От алгебр программ к алгебраическим вычислениям

Непейвода Н.Н.

Аннотация

Строится абстрактная система алгебр, описывающих программы и вычислительные системы. Показывается, как от систем переходят к алгебрам, от алгебр — к декомпозициям, от декомпозиций алгебр — к алгебраическим вычислениям. Рассматриваются алгебраические аналогии для понятий программирования. Описываются некоторые представления структур данных для алгебраических программ, реализуемых на кристаллических элементах. В частности, рассматриваются представления действительных чисел, позволяющие распараллелить сложение.

Abstract

The algebraic formalism for programs and systems (General Algebraic Program Structures, GAPS) is considered. It is shown how to describe some control and data concept in algebraic terms. In particular some representations of data to compute through crystallographic elements are given. A new representation of real numbers (Brouwer systems) is introduced and studied. Theorems substantiating complexity of computations in Brouwer systems are proven.

Введение

Реверсивные вычисления явились толчком, который привел к появлению более общей концепции алгебраических вычислений. Более точно, к этому привел когнитивный диссонанс между блестящими идеями фон Неймана, Ландауэра, Тоффоли, Фейнмана, Меркле и более чем тридцатилетним застоем (вернее. еще хуже: тридцатилетним бегом по кругу) в их реализации. Известно, что если направление слишком долго является новым и перспективным (см., например, термояд и квантовые компьютеры), то либо в его основе что-то не так (гладко было на бумаге, да забыли про овраги), либо опять же в основе всех работ торчит и мешает всем некая священная корова [4], которая на самом деле на виду у всех, но к ней привыкли либо по другим причинам предпочитают не замечать, что она мешается и гадит. Такими священными коровами в данной области явились двоичное представление данных, традиционная концепция вычислений с памятью и группы.

Двоичная система потребует более детального рассмотрения ниже, а понятия стены памяти и группы разберем сейчас.

Ландауэр и фон Нейман установили из термодинамических соображений нижний предел затраты и выделения энергии при выработке одного бита информации (предел Ландауэра) $E_{diss} = k_B \times T \times \ln 2$ Дж, где T — абсолютная температура, k_B — константа Больцмана и теоретически установил, что в случае обратимости вычислений его возможно преодолеть (выделенная энергия поглощается на месте, а термодинамика для обратимых процессов не работает). Эксперимент, поставленный в 2012 году группой авторов [5], показал возможность преодоления

предела Ландауэра в сотни раз при обратимости вычислений, но при замедлении вычислений в те же сотни раз (рис. 1). Заодно заметим, что в этом эксперименте использовались не двоичные, а троичные элементы.

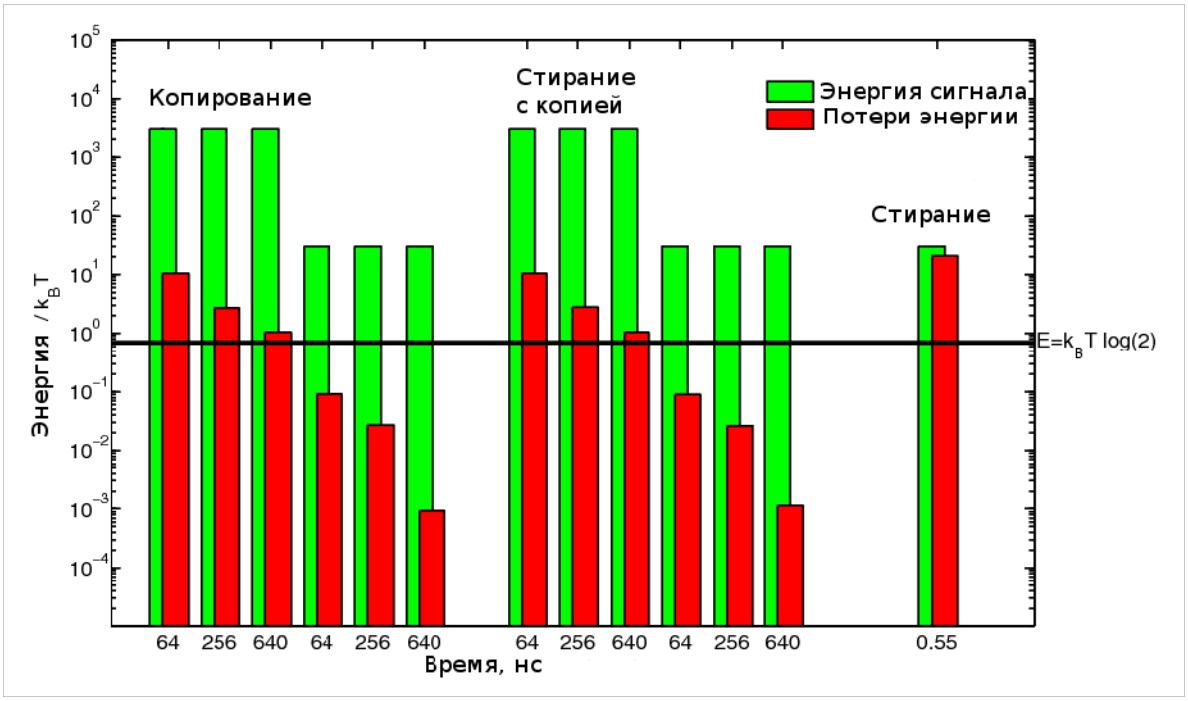

Рис. 1. Результаты эксперимента

Беннет в работе [6] отметил, что предел Ландауэра тепловыделения может быть преодолен лишь если не только вычисления. но и управление также обратимо, и назвал вычисления, которые, будучи однажды запущенными, мчатся сами, баллистическими. Но было в дальнейших «прикладных» исследованиях забыто даже само исходное понятие реверсивности: его понимали как возможность взять назад уже выполненное действие, как инъекцию, а не как биекцию. Тем более игнорировалось замечание Беннета насчет баллистических вычислений, в частности, гейты Тоффоли и Фейнмана были попыткой реверсивно в смысле взаимнооднозначных двоичных преобразований реализовать условные операторы.

Далее, автоматически предполагалось даже теми, кто не забыл понятие взаимной однозначности, что обратимые действия образуют группу, поскольку действительно биекции корректно представляются групповыми структурами [13]. Но есть одна обратимая операция и обратимый элемент вычислений, которые разрушают ассоциативность и, соответственно, не являющийся функцией.

Для обозначения применения действия к аргументу пользуемся обратной польской записью, как принятор в общей алгебре: $(a\ F)$, где F — действие, применяемое к a.

$\stackrel{def}{=}$ **Определение 0.1.**

Абсолютное зеркало — действие, обращающее предыдущее действие: $(f\ M) = f^{-1}$. *UNDO аннулирует предыдущее действие:* $\{a_1;\ \ldots\ \ a_n\}$; *UNDO* = *EMPTY*.

◇ **Пример 0.1.**

M и UNDO неассоциативные сущности и почти никогда не могут быть вложены в структуру полугрупп (не говоря уже о группах):

$$((a \circ b) \circ M) = (b^{-1} \circ a^{-1}) \qquad (a \circ (b \circ M)) = (a \circ b^{-1})$$
$$\{\mathsf{a};\ \mathsf{b}\};\ \mathsf{UNDO} \neq \mathsf{a};\ \mathsf{b};\ \mathsf{UNDO}.$$

Тем самым и двоичные данные не подходят, и алгебраические структуры, и сама концепция вычислений другие. Рассмотрим некоторые возникшие проблемы и найденные решения поподробнее.

