основных парадигм семантики языков программирования... Работа рассчитана на самый широкий круг специалистов по информатике. Она не требует предварительного знакомства с техническими подробностями какой-либо определённой парадигмы, хотя предполагает понимание основных принципов формальной семантики."

В следующем 2002 г. известный учёный Патрик Кузо опубликовал журнальную статью [6], в которой построил иерархию формальных семантик. Эта иерархия включает

[1] more experts, but fewer general users (Перевод Шилова Н.В.)

[2] This paper surveys the main frameworks available for describing the dynamic semantics of programming languages. ... The paper is intended to be accessible to all computer scientists. Familiarity with the details of particular semantic frameworks is not required, although some understanding of the general concepts of formal semantics is assumed. (Перевод Шилова Н.В.)

семантику большого шага, семантику по Плоткину, “демоническую” семантику по Смиту (Smyth’s demoniac) и “ангельскую” реляционную семантику по Хоару (Hoare’s angelic relational), недетерминированную денотационную и просто денотационную семантику по Скотту, обобщённую семантику второго порядка Дейкстры, а также обобщённую семантику частичной/тотальной корректности по Хоару и пр. Все семантики, рассмотренные в этой обширной статье (56 страниц), представлены в терминах[3] “неподвижных точек, связи между семантиками — посредством связок Галуа, каждая из семантик вычисляется посредством некоторой абстрактной интерпретации по отношению к более конкретной семантике с использованием теорем Клини или Тарского о неподвижной точке”.

Но наука не стоит на месте, и одновременно с попытками бросить взгляд с высоты на картину, которую представляют сложившиеся к XXI веку семантические формализмы, продолжают появляться новые семантические парадигмы. Например, так называемая теоретико-игровая семантика Самсона Абрамского и Чих-Хао Лук Онга [5]. Авторы этой семантики утверждают, что этот формализм готов для моделирования и композиционной верификации программного обеспечения.

Но вот на фоне этой бурной академической деятельности по систематизации “старых” и разработке “новых” формальных семантик Дэвид Л. Парнас в январе 2010 г. опубликовал статью “Реальное переосмысление "формальных методов“” [10]. По его мнению, проблема состоит в следующем[4]: “Мы должны подвергнуть сомнению предположения, лежащие в основе хорошо известных современных формальных методов разработки программного обеспечения, чтобы понять, почему они не получили широкого распространения, и что в них следует изменить.” Это пожелание относится и к формальной семантике.

Настоящая статья является расширенным и переработанным вариантом сообщения [4].

2. Привести ум в порядок

Заголовок этой части статьи навеян известным афоризмом Михаила Васильевича Ломоносова: “Математику уж затем любить надо, что она ум в порядок приводит”. Это не случайно, у нас есть веская причина прибегнуть к его авторитету: формальные методы в

[3]... all the semantics are presented in a uniform fixpoint form and the correspondences between these semantics are established through composable Galois connections, each semantics being formally calculated by abstract interpretation of a more concrete one using Kleene and/or Tarski fixpoint approximation transfer theorems. (Перевод Шилова Н.В.)

[4]We must question the assumptions underlying the well-known current formal software development methods to see why they have not been widely adopted and what should be changed. (Перевод В. Кулямина, см. `http://citforum.ru/SE/quality/fm_rethinking/`.)

программировании (и формальная семантика в том числе) выполняют ту же роль — "ум в порядок приводят" в компьютерных науках. Разумеется, математика имеет множество практических приложений, и формальные методы тоже должны найти своё применение в практике программной инженерии. Но роль формальных методов и математики для образования ума тоже имеет немалое значение.

Однако тут мы наталкиваемся на хроническую аллергию программистов-практиков (студентов и инженеров) к формальным методам: программисты-практики считают формальные методы слишком "рафинированными" в теории и неэффективными на практике. На наш взгляд, основа этой устойчивой аллергии — отсутствие элементарных (как в начальной школе) курсов по формальным методам. Общепризнано, что обучение арифметике нельзя начинать с аксиоматики Пеано, сначала решают задачи о сливах, яблоках, карандашах и т.д. Никто не предлагает сразу учить манипуляциям с доказательствами и не предлагает учащимся показать, что в арифметике Пеано доказуемо утверждение

$$\forall x\ \forall y\ \forall z:\ (y \le z \to ((x + y) - z) = (x + (y - z))),$$

для начала учат, как решать задачки вроде следующей: Пете дали 5 яблок, а он отдал 2 яблока Ване; сколько яблок осталось у Пети?[5]

По нашему мнению, аллергия на формальные методы во многом объясняется тем, что эксперты не заботятся о начальном уровне обучения формальным методам. Курсы по формальным методам обычно начинаются с введения понятий *абстрактная машина состояний*, *преобразователь предикатов*, *логический вывод*, *операционная семантика*, *денотационная семантика*, *аксиоматическая семантика* — и никаких элементарных пояснений вроде яблок и карандашей... И никакого простого инструментального средства (наподобие счётных палочек) для знакомства с этими понятиями, а сразу полные версии экспериментальных верификаторов моделей (model checkers), систем поддержки доказательства (proof assistances) или автоматических верификаторов (theorem provers).

