© 2012 г. Н. Н. НЕПЕЙВОДА
(Институт программных систем имени А. К. Айламазяна РАН)

ОТ ЧИСЛЕННОГО МОДЕЛИРОВАНИЯ К АЛГЕБРАИЧЕСКОМУ

Algebraic structures for some practically interested classes of program are described. Those classes are characterized not by program construction used but by classes of transformations. It is shown that actions of programs not always can be described as functions but functional properties of programs induce properties of actions (e. g. invertible programs induce invertible actions as was demanded by E. Bennett). This approach has some interconnections with ideas of PGA but is different because is arrowed up to co-ordination of high level programming and non-functional pragmatic actions.

Программные и алгоритмические алгебры практически одновременно начали рассматриваться в СССР и США. Глушков [1] определил алгоритмические алгебры, основанные на операторах структурного программирования. Маурер [2] — абстрактные программные алгебры вычислений PGA, основанные на модели, близкой к машине Тьюринга. Позже появились разные виды алгоритмических и динамических алгебр, но все они следуют двум основным направлениям, заложенным Глушковым и Маурером. В связи с этим невозможно отделить принципиально важные и общие аспекты от частностей конкретной модели вычислений.

Как показано, например, в [6, 7], во многих случаях практически важные классы вычислений приводят к заведомо неполным по Тьюрингу системам. Поэтому в данной лекции строятся алгебраические модели программ таким образом, чтобы они могли единообразно охватить все многообразие возможных программных систем: от автоматных до полных по Тьюрингу, и далее, до использующих физические устройства и физические датчики и тем самым заведомо выходящих за пределы тьюринговской вычислимости.

Поскольку прослеживается, что в ближайшее время будут играть важнейшую роль инъективные и обратимые программы, эти классы программ анализируются отдельно. Кроме того, наш подход позволяет бросить новый взгляд на нильпотентные (доказуемо завершающиеся) программы и на автоматное программирование.

Внимание! Для большей естественности соотношения с алгебраическими понятиями и для того, чтобы сразу выбить читателя из привычной численной и бинарной парадигмы, мы пишем функцию после ее аргумента и используем принятую в λ-исчислении и функциональном программировании запись применений функций как списков. $(x\ f)$ — применение функции f к аргументу x. При описании конкретных отображений мы пользуемся общепринятыми в современной теоретической информатике λ-обозначениями. Это не означает, что рассматриваемые нами программные и прочие системы обязаны базироваться на λ-исчислении.

Каждой программе сопоставим (частичное) отображение данных, производимое ею. В самом общем случае алгебраическая структура вычисляемых программами отображений, образует полугруппу, операцией в которой является композиция программ (или их последовательное исполнение, что в данном случае то же самое).

Если операция ассоциативна, все скобки можно убрать, и выражение представляется как линейный список. Более того, любая ассоциативная операция допускает интерпретацию как композиция функций. Это следует из классической теоремы.

Теорема 1. (Теорема о представлении для полугрупп) Каждая полугруппа изоморфна полугруппе функций с операцией композиции. [4, стр. 90]

Таким образом, говорить о том, что у нас имеется совокупность отображений, которые мы можем применять последовательно, то же самое, что говорить: у нас есть полугруппа. Тем самым мы получаем заодно и критерий, чтобы проверить, можно ли данную совокупность программных действий рассматривать как функции: ассоциативность композиции.

Пример 1. Рассмотрим совокупность действий программной системы. Пусть для некоторых действий a определено правое обратное a^{-1}: отмена только что произведенного действия a. Если же предшествующее действие было не a, то a^{-1} ничего не делает. Тогда $a * a^{-1} = e$, но $a * (e * a^{-1}) = a * e = a$, $(a * e) * a^{-1} = a * a^{-1} = e$.

Аналогичными же рассуждениями можно показать, что общая операция undo (отменить предыдущее действие) также не может рассматриваться как функция.

Тем самым мы видим, что действия — понятия другой природы, чем преобразования, и нуждаются в других формализмах.

Таким образом, *абстрагирование программ до отображений, до вычисляемых ими функций, не всегда является корректным действием.* Отображение, определяемое программой, дает порою искаженное представление о программе.

Из-за столь важной роли полугрупп, частные понятия из теории полугрупп имеют фундаментальный смысл в информатике. Единица e означает действие «ничего не делать». Поскольку в реальности такое действие не всегда возможно (например, мы не можем остановить время), не всегда нужно предполагать, что полугруппа является моноидом.