1. Предпосылки и прошлые исследования

На самом деле алгоритмические и программные алгебры — классическая ветвь теоретической информатики. Еще в 60-е годы Глушков [7] определил dалгоритмические алгебры с использованием операций, соответствующих базовым конструкциями структурного программирования, а Маурер [8] ввел и исследовал абстрактные алгебры вычислений, отражающие структуру команд компьютера (program generic algebras, PGA). Оба подхода базировались на представлении операций как полугрупп функций. В дальнейшем развитие шло по этим двум направлениям. Основные результаты подхода Глушкова суммированы в [9, 10], а современные достижения, в частности, в [11, 12]. В книге [18] перечислено более сотни работ по программным алгебрам. Имеются две священных коровы в этих работах. Во-первых, почти все они посвящены языкам. полным по Тьюрингу. Но реверсивные или полуреверсивные системы не могут быть полны [20, 21]. Во-вторых, заранее фиксируется минимальный набор команд представляемых алгоритмов, и тем самым формализуется область традиционных вычислений.

В частности, факториал, умножение и деление на традиционном конечном представлении чисел полностью необратимы. Необратимы и арифметические операции при представлении чисел с плавающей точкой. Но даже полной обратимости операций недостаточно для обратимости программы (замечание, впервые явно высказанное Беннетом).

◇ **Пример 1.1.**

Программа сортировки необратима, поскольку забывается начальное состояние массива. Аналогично, хотя каждое вращение кубика Рубика обратимо и они составляют группу, программа сборки кубика существенно необратима.

Такой основательный выход за пределы традиционной парадигмы вычислений заставляет обобщить понятия. А корректное обобщение часто приводит к неожиданным полезным результатам. Ползучее развитие всегда ограничено, нужно подняться наверх и посмотреть на всю картину. даже если сначала трудно дышать и

холодно, и практической пользы вроде бы нет никакой.

Конкретные функции и действия часто будут записываться с помощью λ-обозначений. Но это не означает. что наши конструкцуии базирутся на λ-исчислении. Наоборот, оно оказывается частным случаем наших конструкций.

2. Основные понятия алгебраического подхода к вычислениям

Теперь опишем абстрактную систему алгебр программы и действий и покажем, как от систем переходят к алгебрам, от алгебр — к декомпозициям, от декомпозиций алгебр — к алгебраическим вычислениям. Сначала опишем систему базовых понятий, в основном следуя [17].

$\stackrel{def}{=}$ **Определение 2.1.**

Сигнатура σ — список символов: функциональных (двумкестные функции часто записываются в инфиксной нотации), констант и предикатов. Есть метаоперация $arity(s)$, дающая по каждой функции и предикату количество мест. Если есть предикат $=$, он интерпретируется как равенство. $s \in \sigma$ означает, что символ s присутствует в сигнатуре σ.

Алгебраическая система $\mathbf{S}$ *сигнатуры σ — ее модель в классической логике. Задается тип данных S (носитель) и функция второго порядка ζ, такая. что $\zeta(f) \in S^{\mathrm{arity}(f)} \to S$ для функциональных символов, $\zeta(f) \in S^{\mathrm{arity}(f)} \to \{\mathrm{false}, \mathrm{true}\}$ для предикатов, $\zeta(c) \in S$ для констант.*

$\stackrel{def}{=}$ **Определение 2.2.**

*Группоид — алгебраическая система с одной двухместной функцией и предикатом равенства $\langle \circledast, = \rangle$. Его носитель часто обозначается G, а операция обычно записывается как двухместный инфиксный оператор, символ для которого будем варьировать. Элемент e единичен или является нейтральным, если $\forall x\, x \circledast e = e \circledast x = x$. Элемент 0 является нолем, если $\forall x\, x \circledast 0 = 0 \circledast x = 0$. **Левая единица** такое a, что $\forall x\, a \circledast x = x$. **Левый ноль** такое a, что $\forall x\, x \circledast a = a$. Аналогично для правых единиц и нолей. Идемпотент — такое a, что $a \circledast a = a$. Группоид является полугруппой, если операция ассоциативна. Он коммутативен, если операция коммутативна. Он инъективен, если*

$$\forall x, y, f\, x \circledast f = y \circledast f \Rightarrow x = y.$$

Он моноид, если полугруппа и есть нейтральный элемент. Моноид группа, если для каждого x существует y, такое, что $f \circledast g = g \circledast f = e$. Бигруппоид — алгебраическая система с двумя бирнарными операциями.

Морфизм алгебраических структур — отображение их носителей, сохраняющее все функции и константы и истинность предикатов (но не боязательно их ложность!) Изоморфизм — биективный морфизм, сохраняющий ложность.

Перечисленные выше понятия имеют при алгебраическом подходе прикладной смысл в информатике. Изоморфизмы дают другие представления данных (например, группа двоичных матриц порядка 2 по умножению как группа положений куба или группа преобразований света парателлуритом TeO_2)

Группоид задает списки списков длины 2 в смысле базовго ЛИСПа или двоичные упорядоченные деревья. Коммутативный группоид — неупорядоченные двоичные деревья. Полугруппа — линейные списки или функции с операцией композиции. Другая интерпретация коммутативного группоида иллюстрируется примером известной игры.

◇ **Пример 2.1.**

Носитель состоит из трех элементов: колодец (w), ножницы (s), бумага (p) {w,s,p}. Операция для каждой пары элементов дает победителя.

$$w \circledast s = w; \quad w \circledast p = p; \quad s \circledast p = s; \quad x \circledast x = x.$$

Тогда $(w \circledast s) \circledast p = w \circledast p = p$, но $w \circledast (s \circledast p) = w \circledast s = w$. Так что одновременные либо независимые действия делают операцию коммутативной и наоборот: если операция коммутативна, аргументы можно вычислять в любом порядке. Поэтому коммутативность и ассоциативность операции требуется системами распределенных вычислений, в частности, MAP-REDUCE.

А с логической точки зрения коммутативная полугруппа может рассматриваться как работа над единым хранилищем ресурса, похожего по природе на деньги (линейная логика Жирара [16]). Имеет значение лишь общий объем оставшегося ресурса. Операции расходуют его независимо.

ℂℏ. **Теорема 2.1.**

Каждая полугруппа изоморфна полугруппе функций с композицией в качестве операции. [17]

Таким образом, пространство действий может рассматриваться как пространство функций, лишь если есть ассоциативность. Поэтому операцию в полугруппе будем обозначать $\circ$. И поэтому же полугруппы исключительно важны для алгебраических вычислений.

◇ **Пример 2.2.**

Совокупность функций или отношений с операцией композиции — полугруппа. Это проходит даже для языков с типами данных. Композицией не подходящих по типу операций будет 0, который интерпретируется как фатальная ошибка. Конечный автомат может рассматриваться как конечная полугруппа и наоборот. Действия в каждом языке программирования с операцией последовательного выполнения ;образуют полугруппу, даже если сами функциями не являются (синтаксическая полугруппа).

3. Группоид как вычислительная структура

Группоид может рассматриваться как совокупность «функциональных программ» даже если действия не являются функциями! и как вычислительная структура в потенциале более высокого типа, чем фон Неймановская. Каждый элемент группоида может рассматриваться как действие $\lambda x.\,(x \circledast a)$, как действие над действиями и так далее. Данные и команды совпадают.

С этой точки зрения можно теперь посмотреть на элементы и их свойства в «алгебраическом компьютере».

1) Ассоциативность означает отсутствие побочных эффектов либо их ограничение растратой некоего общего ресурса.
2) Правая единица e — команда «ничего не делай».
3) 0 — критическая неустранимая ошибка. В частности, в математике 0 для полугруппы отношений — пустое отношение, нигде не определенная функция.