Поэтому в следующей части статьи мы приведём пример для первоначального ознакомления с понятиями операционная, денотационная, аксиоматическая семантика и семантика второго порядка; в заключительной части обсудим, как от этого начального уровня перейти к "настоящим" формальным семантикам, а также приведём вариант простого инструментального средства поддержки автоматизации работы с этими семантиками для простых программ.

[5]Между прочим, если Вы думаете, что правильный ответ "3", то ошибаетесь: правильный ответ — "не менее 3".

⟨program⟩ ::= ⟨assignment⟩ | (⟨program⟩) | ⟨program⟩ ; ⟨program⟩ |
| if ⟨condition⟩ then ⟨program⟩ else ⟨program⟩ |
| while ⟨condition⟩ do ⟨program⟩
⟨assignment⟩ ::= ⟨variable⟩:= ⟨expression⟩
⟨condition⟩ ::= ⟨(in)equality⟩ | (⟨condition⟩) | ¬⟨condition⟩ |
| ⟨condition⟩ ∧ ⟨condition⟩ | ⟨condition⟩ ∨ ⟨condition⟩
⟨(in)equality⟩ ::= ⟨expression⟩ = ⟨expression⟩ |
| ⟨expression⟩ < ⟨expression⟩ | ⟨expression⟩ ≤ ⟨expression⟩ |
| ⟨expression⟩ > ⟨expression⟩ | ⟨expression⟩ ≥ ⟨expression⟩
⟨expression⟩ ::= ⟨constant⟩ | ⟨variable⟩ | (⟨expression⟩) |
| ⟨expression⟩ + ⟨expression⟩ | ⟨expression⟩ − ⟨expression⟩ |
| ⟨expression⟩ ∗ ⟨expression⟩

Рис. 1. BNF определение синтаксиса языка TEL

Однако здесь необходимо сказать, что всё-таки есть вполне удачные примеры учебных материалов по формальным методам для начального и пользовательского уровня. Например, в 2010 г. в издательстве БХВ-Петербург вышла учебная монография Юрия Глебовича Карпова "*Model Checking: верификация параллельных и распределённых программных систем*". Эта книга является прекрасным руководством по верификатору моделей SPIN, она учит пользоваться этим инструментальным средством и понимать основы теории верификации конечных моделей последовательно,шаг за шагом, на примере решения увлекательных головоломок.

3. Эзотерический язык TEL

Язык программирования — это любой искусственный язык для автоматической обработки данных на вычислительной машине. Всякий (искусственный или естественный) язык обычно характеризуются своим *синтаксисом*, *семантикой* и *прагматикой*. Синтаксис — это правописание языка. Семантика — это правила придания смысла синтаксически правильным выражениям этого языка. А прагматика — это практика, методы, способы, пути использования грамматически правильных осмысленных (т.е. имеющих семантику) выражений языка.

Так вот, игрушечный язык TEL (Toy Esoteric Language) вообще не является языком программирования, т.к. не предназначен для обработки данных на компьютере. Он пред-

назначен (это его прагматика) для первоначального знакомства с разными видами формальной семантики. А вот синтаксис языка TEL чрезвычайно похож на синтаксис языка программирования. Контекстно-свободный синтаксис языка TEL дан на рис. 1. (Переменные и константы в этом определении — это идентификаторы и целые числа в произвольной фиксированной системе счисления). Однако синтаксис TEL проще объяснить на следующем простом примере правильного выражения, представленном на рис. 2 (которое мы обозначим S для дальнейшего использования).

```
if z<0 then z:= -1
        else (x:= 0 ; y:= 0 ;
              while y≤z do (y:= y + 2*x + 1 ; x:= x + 1) ;
              x:= x - 1)
```

Рис. 2. Пример правильно построенного TEL-выражения (“программы”)

Неформально говоря, правильные выражения языка TEL — это как бы “программы”, построенные из чисел, переменных, арифметических выражений, логических выражений, операторов присваивания при помощи разделителя “;”, условий “if-then-else” и циклов “while-do”.