Если функции взаимно-однозначны, то появляются обратные элементы и полугруппа превращается в группу. Более того, любая полугруппа взаимно-однозначных функций легко пополняется до группы. Группа означает, что для любого действия есть полное обратное, которое может и предотвратить его, и аннулировать его результаты после выполнения

$$a \circ a^{-1} = e; \qquad a^{-1} \circ a = e. \tag{1}$$

Область применения групп также широка. Изоморфизмы алгебраической структуры на себя образуют группу. Симметричность объекта означает, что есть группа преобразований, при которой он сохраняется (например, сдвиги и повороты для мозаики, перестановки переменных для фрагмента программы или формулы и так далее...). Кристаллы описываются группами симметрий. Динамические системы — группами инвариантов и сдвигов. Примеры неисчислимы...

Видно, что не любая совокупность программ и действий может быть описана их функциями. Таким образом, пространство программ в общем случае не является по-

лугруппой. Но синтаксическая структура программ, которая важна при их преобразованиях, полугруппой является.

Заметим, что любой группоид порождает полугруппу действий (которая также не полностью описывает действия программ).

Определение 1. Действие последовательности $f_1, \ldots, f_n$ *элементов группоида — функция*

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

Предложение 1. Действия любого группоида являются полугруппой относительно операции композиции функций.

Доказательство. Композицией $\varphi_{f_1;\ldots;f_n}$ и $\varphi_{g_1;\ldots;g_k}$ является $\varphi_{f_1;\ldots;f_n;g_1;\ldots;g_k}$. Композиция функций всегда ассоциативна.
□

На данном примере видно, почему мы предпочли несколько необычный порядок записи функции и ее аргументов: в сложных случаях он намного естественнее.

Рассмотрим теперь с содержательной точки зрения якобы совершенно ясное понятие: применение программы к аргументу $x \star f$. Эта операция для большинства программных систем неассоциативна, как было показано выше, она согласована с композицией, но не сводится к ней. Поскольку любой элемент у нас может играть роль и функции, и данного ($f \star x$ столь же законная операция), алгебра применений заодно дает возможность описать преобразования программ.

Мы приходим к алгебре, являющейся бигруппоидом с двумя операциями. Одна из них полугрупповая операция композиции $\circ$, вторая — неассоциативная операция применения программы $\star$. Получающуюся полугруппу по композиции будем обозначать большой жирной буквой, а всю алгебру — прозрачной. Они связаны следующими тождествами.

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

$$(0 \star x) = (x \star 0) = 0 \tag{3}$$

$$(x \star e) = e \tag{4}$$

Как бы мы ни определили понятия и композиции и применения, они должны быть связаны вышеприведенными тождествами. В остальном мы, как и всегда при алгебраическом подходе, свободны в интерпретации. Во всех известных автору программных, человеко-машинных и аппаратных системах эти свойства выполнены.

Такие алгебры назовем GAPS (Generic algebraic program structure).

Напомним понятие обогащения алгебраической системы. Пусть дана сигнатура σ_1, расширяющая сигнатуру σ. Алгебра $\mathbb{A}_1$ сигнатуры σ_1 является обогащением алгебры $\mathbb{A}$ сигнатуры σ, если носители, константы и операции из σ остаются без изменения.

Пусть имеется некоторый гомоморфизм $\alpha : \mathbb{G} \to (G \to G)$ нашей полугруппы $\mathbb{G}$ в полугруппу отображений ее носителя, такой, что $(e\ \alpha) = \lambda x.\,x$, $(0\ \alpha) = \lambda x.\,0$. С точки зрения программиста, этот гомоморфизм задает интерпретатор или транслятор наших программ, перерабатывающий запись программы в исполняемый модуль. Синтаксически различные программы могут порождать один и тот же исполняемый модуль, так что считать это отображение α изоморфизмом нельзя.

Предложение 2. Каждая полугруппа $\mathbb{G}$ может быть обогащена до GAPS таким образом, что $x \star f = (x\ (f\ \alpha))$.

Предложение 3. В любой GAPS для любого действия группоида φ имеется элемент α, такой, что $(x \star \alpha) = (x\ \varphi)$.

Пример 2. Заметим, что некоторые «естественные» предположения о GAPS разрушают систему. Например, почему нет тождества $(e \star f) = f$? Потому что тогда обе операции совпадают:

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

Рассмотрим возникшую ситуацию содержательно: почему это 0 может оставаться нулем, а единица при действиях оставаться сама собой не всегда может? 0 с точки зрения интерпретации — ошибка. Дальше либо с нею внутри алгебры делать уже нечего: система вылетела. e может быть интерпретирована как программа, ничего не делающая и не тратящая ресурсов (например, пустая). Но преобразователь программ вполне может выдать какой-то сложный код, исходя из пустых данных. Смотри дальше пример 6.