   А вот «половинчатые» ноли не таковы.
4) Левый ноль $(z \circledast x) = z$ — конечный результат вычислений. Если есть настоящий ноль, то описание ослабляется до $\forall x\,(x \neq 0 \supset (z \circledast x) = z)$ (**левый нулек**).
5) Правый ноль $(x \circledast z) = z$ — 'quine' (программа, вычисляющая сама себя) либо вход, начальное значение, забивающее любого предшественника. В случае настоящего ноля опять уточнение $\forall x\,(x \neq 0 \supset (x \circledast z) = z)$.
6) Идемпотент $(z \circledast z) = z$ — приостановка.
7) Правое сокращение $(a \circledast f) = (a \circledast g) \supset f = g$ сейчас кажется почти бесполезным практически свойством: любая программа работает на каждом элементе по-другому. Но есть важное исключение. В группах такое свойство выполнено
8) Если $(x \circledast a) \circledast \tilde{a} = x$, тогда $\tilde{a}$ называется слабым правым обратным для a. Оно позволяет отмену действия a.
9) Труба p такой элемет, что

$$(x \circledast p) = y \supset (y \circledast m) = 0.$$

   Она пропускает данне лишь в одну сторону.
10) Подгруппоид может рассматриваться и как подпрограмма, и как блок конструкции, и как тип данных.
11) Прямое произведение группоидов означает возможность разбить вычисление на независимые ветви, сооответствующие сомножителям.

Так что можно объявить:

Алгебраическое программирование является функциональным и одновременно супер-фон-Неймановским по природе.

$\stackrel{def}{=}$ **Определение 3.1.**
Действие последовательности элементов группоида $f_1, \ldots, f_n$ *— функция*

$$\varphi_{f_1;\ldots;f_n} = \lambda x.\,(\ldots((x \circledast f_1) \circledast f_2)\ldots) \circledast f_n).$$

➳ **Лемма 3.1.**
Действия группоида образуют полугруппу.
Доказательство.
Композицией $\varphi_{f_1;\ldots;f_n}$ и $\varphi_{g_1;\ldots;g_k}$ является действие, сопоставленное

$$\varphi_{f_1;\ldots;f_n;g_1;\ldots;g_k}.$$

■

Чтобы структурировать алгебраические вычисления, нужно уметь делить системы на подсистемы и иметь набор средств сопряжения подсистем. Основные подсистемы и сопряжения следующие.

1) блок;
2) связь;
3) командник.

Блок является подгруппоидом. Все действия внутри блока не выводят из него. Желательно, чтобы входные и выходные значения также были внутри блока (как его правые и левые ноли). Для дополнительного структурирования вычислений целесообразно потребовать

Если x и y из разных блоков, то $(x \circledast y) = 0$.

Важнейшим преобразованием блока является дуал. Дуалом $\mathbf{G}$ является группоид $\mathbf{G}'$ с тем же носителем и операцией в обратном порядке $(x * y) = (y \circledast x)$. Во многих случаях блок и дуал реализуются одной и той же подпрограммой или элементами. Дуал ассоциативной системы ассоциативен.

$$(a * (b * c)) = (c \circ b) \circ a; \quad ((a * b) * c) = c \circ (b \circ a).$$

И тем не менее в математической и программной моделях блок и дуал необходимо строго различать. Поэтому элемент дуала, соответствующий a, обозначается a'. $'$ не является внутренней операцией системы. И тем не менее в конечной реализации a и a' обычном будут одним и тем же значением, элементом или сигналом, но помещенным в разные контексты.

Связи передают информацию и (или) управление между блоками. Они находятся вне связанных ими блоков. Поэтому на них накладывается требование: если c связь и $(x \circledast c) = y$, то $(x \circledast y) = 0$. Два важнейших вида коннекторов: зеркала и трубы (определенные ранее).

Зеркало m такой элемент, что

$$(x \circledast m) = y \equiv (\tilde{y}' \circledast m) = \tilde{x}'.$$

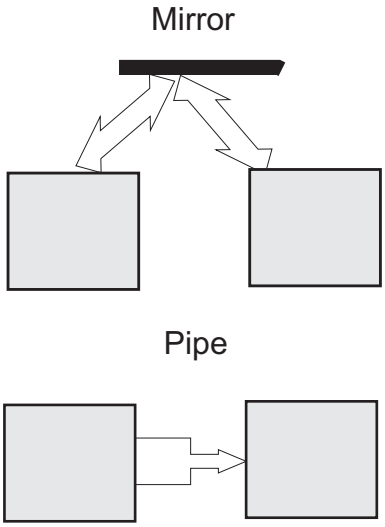

Рис. 2. Связи

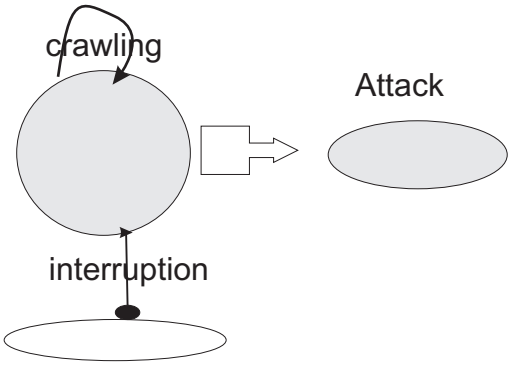

Рис. 3. Чёрная вежливая кошка

Сейчас создалось впечатление, что зеркала и трубы достаточны для структурного конструирования алгебраических вычислений.

Командник — группоид, декомпозированный в прямое произведение $G1 \times G2$ данных и управления (которое часто называется сенсорами или прерывателями), где у данных $G1$ нет ни входов, ни выходов. При его реализации необходимо иметь механизм задания изнутри начального данного и начального управления и обеспечить насильственное прерывание снаружи процесса $G1$, когда $G2$ породил выход.

◇ **Пример 3.1.**

Пусть умная, смелая и вежливая кошка выслеживает мышь в темной комнате. Здесь $G1$ — процесс выслеживания, $G2$ — сенсор, прерывающий выслеживание и начинающий атаку, когда мышь готова заметить кошку. Тогда можно описать действия следующей диаграммой Подкрадывание и атака легко описываются полугруппами (см. [25]). Прерывание и труба неассоциативные и необратимые дей-

ствия. Сенсоры подкрадывания почти обратимы: их можно описать группой, в которой один элемент заменён на выход.

4. Алгебры вычислений (General algebraic program structure, GAPS)

Рассмотрения выше приводят к формальному определению. Обозначим операцию применения f к x через $x \star f$, операцию композиции двух элементов в один блок $a \circ b$.

$\stackrel{def}{=}$ **Определение 4.1.**

Алгебра вычислений (GAPS) — бигруппоид со следующими свойствами:

$$((x \circ y) \circ z) = (x \circ (y \circ z)) \tag{1}$$
$$((x \star f) \star g) = (x \star (f \circ g)) \tag{2}$$

Если в ней есть единица и (или) ноль, они должны удовлетворять уравнениям

$$(0 \star x) = (x \star 0) = 0 \tag{3}$$
$$(x \star e) = x \tag{4}$$
$$x \circ 0 = 0 \circ x = 0 \tag{5}$$
$$e \circ x = x \circ e = x. \tag{6}$$

➳ **Лемма 4.1.**

Любой группоид $\mathbf{G}$ *можно расширить до GAPS.*

Доказательство.

Рассмотрим следующую хорнову теорию. Константами являются все элементы $\mathbf{G}$. Она включает диаграмму $\mathbf{G}$ (множество истинных замкнутых эоементарных формул), новую константу $\mathbf{B}$ и аксиомы для неё:

$$((x \star f) \star g) = (x \star (f \star (g \star \mathbf{B}))) \tag{7}$$
$$(f \star ((g \star (h \star \mathbf{B})) \star \mathbf{B})) = ((f \star (g \star \mathbf{B})) \star (h \star \mathbf{B})) \tag{8}$$

Инициальная модель этой теории и есть искомая GAPS. ■

Можно утверждать, что GAPS минимальное из известных алгебраических описаний свойств совокупностей эффективных вычислений.

Вторая операция добавляет выразительности.