Неформальная семантика языка TEL может быть описана следующим образом. Так как правильные выражения языка TEL выглядят как программы, то их можно изображать графически в виде блок-схем. (Например, на рис. 3 представлена блок-схема правильного выражения S.) Всякая блок-схема — это граф, вершины которого — операторы присваивания и логические выражения. Договоримся измерять “длину” пути по блок-схеме количеством операторов присваивания на этом пути (т.е. логические выражения не идут в счёт). Тогда условимся считать “смыслом” (семантикой) каждого правильного выражения целое (натуральное) число, равное длине кратчайшего пути по графу блок-схемы этого выражения от начала до конца. (В частности, как это видно из рис. 3, семантика правильного выражения S есть число 1.)

4. Операционная семантика: исполнимая машина состояний

Операционная семантика основана на *трансляции* правильно построенных выражений в *машину состояний* (“механическую” процедуру), действия которой преобразуют её

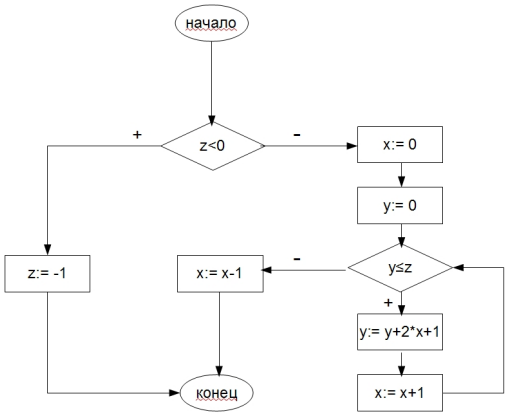

Рис. 3. Блок-схема правильного выражения S

- $F_{\langle \texttt{assignment} \rangle} = 1$;
- $F_{(\beta)} = F_\beta$;
- $F_{\beta;\gamma} = F_\beta + F_\gamma$;
- $F_{\texttt{if } \phi \texttt{ then } \beta \texttt{ else } \gamma} = \min\{F_\beta, F_\gamma\}$;
- $F_{\texttt{while } \phi \texttt{ do } \beta} = 0$,

Рис. 4. Алгоритм трансляции TEL-выражений в арифметические выражения

“состояния”. Например, для языка TEL в качестве такой машины можно принять “арифмометр” — алгоритм вычисления значения арифметического выражения, построенного из целых чисел, операций сложения и выбора минимального из двух значений. Состояния такой машины при вычислении какого-либо такого выражения — это все возможные промежуточные и заключительные стадии вычисления значения этого арифметического выражения. А в качестве алгоритма трансляции — рекурсивный алгоритм F, представленный на рис. 4, преобразующий выражения TEL в арифметические выражения; в описании этого алгоритма β, γ и ϕ — произвольные правильные выражения и условие языка TEL. Для произвольного правильного выражения α этого языка F_α — арифметическое выражение, построенное из целых чисел, операций сложения и выбора минимального из двух значений. Операционная семантика правильного выражения α языка TEL — это значение выражения F_α, которое мы будем обозначать $val(F_\alpha)$.

Например, для правильного TEL-выражения S имеем:

$F_{\texttt{if z<0 then z:=}-1\texttt{ else(x:=0;y:=0; while y}\leq\texttt{z do(y:=y+2*x+1;x:=x+1);x:=x}-1)} =$

$$= \min\{F_{\texttt{z:=-1}},\ F_{\texttt{x:=0;y:=0; while y}\leq\texttt{z do(y:=y+2*x+1;x:=x+1);x:=x-1}}\} =$$
$$= \min\{1,\ F_{\texttt{x:=0}} + F_{\texttt{y:=0; while y}\leq\texttt{z do(y:=y+2*x+1;x:=x+1);x:=x-1}}\} =$$
$$= \min\{1,\ 1 + F_{\texttt{y:=0}} + F_{\texttt{while y}\leq\texttt{z do(y:=y+2*x+1;x:=x+1);x:=x-1}}\} =$$
$$= \min\{1,\ 1+1+\dots\}.$$

Таким образом, выражению S языка TEL сопоставлено арифметическое выражение $\min\{1, 1+1+\dots\}$. Наш алгоритм-аримометр, запущенный на этом выражении, выполнит несколько сложений и одну операцию минимизации и остановится (очевидно) с заключительным значением 1; следовательно, операционная семантика $val(F_S)$ равна 1. Это заключительное значение и все промежуточные выражения, которые получались во время этого вычисления, — состояния нашей машины, которые она проходит при вычислении этого значения $val(F_\alpha)$.