Теперь рассмотрим важный вопрос: когда можно некоторую систему преобразований программ соединить с данной синтаксической полугруппой программ, тем самым образовав единый язык метапрограммирования?

Определение 2. Полугруппа $\mathbb{F}$, порожденная совокупностью функций $\mathbf{F}$ — минимальный тип функций, включающий $\mathbf{F}$ и замкнутый относительно композиций, то есть

$$\forall f, g\,(f \in \mathbb{F} \& g \in \mathbb{F} \supset f \circ g \in \mathbb{F})$$

Определение 3. Пусть задана некоторая совокупность действий $\mathbb{A}$ как функции на носителе полугруппы $\mathbb{G}$. Она консервативна*, если имеется GAPS $\mathbb{AG}$, которая является обогащением $\mathbb{G}$, такая, что каждое $\varphi \in \mathbb{A}$ представляется в ней как действие группоида: $(x \star \alpha) = (x\ \varphi)$ для некоторого α. Она* консервативна над подполугруппой $\mathbb{G}_0$*, если она консервативна и в $\mathbb{AG}$ $a \star b = a \circ b$ для всех элементов $\mathbb{G}_0$. Она* слабо допустима*, если имеется полугруппа $\mathbb{G}_1$, подполугруппой которой является $\mathbb{G}$, и функции из $\mathbb{A}$ могут быть продолжены на $\mathbb{G}_1$ таким образом, что полученная $\mathbb{A}_1$ консервативна над $\mathbb{G}$. Она* допустима*, если она слабо допустима и в $\mathbb{A}_1$ остаются истинными все факты вида $f_1 \circ \cdots \circ f_n = g_1 \circ \cdots \circ g_k$, истинные в $\mathbb{A}$.*

Очевидно, что самое сильное из этих понятий — консервативность над полугруппой. Далее идут консервативность, допустимость, слабая допустимость. Рассмотрим простейшие условия возможности всех четырех обогащений и расширений.

Предложение 4. Совокупность действий $\mathbb{A}$ консервативна тогда и только тогда, когда ее замыкание изоморфно подполугруппе $\mathbb{G}$.

Доказательство. По предложению 3, если обогащение удалось осуществить, то любой элемент полугруппы, порожденной $\mathbb{A}$, представляется как действие некоторого элемента $\mathbb{G}$. Значит, мы изоморфно вложили замыкание $\mathbb{A}$ в $\mathbb{G}$.

Обратно, если замыкание $\mathbb{A}$ вкладывается в $\mathbb{G}$, то просто сопоставим каждому действию из $\mathbb{A}$ соответствующий ему при изоморфизме элемент $\mathbb{G}$.
□

Заметим, что важнейшим фактором является вложение структуры полугруппы действий, а не самих действий.

Пример 3. Если у нас имеется всего одно действие сильной инверсии программ $\mathbf{M}$, обладающее свойством $((a\ \mathbf{M})\ \mathbf{M}) = a$, то обогатить полугруппу программ этим действием удается тогда и только тогда, когда у нас имеется элемент порядка 2, то есть такая программа, что $f \neq e \& f \circ f = e$. Как эта f действует на данные, нам все равно. Важно, что мы можем доопределить ее как операцию метапрограммирования без разрушения общей структуры вычислений.

Предложение 5. Совокупность действий $\mathbb{A}$ консервативна над $\mathbb{G}_0$ тогда и только тогда, когда имеется такое изоморфное вложение ψ ее замыкания в $\mathbb{G}$, что для каждого действия f, такого, что $(f\ \psi) \in \mathbb{G}_0$, выполнено $(a\ f) = (a \circ (f\ \psi))$.

Пример 4. Например, пользуясь этим критерием, можно проверить возможность обогащения языка до языка метавычислений, рассматривая строки как программы, а подъязык, работающий с числами, оставляя неизменным.

Теорема 2. Любая совокупность действий $\mathbb{A}$ допустима над любой $\mathbb{G}$.

Пусть $\mathrm{Th}\{\mathbb{GA}\}$ теория, получающаяся добавлением кTh матрицы полугруппы $\mathbb{G}$, то есть всех истинных и ложных фактов об элементах этой полугруппы.

Теорема 3. Совокупность действий $\mathbb{A}$, описываемая теорией Th_0, допустима над $\mathbb{G}$ в том и только в том случае, когда теория $\mathrm{Th}\{\mathbb{GA}\}$ непротиворечива.