$\stackrel{def}{=}$ **Определение 4.2.**

Элемент $x \neq 0$ — делитель ноля, если есть такое $y \neq 0$, что $x \circ y = 0$. Элемент y — правый обратный для x, если $x \circ y = e$. Элемент x^{-1} обратный к x, если $x \circ x^{-1} = x^{-1} \circ x = e$. Два элемента взаимно обратны, если $a \circ b \circ a = a$, $b \circ a \circ b = b$[1].

[1]Взаимно обратными могут быть даже необратимые элементы; но если есть настоящий обратный, то других взаимно обратных нет

Каждый из взаимно обратных элементов аннулирует действие другого, но лишь на области его значений, а в других случаях может быть не определен (давать 0).

Следующая теорема показывает, что каждая система функций (полугруппа) может быть обогащена системой глобальных действий над ними, поэтому ее естественно назвать **теоремой об абстрактных метавычислениях**. Она нуждается в предварительных обсуждениях.

Напомним понятие обогащения алгебраической системы (его прямой аналог для классов в языках программирования называется специализацией). Пусть сигнатура σ_1 расширяет сигнатуру σ. Алгебра $\mathbb{A}_1$ in σ_1 есть обогащение $\mathbb{A}$ in σ если все носители, функции, константы и предикаты σ совпадают.

Пусть дан морфизм $\alpha : \mathbb{G} \to (G \to G)$ полугруппы $\mathbb{G}$ в полугруппу преобразований её носителя, такой, что $(e\ \alpha) = \lambda x.\, x$, $(0\ \alpha) = \lambda x.\, 0$. С точки зрения программирования, это интерпретатор, транслятор, компилятор или суперкомпилятор, преобразующий программу высокого уровня в исполняемый модуль.

Пример такого морфизма. $(x\ \alpha) = \lambda x.\, 0$ для всех делителей ноля. $(x\ \alpha) = \lambda x.\, x$ для остальных.

𝕋. **Теорема 4.1.**

Каждая полугруппа $\mathbb{G}$ может быть обогащена до такой GAPS, что $(x \star f) = (x\ (f\ \alpha))$.

Доказательство.

Нетривиален лишь закон (8).

$$((x \star f) \star g) = ((x\ (f\ \alpha))\ (g\ \alpha)) = (x\ ((f\ \alpha) \circ (g\ \alpha))) =$$
$$(x\ (f \circ g\ \alpha)) = (x \star (f \circ g))$$

□ ■

➳ **Лемма 4.2.**

Для каждого действия группоида φ есть элемент GAPS α, такой, что

$$(x \star \alpha) = (x\ \varphi).$$

Доказательство.

Пусть $f_1, \ldots, f_n$ — действие. Применяя (2) и ассоциативность $\circ$, получаем

$$(\ldots((x \circledast f_1) \circledast f_2) \ldots) \circledast f_n) = (x \circledast f_1 \circ f_2 \cdots \circ f_n).$$

□ ■

◇ **Пример 4.1.**

Некоторые ‘естественные’ предположения разрушают структуру GAPS. Например, если $\forall f\ (e \star f) = f$, то две операции совпадают.

$$(x \star y) = ((e \star x) \star y) = (e \star (x \circ y)) = x \circ y.$$

Рассмотрим этот казус (принципиальное различие ноля и единицы) подробнее. 0 является фатальной ошибкой, с которой ничего не сделаешь внутри системы. e можно понимать как пустую программу, но преобразователь программ может сделать из неё всё, что угодно. Смотри пример 6.

◇ **Пример 4.2.**

Прямое применение теоремы о пополнении группоида до GAPS почти всегда приводит к бесконечным структурам. Покажем. как можно получать конечные GAPS.

Рассмотрим для примера группоид из примера 1. Прежде всего, выпишем все различные действия группоида в форме `последовательность ходов: ее воздействие на все элементы`.

w: wwp, s: wss, p: psp, ws: wws, sw: www, ps: sss,
sp: pss, wp: ppp, pw: pwp, wsp: pps, spw: pww, pws: sws.

Полугруппа действий, следовательно, состоит из 12 элементов и некоммутативна, хотя исходный группоид был коммутативен. Теперь осталось распространить на неё операцию группоида $*$. Часто GAPS может быть построена через одну и ту же полугруппу разными способами. В данном примере по меньшей мере два варианта:

(ws * sw) = wssw = wsw = ws;
(ws * sw) = (ws)*(s*w) = (ws)w = (w*w)(s*w) = ww = w.

Установим, когда можно соединить систему преобразований программ с данной программной системой (= полугруппой программ), получив единый язык метапрограммирования. Есть три естественные подзадачи.

1) Добавить систему преобразований, не изменяя самого языка.
2) Как сохранить подъязык (подполугруппу), возможно, преобразуя другие конструкции в метавычисления?
3) Расширить язык до метаязыка не изменяя понятия внутри самого языка.

$\stackrel{def}{=}$ **Определение 4.3.**

Пусть система преобразований $\mathbb{A}$ *задана как система функций на носителе полугруппы* $\mathbb{G}$, *свойства которой описаны теорией* Th, *а желаемые свойства преобразований описаны теорией* Th_1. *Система* консервативна, *если имеется обогащение* $\mathbb{G}$ *до GAPS* $\mathbb{AG}$, *такое, что* $\varphi \in \mathbb{A}$ *представимо действием группоида* $(x \star \alpha) = (x\ \varphi)$ *для некоторого* α. *Она* консервативна над подполугруппой $\mathbb{G}_0$ *если консервативна и* $a \star b = a \circ b$ *для всех элементов* $\mathbb{G}_0$. *Она* допустима, *если есть такая полугруппа* $\mathbb{G}_1$, *что* $\mathbb{G} \subseteq \mathbb{G}_1$ *и* $\mathbb{A}$ *консервативно над* $\mathbb{G}_1$.

Пусть Th_P — *теория* Th, *в которой все кванторы ограничены одноместным*

предикатом P: $\forall x\, A(x)$ заменяется на $\forall x\, (P(x) \supset A(x))$; $\exists x\, A(x)$ заменяется на $\exists x\, (P(x)\&A(x))$.

$\mathbb{A}$ ***сильно допустимы*** *если они допустимы и в результирующей алгебре выполнены* Th_1 *и* Th.

➳ **Лемма 4.3.**

Совокупность действий $\mathbb{A}$ консервативна тогда и только тогда, когда ее замыкание изоморфно подполугруппе $\mathbb{G}$.

Доказательство.

По предложению 4.2, если обогащение успешно, то каждый элемент полугруппы, порождаемой $\mathbb{A}$, представляет собой действие некоторого элемента $\mathbb{G}$.

Обратно, если замыкание $\mathbb{A}$ вложимо в $\mathbb{G}$, то каждое действие $\mathbb{A}$ представляется образом в этом вложении.

□ ■

◇ **Пример 4.3.**

Несмотря на простоту, предыдущее предложение полезно. Рассмотрим единственное действие: обращение программ $\mathbf{M}$, и единственное его свойство $((a\ \mathbf{M})\ \mathbf{M}) = a$. Обогатить полугруппу программ этим действием возможно тогда и только тогда, когда в полугруппе имеется элемент порядка 2: $f \neq e \& f \circ f = e$. При этом безразлично, как это f ведёт себя внутри исходного языка.

➳ **Лемма 4.4.**

Совокупность действий $\mathbb{A}$ консервативна над $\mathbb{G}_0$ тогда и только тогда. когда имеется мономорфизм ψ ее замыкания в $\mathbb{G}$, и для каждого f, такого, что $(f\ \psi) \in \mathbb{G}_0$, выполнено $(a\ f) = (a \circ (f\ \psi))$.

Следующая теорема опубликована в более слабом варианте в [25]. Ее доказательство требует серьёзной теоретико-модельной техники, и, конечно же, не всегда даёт алгоритмическое построение.