Здесь уместно заметить, что наличие семантики для языка позволяет определить эквивалентность выражений: эквивалентные выражения — это выражения, у которых одинаковая семантика. (Разумеется, если определено несколько семантик, то отношения эквивалентности на выражениях могут не совпадать.) Так, для определённой выше операционной семантики языка TEL эквивалентным будут все выражения, в которых совпадают длины кратчайших (по числу присваиваний) путей от начала до конца в блок-схемах. В частности, наше выражение S (в рассмотренной операционной семантике) эквивалентно любому выражению, в котором есть путь от начала до конца по графу его блок-схемы, содержащий ровно одно присваивание, и всякий путь от начала до конца содержит хотя бы одно присваивание.

5. Денотационная семантика: алгебра для вычислений

Алгебра — это множество элементов и операций над ними. Например, множество натуральных чисел $\mathbb{N}$ с нуль-местными операциями (константами) 0 и 1, бинарными операциями “+” и “–” — это алгебра. А то же множество $\mathbb{N}$ с теми же константами 0 и 1, операцией “+”, но с бинарной операцией min вместо “–” — это уже другая алгебра, т.к. использует другой набор операций.

Денотационная семантика — это определение семантики языка в терминах подходящей алгебры путём согласованного сопоставления каждому правильному выражению языка элемента алгебры, а каждому конструктору правильных выражений — алгебраической

- $[\![\langle \texttt{assignment} \rangle]\!] = 1$;
- $[\![(\alpha)]\!] = [\![\alpha]\!]$;
- $[\![\alpha; \beta]\!] = [\![\alpha]\!] + [\![\beta]\!]$;
- $[\![if\ \phi\ then\ \alpha\ else\ \beta]\!] = \min\{[\![\alpha]\!]\ [\![\beta]\!]\}$;
- $[\![while\ \phi\ do \ldots]\!] = 0$.

Рис. 5. Определение денотационной семантики языка TEL

операции; часто такое сопоставление (функцию) обозначают посредством двойных квадратных скобок $[\![\ ]\!]$, а элемент алгебры или операция алгебры, сопоставленные функцией $[\![\ ]\!]$ правильному выражению или конструктору выражений, называют *денотантом* этого выражения или, соответственно, *денотацией* конструкции. Например, для определения денотационной семантики нашего игрушечного языка TEL возьмём следующую алгебру натуральных чисел $\mathbb{N}$ с константами 0, 1, унарной *тождественной* операцией $\lambda x.x$, бинарными операциями сложения (+) и минимизации (min), а функцию $[\![\ ]\!]$ определим так, как это показано на рис. 5

В частности, для правильного выражения S имеем:

$[\![$`if z<0 then z:= -1`
`else (x:= 0 ; y:= 0 ;`
`while y≤z do (y:= y + 2*x + 1 ; x:= x + 1) ;`
`x:= x - 1)`$]\!] =$

$= [\![$`if` $-$ `then...else...`$]\!]$
$([\![$`z:= -1`$]\!], [\![$`x:= 0 ; y:= 0 ;`
`while y≤z do (y:= y + 2*x + 1 ; x:= x + 1);`
`x:= x - 1`$]\!]) =$

$= \min\{\ 1\ , [\![;]\!]([\![$`x:= 0`$]\!], [\![$`y:= 0 ;`
`while y≤z do (y:= y + 2*x + 1 ; x:= x + 1);`
`x:= x - 1`$]\!])\} =$

$= \min\{\ 1\ , 1 + [\![;]\!]([\![$`y:= 0`$]\!],$
$[\![$`while y≤z do (y:= y + 2*x + 1 ; x:= x + 1) ;`
`x:= x - 1`$]\!])\} =$

$= \min\{\ 1\ , 1 + (1 + \ldots)\} = 1.$

Совпадение $val(F_S)$ и $[\![S]\!]$ не случайно: операционная и денотационная семантики TEL

Присваивание (Assignment): $\dfrac{}{1{\leq}x{:=}t{\leq}1}$	Цикл (Loop): $\dfrac{}{0{\leq}\texttt{while}...\texttt{do}...{\leq}0}$
Блок (Block): $\dfrac{m{\leq}\alpha{\leq}n}{m{\leq}(\alpha){\leq}n}$	Ослабление ограничений (Stretching): $\dfrac{m'{\leq}\alpha{\leq}n'}{m{\leq}\alpha{\leq}n}$, $m \leq m'$, $n' \leq n$
Композиция (Composition): $\dfrac{m'{\leq}\alpha{\leq}n' \qquad m''{\leq}\beta{\leq}n''}{m{\leq}\alpha;\beta{\leq}n}$, $m = m' + m''$, $n = n' + n''$	
Then-ветвление: $\dfrac{m{\leq}\alpha{\leq}n \qquad m{\leq}\beta{\leq}\infty}{m{\leq}\texttt{if}...\texttt{then}\ \alpha\ \texttt{else}\ \beta{\leq}n}$	Else-ветвление: $\dfrac{m{\leq}\alpha{\leq}\infty \qquad m{\leq}\beta{\leq}n}{m{\leq}\texttt{if}...\texttt{then}\ \alpha\ \texttt{else}\ \beta{\leq}n}$

Рис. 6. Аксиоматическая семантика языка TEL

тесно связаны.