Принципиально данное решение полное. Но на практике оно, конечно же, редко применимо. Однако есть один практический критерий, который следует из данной теоремы.

Практическое следствие. *Если система преобразований программ разрушает свойства каких-то уже имеющихся программ или заставляет отождествить разные программы, она принципиально некорректна.*

Пример 5. (Постановка задачи принадлежит С. Мешвелиани) λ-исчисление лежит в основе современной математический теории полных по Тьюрингу программных систем. Имеется модель стратифицированного λ-исчисления как GAPS. Напомним, что λ- выражение называется стратифицированным. если в нем можно корректно расставить типы всех переменных и констант.

Комбинаторная логика естественно и взаимно-однозначно переформулируется как группоид действий: применение терма к терму $(t\ r)$ записывается в наших обозначениях $(r \star t)$. Определяющие равенства для констант $\mathbf{K}$ и $\mathbf{S}$ переписываются в форме

$$(x \star (y \star \mathbf{K})) = y; \tag{5}$$

$$(x \star (y \star (z \star \mathbf{S})) = ((x \star y) \star (x \star z)). \tag{6}$$

В комбинаторной алгебре определяется комбинатор композиции $\mathbf{B}$:

$$\mathbf{B} \triangleq (\mathbf{K} \star ((\mathbf{S} \star \mathbf{K}) \star S)). \tag{7}$$

Преобразуя согласно определяющим равенствам для $\mathbf{K}$ и $\mathbf{S}$, получаем:

$$\begin{gathered}(x \star (f \star (g \star (\mathbf{K} \star ((\mathbf{S} \star \mathbf{K}) \star \mathbf{S}))))) = \\ (x \star (f \star ((g \star \mathbf{K}) \star (f \star (\mathbf{S} \star \mathbf{K})))))) = \\ (x \star (f \star ((g \star \mathbf{K}) \star \mathbf{S}))) = \\ ((x \star f) \star (x \star (g \star \mathbf{K}))) = \\ ((x \star f) \star g).\end{gathered}$$

Таким образом, $(f \star (g \star \mathbf{B}))$ определяет последовательное применение (композицию) f и g.

Добавим $\mathbf{B}$ как новую константу с определяющим равенством (7). Такое расширение теории с привычной точки зрения эквивалентно, поскольку добавили определимое понятие, но оно избавляет от нежелательного эффекта: как бы мы ни определили $\mathbf{B}$, $(f \star (g \star \mathbf{B}))$ не будет нормальной формой, поскольку последняя константа в определении заведомо применится (даже если она $\mathbf{S}$, требующая три аргумента, то, поскольку определение не сводится к самой $\mathbf{S}$, третий аргумент будет взят из текста определения).

Далее, два выражения

$$(f \star ((g \star (h \star \mathbf{B})) \star \mathbf{B})) \qquad ((f \star (g \star \mathbf{B})) \star (h \star \mathbf{B}))$$

функционально и вычислительно эквивалентны, но находятся в нормальной форме и поэтому не могут быть преобразованы друг в друга. Равенство между ними также постулируем как тождество.

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

Это тождество может быть эффективно реализовано как одностороннее и заведомо фундированное правило переписывания левой части в правую. Таким образом, носитель алгебры у нас множество нормальных форм выражений в комбинаторной логике, пополненной комбинатором $\mathbf{B}$, нормализованных применением правила, соответствующего равенству (8). Композицией двух выражений считается $(f \star (g \star \mathbf{B}))$.

Это определение корректно, поскольку стратифицированные λ-выражения всегда имеют нормальную форму, и композиция стратифицированных выражений без свободных переменных стратифицирована.

Построение завершено.

Таким образом, мы показали, что GAPS позволяет адекватно описать одну из самых общих теоретических концепций современной информатики.

Если перейти к бестиповому λ-исчислению, то конструкция так же хорошо работает, если значением выражения, не имеющего нормальной формы, считать 0.

Тем самым на этапе принципиальной проработки теоретических концепций представление «конкретных» и «практических» алгоритмических языков становится излишним: они теоретически сводимы к λ-исчислению.

Пример 6. Рассмотрим еще один пример превращения программной системы в абстрактную алгебру. Пусть у нас есть полный по Тьюрингу алгоритмический язык, в котором программы и данные являются строками символов в одном и том же алфавите. Пусть соединение строк дает композицию функций программ. Пусть пустая строка — ничего не делающая программа. Примером такого языка может служить Brainfuck. Добавим константы 0 и e. первая из них является значением вылетающей или зацикливающейся программы, вторая — пустая строка.