𝕋h. **Теорема 4.2.**

(2012–2014) Пусть система действий $\mathbb{A}$ описана теорией Th_1*, и* Th *— теория, описывающая полугруппу $\mathbb{G}$, P — новый унарный предикат.*

$\mathbb{A}$ строго допустима над $\mathbb{G}$ т. и т. т., когда есть частичная сюръекция $\psi : \mathbb{G} \to \mathbb{A}$, $(g_1 \circ g_2\ \psi) = (g_1\ \psi) \circ (g_2\ \psi)$, если все результаты определены, и теория $\mathrm{Th}_1 \cup \mathrm{Th}_P$ *непротиворечива.*

Из теоремы следует

➳ **Предложение 4.1.**

Каждая совокупность действий $\mathbb{A}$ допустима над $\mathbb{G}$, если обе теории содержат лишь факты (истинные формулы вида $[\neg](a\{\circ, *\}b) = c$).

◇ **Пример 4.4.**

Рассмотрим аддитивную группу $\mathbb{Z}_3$ и добавим действия ее объектов $\{0, 1, 2\}$ так, что они станут колодцем, ножницами и бумагой из примера 1. Их действия можно описать как функции над $\mathbb{Z}_3$ со значениями 002, 011, 212. А теперь расширим группоид игры wsp до двенадцатиэлементной полугруппы, как в примере

2. Необходимо добавить ещё единицу и ообзначим полученный моноид $\mathbb{C}$. Определим действия на прямом произведении $\mathbb{C} \times \mathbb{Z}_3$ следующим образом:

$$(\langle c, x\rangle \star \langle d, y\rangle) = \langle c \circ d, (x + y\ d)\rangle\,.$$

Эта GAPS содержит $\mathbb{Z}_3$. Элементы $\mathbb{C}$ могут рассматриваться как команды, а элементы $\mathbb{Z}_3$ как данные. $(x\ d)$ — применение последовательности действий к элементу, закодированному x, и кодирование результата. Команды преобразуются как полугурппа. но их действия образуют группоид.

Есть и другой способ определить GAPS на том же носителе:

$$(\langle c, x\rangle \star \langle d, y\rangle) = \langle (c \star d), (x + y\ (c \star d))\rangle\,.$$

Наша теорема существования чистая. она даёт построение алгебры, но не гарантирует её конечности или алгоримичности. Но из неё следует важный негативный принцип.

Практическое следствие. *Если система преобразований разрушает свойства программ или соливает разные программы, она некорректна.*

Есть частный случай, когда построенная алгебра перечислима и операции алгоритмичны.

$\stackrel{def}{=}$ **Определение 4.4.**

Квазитождество — формула вида

$$\forall x_1, \ldots, x_k\,(Q_1 \& \cdots \& Q_n \supset P),$$

где x_i — все встречающиеся в ней переменные, а все Q_i, P предикаты.

Если теория состоит из квазитождеств, то GAPS может быть построена как факторизация свободной GAPS по доказуемому равенству термов (инициальная модель).

◇ **Пример 4.5.**

Поскольку λ-исчисление лежит в основаниях и является основным инструментом в современной математической теории языков. полных по Тьюрингу (см. [18]), достаточно построить модель λ-исчисления как GAPS. Перейдём к эквивалентному представлению λ-исчисления как комбинаторной логики в базисе $\{\mathbf{I}, \mathbf{B}, \mathbf{C}, \mathbf{S}\}$ [19] и опишем её с помощью тождеств

$$(x \star \mathbf{I}) = x$$
$$(x \star (f \star (g \star \mathbf{B}))) = ((x \star f) \star g)$$
$$(x \star (f \star (g \star \mathbf{C}))) = ((x \star g) \star f)$$
$$(x \star (y \star (z \star \mathbf{S})) = ((x \star y) \star (x \star z)).$$

Добавляя аксиому ассоциативности композиции $\mathbf{B}$ (8) и определение

$$f \circ g = (f \star (g \star \mathbf{B}))$$

получаем модель λ-исчисления и, как следствие, модели всех стандартных и почти всех нестандартных языков программирования.

Чтобы получить модель типизированного λ-исчисления, достаточно добавить ноль 0 и переопределить $\star$ как ноль, когда типы не согласуются. Чтобы получить единицу, достаточно добавить аксиомы:

$$(\mathbf{I} \star (f \star \mathbf{B})) = f \qquad (f \star (\mathbf{I} \star \mathbf{B})) = f.$$

◇ **Пример 4.6.**

Рассмотрим ещё один пример превращения программы в GAPS. Пусть есть алгоритмический язык, все символы которого являются командами а соединение строк — композицией программ (например, Brainfuck [22]). Тогда пустая строка служит программой, не делающей ничего. Она может быть естественно представлена как GAPS. $a \circ f$ — соединение строк. $a \star f$ определяется следующим образом. Если f синтаксически правильная программа, то ее значение на a есть $a \star f$. Если f приводит к ошибке или синтаксически некорректна, то значение 0.

Видно, что результат действия, применённого к пустой программе, может оказаться любым.

◇ **Пример 4.7.**

В 1972 одно из исследований остановилось в шаге перед GAPS [23].

Рассмотрим алфавит $\{K, S, (,)\}$. Его символы переведём в комбинаторы следующим образом

$$(K\ \varphi) = \mathbf{K} \quad (S\ \varphi) = \mathbf{S} \quad (`(`\ \varphi) = \mathbf{B} \quad (`)`\ \varphi) = \mathbf{I}$$

Результат перевода строки определяется рекурсивно (a символ, σ строка):

$$(a\sigma\ \varphi) = ((\sigma\ \varphi) \star (a\ \varphi)).$$

Эта интерпретация превращает комбинаторную логику в язык типа Brainfuck. Но результирующая GAPS не подходит для комбинаторной логики. Она включает и все синтаксические некорректные конструкции, подобные))))**SK**((.

◇ **Пример 4.8.**

Если наша полугруппа моноид, то универсальная функция в GAPS тривиальна $\mathbf{U} = e$:

$$(x \star (f \star \mathbf{U})) = (x \star f),$$

Частичное вычисление зато нетривиально:

$$(f \star (x \star \mathbf{PE})) = (x \star f),$$

Оператор фиксированной точки

$$((f \star \mathbf{Y}) \star f) = (f \star \mathbf{Y}),$$

тривиален, если есть 0: $\mathbf{Y} = 0$.

В GAPS легко выражаются функциональные ограничения на программы. В статьях [24, 25] описаны и исследованы классы GAPS для обратимых, полуобратимых и полностью необратимых программ.

5. О некоторых инструментах композиции алгебраических структур

Алгебраическое программирование сводится к системе инструментов композиции и декомпозиции алгебраических подструктур GAPS. Некоторые из инструментов были показаны в главе о группоидах 3.

В примере4 использован важный инструмент (де)композиции структур. Элементы $\mathbb{Z}_3$ могут рассматриваться как данные, а элементы полугруппы как команды. Сформулируем общий случай такой конструкции.

$\stackrel{def}{=}$ **Определение 5.1.**

Полупрямое произведение GAPS. Пусть даны две GAPS: GAPS команд C и GAPS данных D. Пусть $\varphi : \mathbf{C} \to \mathrm{Hom}(\mathbf{D}, \mathbf{D})$ — морфизм полугруппы команд в полугурупу морфизмов полугруппы даннх. Тогда полупрямое произведение $C \rtimes D$ есть $C \times D$ со следующими операциями:

$$\langle c_1, d_1\rangle \circ \langle c_2, d_2\rangle = \langle c_1 \circ c_2, d_1 \circ (d_2\ (c_2\ \varphi))\rangle$$
$$\langle c_1, d_1\rangle \star \langle c_2, d_2\rangle = \langle c_1 \star c_2, d_1 \star (d_2\ (c_2\ \varphi))\rangle\,.$$

Введём типы данных и их связи, учитывая, что каждое данное заодно является действием и что не должно быть прямых передач данных или управления от одного типа к другому.