Утверждение 1. *$val(F_\alpha) = [\![\alpha]\!]$ для любого правильного выражения α языка TEL.*

Доказательство легко выполнить индукцией по структуре выражения α. ■

Обычно, когда можно доказать свойство "совпадания" двух семантик (наподобие равенства в утверждении 1), говорят о *(взаимной) корректности*. Поэтому мы будем называть утверждение 1 теоремой о взаимной корректности операционной и денотационной семантики языка TEL. Заметим однако, что в общем случае две семантики не обязательно будут взаимно корректны и могут быть связаны друг с другом более сложным образом или даже вообще не связаны.

6. Аксиоматическая семантика: синтаксически управляемое доказательство

Аксиоматическая семантика — это синтаксическое исчисление для вывода ("доказательства") новых утверждений ("теорем") по правилам вывода из заранее определённых аксиом ("фактов"). (Для того, чтобы пользоваться аксиоматической семантикой, надо иметь некоторые навыки построения доказательства в аксиоматических системах.)

В частности, аксиоматическая семантика языка TEL имеет дело с утверждениями вида $m \leq \alpha \leq n$, где m — натуральное число, α — правильное выражение языка TEL, а n — или натуральное число такое, что $m \leq n$, или символ бесконечности ∞. Аксиоматическая семантика для языка TEL представлена на рис. 6 в виде аксиоматической системы. Будем

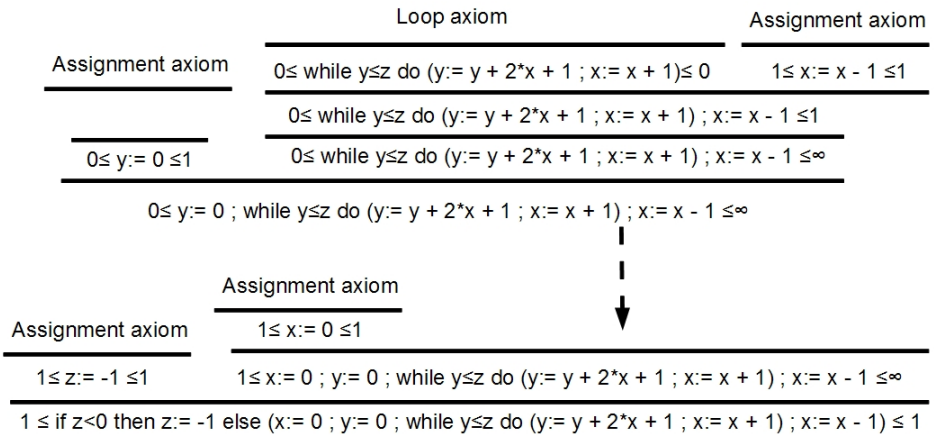

Рис. 7. Пример доказательства в аксиоматической семантике TEL

говорить, что утверждение $m \le \alpha \le n$ доказуемо и писать $\vdash m \le \alpha \le n$, если существует доказательство этого утверждения в этой аксиоматической системе.

Будем говорить, что утверждение $m \le \alpha \le n$ верно (или истинно) и писать $\models m \le \alpha \le n$, если $m \le [\![\alpha]\!] \le n$. Говорят, что аксиоматическая семантика *непротиворечива*, если любое доказуемое утверждение верно; говорят, что аксиоматическая семантика *полна*, если всякое верное утверждение доказуемо. Пример верного и доказуемого утверждения — это $1 \le S \le 1$: его истинность следует из того, что $[\![S]\!] = 1$, а доказуемость подтверждается доказательством, приведённым на рис. 7.

Совпадение $\models 1 \le S \le 1$ и $\vdash 1 \le S \le 1$ не случайно.

Утверждение 2. *Для любых $n, m \in \mathbb{N} \cup \{\infty\}$ и правильного выражения α языка TEL*

$$\models m \le S \le n \;\Leftrightarrow \vdash m \le S \le n.$$

Доказательство непротиворечивости легко получить индукцией по высоте дерева вывода, а доказательство полноты — индукцией по структуре правильного выражения. ■

Заметим опять же, что в общем случае аксиоматическая семантика не обязательно будет полной; она также может быть противоречивой (то есть некоторые доказуемые утвер-

ждения будут неверными).