$\stackrel{def}{=}$ **Определение 5.2.**

Система типов на GAPS — система под-GAPS T_i, таких, что если $i \neq j$, $a \in T_i$, $b \in T_j$, то $(a \star b) = 0$ и $T_i \cap T_j \supseteq \{0\}$.

Связь (между T_i и T_J) — элемент c, не принадлежащий ни одному из типов и если $(a \star c) = b \neq 0$, то есть $i \neq j$, такие, что $a \in T_i$, $b \in T_j$.

Дуал T_i' типа T_i — GAPS, для которой есть биекция на T_i $x \leftrightarrow x^p rime$, и если $(x \star y) = z$, то $(z' \star y^p rime)e = x'$. Дуал совершенный, если $(x \circ y)' = (y' \circ x'$.

Зеркало — связь m, такая, что если $(a \star m) = b$, $a \in T_i$, $b \in T_j$ то $(b' \star m) = a'$. Зеркало идеальное, если оно связь между T_i и его совершенным дуалом T_i'.

Обратимый тип такая под-GAPS R, для которой существует идеальное зеркало M и для каждых $x, y \in R$

$$((x \star y)' \star (y \star M)) = x' \quad ((x' \star (y \star M))' \star y) = x.$$

кристалл такая под-GAPS, что $\star$ задаёт группу.

Труба — связь p, такая, что $(x \star p) = y \neq 0 \supset (y \star p) = 0$.

Результат типа T_i — левый ноль T_i относительно обеих операций.

прерывательПрерыватель тип с носителем $T \times \{1, 2\}$, где T GAPS, в которой нет результатов, кроме, возможно, 0 и операции определены следующим образом:

$$(\langle a, 1\rangle \star \langle b, x\rangle) = \langle (a \star b), x\rangle\,; \tag{9}$$

$$(\langle a, 2\rangle \star \langle b, y\rangle) = \langle a, 2\rangle \ \textit{если}\ b \neq 0 \tag{10}$$

Колесо — прерыватель, базирующийся на группе $\mathbb{Z}_n$.

Управление по прерыванию — система типов T_i, прерыватель S и труба p между T_i и $\{1, 2\}$, такая, что $(x \star p) = 1$ если x не результат и $(x \star p) = 2$, если x результат.

Управление по прерыванию может быть представлено как полупрямое произведение, при таком описании трубы отсутствуют, зато число элементов растёт мультипликативно.

В алгебраическом программировании традиционные типы данных работают плохо.

◇ **Пример 5.1.**

При умножении матриц представление их как массивов действительных чисел оказывается худшим.

В частности, матрицы 2*2 можно расслоить по «двоичным разрядам» и умножение невырожденных матриц (иначе необратимо) представляет из себя группу положений куба, которая, в свою очередь, реализуется преобразованиями света в кристалле парателлурита TeO_2.

6. Базовые понятия для чисел

В связи с последним замечанием, рассмотрим представления для чисел, позволяющие обойти эффект стены памяти.

В соответствии с общей идеологией: не привязываться к конкретным частным представлениям алгоритмов — будем доказывать отрицательные результаты о вычислимости на базе лишь принципа конечной информации, сформулированного в [1]:

Принцип конечной информации

> *Пусть* Φ *— эффективное преобразование объекта* α *в объект* β*. Тогда всякая конечная информация, которую можно вычислить по* β*, может быть вычислена, с использованием* Φ *и конечной информации об* α*.*

Методом доказательства неразрешимости задачи о преобразованиях функций в абстрактной теории вычислимости чаще всего служит метод провокации, также сформулированный в абстрактной форме в [1].

Метод провокации

Пусть дана проблема построения по каждому α, имеющему тип T1, *для которого выполнено свойство $A(\alpha)$, объекта β типа* T2, *удовлетворяющего свойству $B(\alpha, \beta)$. На конструктивном языке она в общем случае записывается формулой*[2]

$$\forall \alpha : \mathrm{T1}\,(A(\alpha) \Rightarrow \exists \beta : \mathrm{T2}\, B(\alpha, \beta)). \tag{11}$$

или же, в частном случае, когда построенный объект заведомо конечен,

$$\forall \alpha : \mathrm{T1}\,(A(\alpha) \Rightarrow B_1(\alpha) \vee \cdots \vee B_n(\alpha)). \tag{12}$$

Пусть функционал Ψ претендует на то, что он строит по каждому объекту α, удовлетворяющему свойству $A(\alpha)$, объект, удовлетворяющий свойству $B(\alpha, (\Phi\ \alpha))$ либо на то, что Ψ указывает верный член дизъюнкции. Чтобы установить, что данная проблема эффективно неразрешима, достаточно дать метод построения последовательности приближений $(\Psi\ n\ f)$, который работает следующим образом. Он вырабатывает вспомогательную последовательность приближений $(n\ f_0)$, не зависящую от Φ, подает её на вход Ψ и в момент, когда функционал Ψ выдаст некоторое приближение к решению, обладающее заранее заданной заведомо достижимой характеристикой (например, некоторой точностью либо ответом на вопрос в конкретной точке), далее выдает в зависимости от этого ответа одну из конечного числа последовательностей приближений $(n\ f_i)$, показывающую ошибочность выданного ответа (например, если выдано число с некоторой точностью, дальнейшие наши приближения находятся вне интервала, в котором может лежать данное число).

Таким образом, структуру провоцирующего функционала можно описать следующей (квази)программной конструкцией.

Квазипрограммное описание метода провокации, данное ниже, показывает, что программа анализа и выдачи контрпримера ни в коем случае не оказывается *внутри* опровергаемой программы. Она ее вызывает как модуль.

Чтобы применять принцип конечной информации к преобразованиям действительных чисел, необходимо осознать, что мы считаем конечной информацией о действительно числе. Как самое общее из конструктивных определений действительного числа (согласно анализу В. А. Успенского [2]) рассмотрим следующее.

$\stackrel{def}{=}$ **Определение 6.1. (Действительное число).**

[2]Именно в общем случае! Если формулы A и B не являются чисто дескриптивными, то ситуация может осложниться. Более того, ниже разобранный случай с $\vee$ относится именно к данной группе исключений.

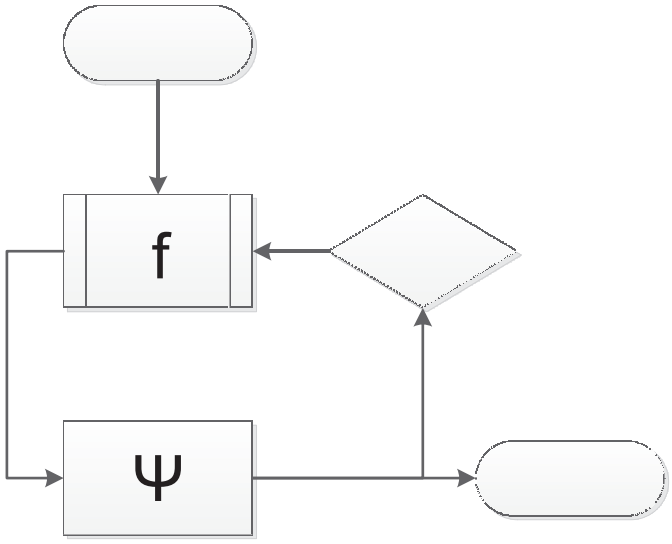

Рис. 4.

Общее представление действительного числа x — эффективная функция, сопоставляющая каждому положительному рациональному числу ε рациональный сегмент $[a_\varepsilon, b_\varepsilon]$ длины не больше ε, такая, что все вырабатываемые ею сегменты попарно пересекаются.

Представления называются эквивалентными, *если любые два сегмента из них пересекаются.*

Традиционный следующий шаг: действительное число — класс эквивалентности представлений — в конструктивизме некорректен.

Соответственно, конечной информацией о действительном числе является любой из порождаемых его представлением сегментов.

Определим (для положительных чисел, чтобы избежать лишней несущественной техники) понятие позиционной системы и представления в ней числа.

$\stackrel{def}{=}$ **Определение 6.2. (Позиционная система).**

Позиционная система — бесконечная в обе стороны последовательность натуральных чисел, такая, что $n_i \geqslant 2$ для всех i. i-тая цифра — натуральное число от 0 до $n_i - 1$. Представлением в позиционной системе будем называть пару из количества знаков целой части числа m и последовательности, выдающей для каждого знака целой и дробной части цифру. Отрезок $[k_i, l_i)$, определяемый очередной цифрой числа, задается индуктивно. Цифру $m+1$ по умолчанию можно считать 0 и взять за базис рекурсии.

$$[k_{m+1}, l_{m+1}] = \left[0, \prod_{i=0}^{m} n_i\right).$$

$$[k_i, l_i) = \left[\frac{l_{i+1} - k_{i+1}}{n_i} \cdot c_i, \frac{l_{i+1} - k_{i+1}}{n_i} \cdot (c_i + 1)\right)$$

$\frac{l_{i+1}-k_{i+1}}{n_i}$ *будем называть длиной разряда. Отрезок, определяемый цифрой позиционного представления некоторого числа, будем называть естественным для данного представления.*

Конечной информацией о числе является начальный отрезок этой последовательности. Наши рассмотрения работают, в частности, для таких систем, как гиперэкспоненциальная и факториальная. Единственное. что предполагается: каждый раз n_i цифр делят предыдущий отрезок длины a на n_i равных частей длины $a/n_i|$, соответственно.

Th. **Теорема 6.1. (Общность).**

(Первые два пункта Брауэр, 1922 [14] Банах, Мазур, 1938, Успенский, 1956; все утверждения обобщены)

1) *Имеется алгоритм преобразования чисел из любой позиционной системы в общее представление.*
2) *Ни для какой позиционной системы нет алгоритма преобразования числа из общего представления в данную систему.*
3) *Алгоритм преобразования числа из позиционной системы α в систему β имеется в том и лишь в том случае, если любой естественный отрезок системы β представляется как конечное объединение отрезков системы α.*

В оригинальных формулировках рассматривались лишь системы с постоянным основанием.

Причиной неразрешимости является некорректность задач.

Доказательство.

Пункт 1. Покажем неразрешимость задачи перевода чисел из позиционной системы в общее представление. Более того, покажем, что даже целую часть числа не удастся вычислить

Пусть дан некоторый функционал $\Phi : (\mathrm{Real} \to \mathbb{N})$, претендующий на решение этой проблемы. Он получает на вход функцию, перерабатывающую точность в приближение. Будем подавать на его вход в ответ на ε отрезок $[1-\varepsilon/2, 1+\varepsilon/2]$, одновременно запоминая минимум запрошенных ε, пока Φ не выдаст целую часть. Если он выдаст 1, запомним теперь уже навсегда текущий минимум ε_0 и при запросах отрезка меньшей длины будем выдавать $[1-\varepsilon/2, 1-\varepsilon/4]$, если 0, то $[1+\varepsilon/4, 1+\varepsilon/2]$. Тем самым мы обманули данную функцию, спровоцировав ее дать ответ и в дальнейшем изменив последовательность так, чтобы она под выданный вариант не подходила.

Пункт 2. Аналогично, даже целую часть не удается вычислить. Складываем два числа с целой частью 0, подавая у первого на вход максимальные цифры, у второго ноли. Как только будет выдана целая часть числа, в случае 0 изменим последующие цифры первого на 0, а в случае 1 второго на максимальные. Для умножения рассуждения подобны, достаточно взять два одинаковых приближения к 1 и после ответа сделать их разными.

Пункт 3. Представление естественного отрезка второй системы как конеч-

ного объединения отрезков первой позволяет задать для каждого i число знаков системы α, достаточное для вычисления i-того знака в системе β: максимальное число знаков отрезков представления. Если же такого представления для некоторого естественного отрезка β нет, то один из его концов не является концом никакого отрезка α, и в его окрестности применим метод провокации. ■

Таким образом, традиционные системы представления чисел отнюдь не идеальны. Дополнительной «прелестью» их использования является необходимость запоминания переносов, а приведённые выше теоремы показывают, что избавиться от переносов или ограничить их глубину невозможно.

Следующие построения возникли в связи с реальной задачей: обеспечить обработку чисел без использования стены памяти. Один из способов этого известен: аналоговые вычисления. Но они заранее ограничены точностью измерения сигнала (максимум 4–5 десятичными знаками). Здесь предлагается способ обеспечить любую требуемую точность за счёт изменения представления чисел.

Поскольку любой тип вычислений, подобных аналоговым, требует фиксированного диапазона представления чисел, и в неявной форме такие диапазоны протащены в стандартное представление чисел с плавающей запятой (диапазон представления числа определяется его порядком), то в нашем представлении числа заранее ограничены отрезком. Но этот отрезок может быть любым и в ходе вычислений варьироваться (как в представлении с плавающей запятой). В некотором смысле это инверсия задачи интервальной арифметики: в ней думали. как при стандартном представлении чисел получить гарантированные оценки, а здесь сама система интервальных оценок даёт представление чисел.

$\stackrel{def}{=}$ **Определение 6.3. (Брауэрова система).**

Брауэровой системой *представления чисел на отрезке* $[\mathrm{low}, \mathrm{high}]$*называется четвёрка*

$$\langle \mathrm{low}, \mathrm{high},\ \nu = \lambda i.\, \nu_i,\ \varepsilon = \lambda i.\, \varepsilon_i \rangle$$

где

$$\nu : \mathbb{N}^+ \to \mathbb{N}^+, \quad \varepsilon : \mathbb{N}^+ \to \mathbb{Q},$$
$$\forall i \in \mathbb{N}^+\, \nu_i > 1,$$
$$\exists \alpha_1, \alpha_2 \in \mathbb{Q}\, \forall i \in \mathbb{N}^+\, 0 < \alpha_1 < \varepsilon_i < \alpha_2 < 1.$$

Если функции постоянны, то система идентифицируется интервалом изменения и их значениями и называется равномерной. *Числа* ν_i *называются* основаниями, ε_i — перекрытиями.

Основание определяет, на сколько равных отрезков делится предыдущий отрезок, а перекрытие — на какую долю пересекаются два соседних отрезка.

Ниже мы сосредоточимся на равномерных системах, хотя результаты могут быть перенесены на неравномерные, но с утяжелением выкладок. Заметим, что

неравномерные системы представляются также перспективными. Понятие естественного отрезка переносится на брауэровы системы.

В оригинальной брауэровой системе $\langle 0, 1, 2, 0.5\rangle$ отрезок $[0, 1]$ делится на пересекающиеся отрезки $[0, 2/3]$, $[1/3, 1]$, те в свою очередь на такие же, например, $[0, 2/3]$ на $[0, 4/9]$, $[2/9, 2/3]$. В оптимальной брауэровой системе $\langle 0, 1, 3, 0.5\rangle$ отрезок $[0, 1]$ делится на пересекающиеся отрезки $[0, 0.5]$, $[0.25, 0.75]$, $[0.5, 1]$.

Th. **Теорема 6.2. (Вычислимость).**

Все вычислимые функции действительных чисел вычислимы в любой брауэровой системе на любом отрезке, на котором они ограничены.

Доказательство.