7. Семантика второго порядка: преобразователи предикатов

Пусть D — произвольное множество, а $k \geq 0$ — произвольное целое число. Тогда k-местный предикат на множестве D — это произвольное подмножество D^k. В частности, одноместный предикат — это некоторое множество значений (элементов) D. Очевидно, что множество всех одноместных предикатов — это множество 2^D всех подмножеств D.

Преобразователь (одноместных) предикатов (на D) — это произвольная функция $F : 2^D \to 2^D$, то есть функция, которая преобразует "входной" предикат $S \subseteq D$ в "выходной" предикат $F(S) \subseteq D$.

Пусть D — некоторое множество ("область"). Элементы D и векторы с компонентами из D (то есть элементы пространства $D_1 = \bigcup_{k\geq 1} D^k$) называют элементами *первого порядка* над D. Подмножества D_1 (и, в частности, все функции из D^m в D^n) называются множествами (предикатами) первого порядка над D (функциями соответственно). Всякая "теория", изучающая или описывающая свойства элементов первого порядка, называется теорией первого порядка области D.

Подмножества D_1 и векторы, компонентами которых являются подмножества D_1 (то есть элементы пространства $D_2 = \bigcup_{k\geq 1}(2^{D_1})^k$), называют элементами *второго порядка* над D. Подмножества D_2 и функции из D_2 в D_2 называются множествами (предикатами) второго порядка и функциями второго порядка (*функционалами*) над D. Всякая "теория", изучающая или описывающая свойства элементов второго порядка, называется теорией второго порядка области D.

(Подобным образом можно определить элементы *элементы*, *предикаты*, *функции* и *теории* третьего и высших порядков.)

Рассмотрим с точки зрения классификации теорий по порядкам такие учебные предметы, как элементарная алгебра, изучаемая в старших классах средней школы, и математический анализ, изучаемый на первых курсах вузов.

Элементарная алгебра занимается решением (систем) уравнений и неравенств в вещественных числах. Эти уравнения заданы многочленами с параметрическими коэффициентами, которые обозначают опять же вещественные числа. Например, типичной задачей элементарной алгебры является нахождение всех вещественных корней квадратного уравне-

ния с вещественными коэффициентами $a \times x^2 + b \times x + c = 0$. В школьном курсе доказывается, что для любых вещественных значений коэффициентов уравнение

$$a \times x^2 + b \times x + c = 0$$

имеет решение в вещественных числах тогда и только тогда, когда

$$b^2 - 4 \times a \times c \geq 0,$$

то есть, что предложение

$$\forall a\ \forall b\ \forall c : \big(b^2 - 4 \times a \times c \geq 0 \leftrightarrow \exists x : (a \times x^2 + b \times x + c = 0)\big).$$

принадлежит теории первого порядка поля вещественных чисел. Можно предположить, что курс элементарной алгебры изучает теорию первого порядка поля вещественных чисел.

Математический анализ занимается изучением свойств непрерывных и дифференцируемых функций вещественной переменной. Примером может служить теорема о промежуточном значении непрерывной функции на замкнутом отрезке, авторство которой приписывается К.Т.В. Веерштрассу, Б. Больцано и О.Л. Коши (см. рис. 8): для любой функции f, непрерывной на отрезке вещественных чисел $[a..b]$, для любого числа $y \in [f(a)..f(b)]$ найдётся такое число $x \in [a..b]$, что $f(x) = y$.

Вспомним, что всякая вещественная функция — это множество пар чисел. Поэтому теорема может быть сформулирована в терминах логики второго порядка над полем вещественных чисел следующим образом:

$$\forall f \in 2^{R \times R}\ \forall a \in R\ \forall b \in R \forall c \in R\ \forall d \in R\ \forall y \in R :$$
$$(C(f,a,b)\ \wedge\ (a,c) \in f\ \wedge\ (b,d) \in f\ \wedge\ c \leq y \leq d\ \rightarrow$$
$$\rightarrow\ \exists x \in R : (a \leq x \leq b\ \wedge\ (x,y) \in f)),$$

где C — трёхместное отношение “быть непрерывной вещественной функцией на отрезке”. Поэтому можно сказать, что курс математического анализа формализуем как теория второго порядка поля вещественных чисел.

Функциональный анализ изучает свойства функционалов, то есть отображений, которые сопоставляют функциям некоторые числа или функции. Часто изучаются свойства линейных операторов, к которым относятся, например, дифференцирование и интегрирование вещественных функций. Поэтому часть функционального анализа, изучающая теорию линейных операторов на функциях вещественной переменной, может быть формализована как теория третьего порядка поля вещественных чисел.

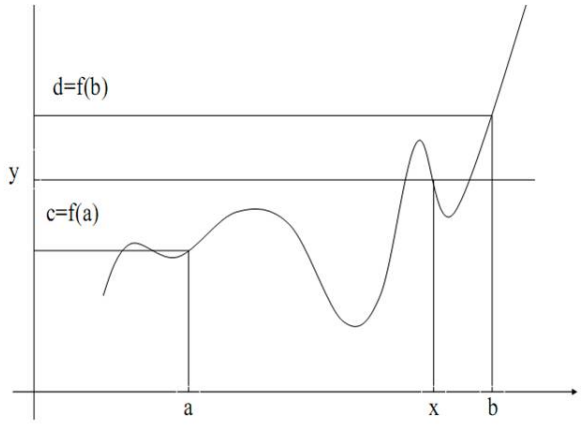

Рис. 8. Иллюстрация к теореме Веерштрасса – Больцано – Коши

Но вернёмся к формальной семантике: семантика второго порядка — это сопоставление каждому синтаксически правильному выражению некоторого преобразователя предикатов (первого порядка) над некоторой "семантической" областью D, то есть функции $F : 2^D \to 2^D$, которая преобразует "входной" $S \subseteq D$ предикат — в "выходной" предикат $F(S) \subseteq D$. В частности, для любого выражения α языка TEL определим преобразователь предикатов PT_α на натуральных числах $\mathbb{N}$ следующим образом: для любого множества натуральных чисел S пусть $PT_\alpha(S)$ — это множество таких натуральных чисел $n \in \mathbb{N}$, что для любого правильного выражения β языка TEL, если $[\![\beta]\!] = n$, то $[\![\alpha;\beta]\!] \in S$. Следующее утверждение устанвливает связь семантики второго порядка с денотационной семантикой языка TEL и является очевидным.

Утверждение 3. *Для любого выражения α языка TEL и множества натуральных чисел $S \subseteq \mathbb{N}$ имеет место равенство $PT_\alpha(S) = S \ominus [\![\alpha]\!]$, где операция $\ominus$ — операция поэлементного вычитания без перехода через 0.*

8. Заключение

Итак, в чем же польза формальной семантики языков программирования? С одной стороны, автор утверждает, что изучение её ум в порядок приводит, но тут же пишет, что хороших курсов для начального обучения нет. Кроме того, автор однозначно согласны с мнением Дэвида Парнаса[6], что "исключительно редкое применение формальных методов в производственной разработке ПО только подчеркивает, что использование таких методов не стало установившейся практикой". О чём же тогда эта статья, о какой пользе?

[6] Applications of formal methods to industrial practice remain such exceptions that they confirm that the use of formal methods is not common practice. (Перевод В. Кулямина, см. `http://citforum.ru/SE/quality/fm_rethinking/`.)

Она — всё-таки о пользе формальных методов (формальной семантики в частности) для образования и дисциплины ума программистов. Но для того, чтобы эту пользу ощутить, необходимо делать следующее.

Нельзя оглоушивать новичков непонятными и непривычными терминами, теориями и методами, обещая, что если они одолеют все эти новые понятия, то в результате создадут полностью автоматический универсальный промышленный верификатор программ.

Надо подготовить систему курсов по формальным методам, начиная с элементарных курсов с игрушечными примерами и простыми (до тривиальности) объяснениями, что есть что в формальных методах (как, например, это сделано в настоящей статье с понятиями денотационная и аксиоматическая семантика).

Нельзя новичков учить пользоваться современными экспериментальными системами, поддерживающими конкретный формализм и нотацию: при таком обучении происходит не образование ума, а только накопление навыков продвинутого пользователя одной конкретной системы.

Для начинающих надо разрабатывать специальные простые и понятные по своей архитектуре и используемым алгоритмам (хотя не очень эффективные) инструментальные средства, как, например, верификатор F@BOOL@ для модельного языка программирования NIL [3].

Верификатор F@BOOL@ использует булевские решатели в качестве средства доказательства теорем, а язык программирования NIL имеет виртуальную машину, реализационную семантику (трансляцию в виртуальную машину), структурную операционную семантику (аксиоматическую систему доказательства возможности вычисления), денотационную семантику (в алгебре бинарных отношений) и аксиоматическую семантику (для утверждений частичной корректности). Доказана корректность и полнота реализации операционной семантики на виртуальной машине, доказано свойство взаимной корректности операционной и денотационной семантики, а для аксиоматическая семантики — полнота и непротиворечивость. Для расширений языка NIL предложен метод трансформационной семантики (реализация новых конструкций средствами ядра языка).