Согласно принципу конечной информации, любое приближение к результату (x f) вычисляется по приближению к x, которое является отрезком с рациональными концами. Пусть нам надо вычислить i-тый знак приближения к $(x\ f)$. Перекрытие для i-тых знаков гарантирует, что два любых соседних отрезка цифр пересекаются по длине $\epsilon_0 > 0$. Вычислив приближение к результату с точностью до $\epsilon_0/2$, мы с гарантией попадаем внутрь одного из отрезков цифр. Для его вычисления достаточно некоторого приближения к аргументу $[a_0, b_0]$. Поскольку длины отрезков уменьшаются, вычислив аргумент с количеством цифр таким, что перекрытие будет меньше $b_0 - a_0$, получаем гарантированно нужные цифры результата. ■

Заметим, что это рассуждение не зависит от конкретных систем представления аргумента и результата. В частности, теорема влечет вычислимость умножения и сложения, а деления на любом отрезке, не включающем 0.

Рассмотрим самый практически важный случай равномерной системы с $\varepsilon = 0.5$ и проанализируем выполнение операции сложения. Поскольку любой отрезок линейно преобразуется в любой другой, достаточно рассмотреть аргументы на отрезке $[0, 1]$. Соответственно, результат сложения будет на отрезке $[0, 2]$.

Введем три важных величины для цифр первого разряда. δ — сдвиг каждого отрезка увеличенной на 1 цифры относительно предыдущей, a — длина отрезка цифры, ϵ — длина перекрытия. Они связаны соотношениями:

$$\delta = a - \epsilon \quad \epsilon = a \cdot \varepsilon = \delta = a/2 \quad (n-1)\cdot\delta + a = 1$$

Тем самым длина отрезка цифры в системе с основанием n и перекрытием 0.5 $\dfrac{2}{n+1}$, $\delta = \dfrac{1}{n+1}$. Например, в двоичной длина отрезка $2/3$, в троичной $1/2$, в четвертичной $2/5$.

В брауэровой системе представление чисел неоднозначно для всех, кроме концов интервала системы. Например, в двоичной системе $1/2$ представляется и как $01010101\ldots$, и как $10101010\ldots$.

При сложении чисел, если сумма первых знаков $i_1 + j_1$ чётна, то получившийся интервал в точности совпадает с интервалом цифры $\dfrac{i_1 + j_1}{2}$. Поэтому в данном

случае можно обойтись вообще без переносов. Например, в троичной системе $2+2=2$, $0+2=1$. Случай нечётной суммы цифр намного интереснее, и по-разному решается в системах с основанием 2, 3 и более высоких.

Сумма интервалов сдвигается относительно интервала суммы на δ, то есть в случае основания 3 на половину шага интервала суммы. Если рассмотреть следующие знаки слагаемых, то, поскольку длина интервалов уменьшается в два раза, выполнены следующие два правила (их области действия пересекаются; в некоторых случаях можно применить любое из них):

Если $i_2 + j_2 \leqslant \dfrac{n+1}{2} = 2$то сумма $\dfrac{i_1 + j_1 - 1}{2}$.

Если $i_2 + j_2 \geqslant \dfrac{n+1}{2} = 2$то сумма $\dfrac{i_1 + j_1 + 1}{2}$.

В частности, в случае $0+1$, если второй знак одного из чисел 0, то вторым знаком другого числа можно не интересоваться и выдать 0. Если он 2, то можно выдать 1. А если оба вторых знаки 1, можно выдать любой из этих результатов.

Оба этих правила работают и в системах с основанием больше 3, но поскольку уменьшение длины результирующего интервала при учете второго знака становится больше δ, то интервал, когда возможны оба решения, начинает увеличиваться, а случаев, когда не требуется второго знака, становится все меньше.

В двоичной системе, из-за того, что уменьшение отрезка при учёте второго знака меньше половины, требуется учёт трёх знаков числа.

$\stackrel{def}{=}$ **Определение 6.4. (Аккуратное представление).**

Представление числа x аккуратное, если x не менее чем на долю $\varepsilon/2$ отстоит от границ (не совпадающих с границами предыдущего интервала) любого интервала, определяемого его очередным знаком, и попадает в крайний интервал, лишь если он не попадает ни в какой другой.

Любое число из интервала $[\epsilon/2, 1-\epsilon/2]$ может быть представлено аккуратно. Если система не двоичная, то это представление вычислимо.

$\mathbb{Th}$. **Теорема 6.3. (Параллельность).**

В любой равномерной брауэровой системе, где $\varepsilon = 0.5$ и основание больше 2, сложение чисел, представленных аккуратно, вычислимо за один такт, полностью параллельно.

Перевод чисел из брауэровых систем в обычные тривиален. Интервал последней цифры дает доказательно правильную верхнюю и нижнюю границу результата.

Практический вывод. Для быстрых вычислений лучше всего троичная система.

Список литературы

1. Непейвода Н. Н. Вызовы конструктивизма. — М.: Издательство, 2012. — 700 с.
2. Успенский В.А. Лекции о вычислимых функциях — ГИФМЛ, 1960, 492 с.

3. Landauer, R. (1961). Irreversibility and Heat Generation in the Computing Process. // IBM J. Res. Develop. 5 (3): p. 183–191.
4. Nepejvoda, N. N. Three-headed Dragon. https://docs.google.com/document/d/1hGzUB3p3j2zYcksoUnxtv7QamHzcgYVYr66iJJiMOhY
5. Berut A., Arakelyan A., Petrosyan A., Ciliberto S., Dillenschneider R., Lutz E. Experimental verification of Landauer's principle linking information and thermodynamics> // Nature **483**, p. 187–189. 2012.
6. Bennett C.H. Logical reversibility of computation // IBM J. Res. Develop. — 1973. — Vol. 17, N 6. — P. 525—532.
7. Glushkov, V. M. Abstract Theory of Automata // Cleaver-Hume Press Ltd., 1963.
8. Maurer, W. D. A theory of computer instructions. Journal of the ACM, vol. 13, No 2 (1966) pp. 226–235.
9. Gluschkow W. M., Zeitlin G. E., Justchenko J. L. — Algebra. Sprachen. Programmierung. — Akademie-Verlag, Berlin 1980. — 340 p.
10. Ivanov, P. M. Algebraic modelling of complex systems. — Moscow, 1996. — 274 p.
11. Alban Ponse and Mark B. van der Zwaag. An Introduction to Program and Thread Algebra. LNCS 3988, 2006, pp 445–488.
12. Bergstra J. A., Bethke I, Ponse A. Program Algebra and Thread Algebra. Amsterdam, 2006, 114p.
13. Nepejvoda N. N. *Reversivity, reversibility and retractability.* Third international Valentin Turchin workshop on metacomputation. Pereslavl: 2012. pp 203–215.
14. *Brouwer L.E.J.* Besitzt jede reele Zahl eine dezimalbruchentwicklung?// Proc. Acad. Amsterdam, **23**, p 949–954.
15. *Гейтинг А.* Интиционизм. Введение. М: 2010, 163 с.
16. Girard J.-Y. *Linear Logic.* Theoretical Computer Science **50**, 1987, 102 pp.
17. Malcev, A.I. Algebraic Systems. Springer-Verlag, 1973, ISBN 0-387-05792-7.
18. Mitchell, J. C. Foundation for programming languages, MIT, 1996.
19. Barenfregt, H. The lambda-calculus. Its syntax and semantics. Elsevier 1984, ISBN 0-444-87508-5.
20. Nepejvoda, N. N. Reversive constructive logics. Logical Investigations, **15**, 150–169 (2008).
21. Axelsen, H. G., Glück, R. What do reversible programs compute? FOCSSACS 2011, LNCS 6604, pp. 42–56, 2011
22. http://www.muppetlabs.com/ breadbox/bf/
23. Böhm, C., Dezani-Ciancaglini, M. Can syntax be ignored during translation? In: Nivat M. (ed.) Automata, languages and programming. North-Holland, Amsterdam, 1972. p.197–207.
24. Nepejvoda, N. N. Abstract algebras of different classes of programs. Proceedings of the 3rd international conference on applicative computation systems (ACS'2012), 103–128.
25. Nepejvoda, N. N. Algebraic approach to control. Control Sciences, 2013, N 6, 2-14.