А как быть с практической пользой от формальных методов, экономическим эффектом и т.д.? Неужели нельзя добиться отдачи, и вся польза навсегда останется "виртуальной"?

На наш взгляд, о практической пользе формальных методов можно вести речь. Только для этого, по-видимому, надо ориентироваться не на создание универсальных фор-

мальных методов (и семантик), а на проблемно-ориентированные формальные методы (и семантики). Опыт показывает, что как только специалисты пробуют разработать универсальную формальную семантику языка программирования, то дело не доходит до практического её использования в силу её громоздкости. А специализированные, проблемно-ориентированные формальные методы и семантики программ находят практическое применение. Наиболее яркий пример такого сорта — это статические анализаторы программ, основанные на использовании упрощённой (обычно посредством абстрактной интерпретации) семантики того или иного вида.

Относительно новый пример перспективной проблемно-ориентированной денотационной семантики программ с указателями — это так называемое исчисление алиасов (Calculus of Aliasing) Бертрана Мейера [8]. Эта денотационная семантика предназначена для обнаружения программных дефектов или ошибок, обусловленных тем, что несколько разных выражений указывают на одну область памяти. Аксиоматическая семантика для этих же целей известна под именем логики отделимости (Separation Logic), она разрабатывается уже более 15 лет и имеет несколько экспериментальных реализаций. Её основы были заложены Джоном С. Рейнольдсом с соавторами [11]. Вопрос о связи этих двух формальных семантик для программ с указателями открыт, хотя в работе [13] сделана попытка построить исчисление алиасов для языка программирования, используемого для задания семантики логики отделимости.

Список литературы

1. Аджиев В. Мифы о безопасном ПО: уроки знаменитых катастроф // Открытые системы. 1998. N 6. [Электронный ресурс]. URL: `http://www.osp.ru/os/1998/06/179592` (дата обращения: 20.12.2015).
2. Живич М., Каннингэм Р. Истинная цена программных ошибок // Открытые системы. 2009. N 3. [Электронный ресурс]. URL: `http://www.osp.ru/os/2009/03/8158133/` (дата обращения: 20.12.2015).
3. Шилов Н.В. Пример верификации в проекте FBOOL, основанном на булевских решателях // Моделирование и анализ информационных систем. 2010. Т. 17, N 4. С.111-124.
4. Шилов Н.В., Шилова С.О. О преподавании логических и алгебраических оснований формальной семантики программ // Информатикии информационные технологии в образовании: теория, приложения, дидактика. Материалы Всероссийской научной школы-конференции с международным участием. Новосибирск: ФГБОУ ВПО НГПУ, 2012. Т. 2. С.64-71.
5. Abramsky S., Ghica D.R., Murawski A.S., and Ong C.-H. L. Applying game semantics to compositional software modeling and verification. // Proceedings of 10th International Conference

"Tools and Algorithms for the Construction and Analysis of Systems" TACAS-2004. Lecture Notes in Computer Science. Springer. 2004. Vol. 2988. P. 421-435.

6. Cousot P. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. // Theoretical Computer Science. Springer. 2002. Vol. 277, N 1-2. P. 47-103.
7. Floyd R.W. Assigning Meanings to Programs // Proc. Symp. Applied Mathematics. Am. Math. Soc. 1967. P. 19-31.
8. Meyer B. Steps Towards a Theory and Calculus of Aliasing // International Journal of Software and Informatics. 2011. Vol., N 1-2. P. :77-115.
9. Mosses P.D. The Varieties of Programming Language Semantics And Their Uses // Perspectives of System Informatics. Lecture Notes in Computer Science. Springer. 2001. V. 2244. P. 165-190.
10. Parnas D.L. Really Rethinking "Formal Methods" // Computer (IEEE Computer Society). 2010. Vol. 43, N 1. P. 28-34.
11. Reynolds J.C. Separation Logic: A Logic for Shared Mutable Data Structures // IEEE Symposium on Logic in Computer Science. 2002. P. 55-74.
12. Schmidt D.A. On the Need for a Popular Formal Semantics // ACM SIGPLAN Notices. 1997. Vol. 32. P. 115-116.
13. Shilov N.V., Satekbayeva A., Vorontsov A.P. Alias calculus for a simple imperative language with decidable pointer arithmetics // Bulletin of the Novosibirsk Computing Center. 2014. Vol. 37. P.131-148.