Композиция программ определяется как соединение строк. А вот действие строки $a \star f$ определим следующим образом. Вычисляется строка, являющаяся значением f на a. Если f выдала ошибку или получившаяся строка синтаксически некорректна как программа, значением является 0. В противном случае получившаяся программа.

В этом примере видно, что результатом действия над пустой программой может быть любая другая программа. А вот результат действия пустой программы — та же программа, что была.

Если в полугруппе есть единица, то универсальная функция, описываемая в нашей алгебре тождеством

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

тривиальна: $\mathbf{U} = e$. Зато специализатор программ, описываемый как

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

уже нетривиален почти всегда. И на этом тождестве видно, что специализатор является одним из видов универсальной функции.

Неподвижная точка программ, определяемая как

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

тривиальна, если в полугруппе есть 0: она просто тождественно равна 0.

Алгебра полностью обратимых программ (AFIP)

Обратимые действия составляют группу. Преобразования программ также должны быть в этой обстановке обратимы, и мы получаем характеристику алгебр полностью обратимых программ как GAPS с дополнительной константой M (обращение программ) и дополнительными аксиомами:

$$x \circ (x \star M) = e \qquad (x \star M) \circ x = e \tag{9}$$

Из этих аксиом автоматически следует биективность операции применения программ $\star$:

Предложение 6. Отображение $\lambda x.(x \star f)$ биективно для любого f.

Наличие этих двух аксиом автоматически превращает полугруппу в группу: $(x \star M)$ оказывается обратным элементом для x. Оно же окончательно различает операции и функции: в большинстве групп нет такого элемента, который при умножении на все остальные давал бы обратные им.

Предложение 7. Каждая группа $\mathbb{G}$, в которой имеется элемент порядка 2 (то есть такой x_0, что $x_0 \circ x_0 = e$) может быть обогащена до AFIP.

Предложение 8. Группа $\mathbb{G}$ обогащается до AFIP с сохранением групповой операции $\circ = \star$ тогда и только тогда, когда она коммутативна и каждый элемент в ней имеет порядок 2.

Из известных результатов теории групп следует утверждение:

Предложение 9. Группа $\mathbb{G}$ обогащается до AFIP с сохранением групповой операции $\circ = \star$ тогда и только тогда, когда она изоморфна полному прямому произведению некоторого числа групп вычетов по модулю 2 $\mathbb{Z}_2$ либо, в бесконечном случае, его подгруппе, в которой множества индексов, где стоят нули перемножаемых групп, принадлежит некоторому фильтру.

К несчастью, в данном случае вычисления в бинарной памяти оказываются слишком ограниченными: мы не имеем права организовывать взаимодействие разных битов памяти между собой.

Предложение 10. Пусть группа $\mathbb{G}$ не является группой порядка 2. Пусть G_0 — ее подгруппа, не совпадающая с $\{e\}$ и не являющаяся группой порядка 2 (хотя она может содержать элементы порядка 2). Существует алгебра AFIP $\mathbb{G}_0$ такая, что для всех $f \in G_0$ $(x \star f) = (x \circ f)$ тогда и только тогда, когда есть элемент порядка 2, не входящий в $\mathbb{G}_0$.

Заметим еще одну особенность алгебраических определений. Маленькая «эквивалентная» переформулировка может полностью изменить свойства алгебраической системы. Например, если заменим равенства композиций программы и ее обращения единице на равенства их вычислительного эффекта тождественному преобразованию

$$((x \star f) \star (f \star M)) = x \qquad ((x \star (f \star M)) \star f) = x, \tag{10}$$

то доказательство биективности всех действий сохранится, а вот условия расширяемости (предложения 7 и 10) станут более «комфортными»: необходимо и достаточно наличие хотя бы одного элемента, не имеющего нечетный порядок.

Алгебра инъективных частично обратимых программ (APIP) Эта алгебра соответствует классу инъективных программ, описанному в работе Глюка и Аксельсена [7].

Поскольку программы не предполагаются всюду определенными (иначе для инъективных программ невозможно было бы определить обращение), в полугруппе обязательно имеется константа 0 (**ошибка**). Поскольку есть ничего не делающие программы, полугруппа является моноидом и имеется единица e.

$$(0 \star x) = (x \star 0) = 0 \tag{11}$$

$$f \neq 0 \supset (f \star M) = f^{-1} \tag{12}$$

$$\forall x, f((x \star f) \neq 0 \supset (x \star (f \circ f^{-1})) = x \circ f \circ f^{-1}) \tag{13}$$

(14)

Теперь установим аналог инъективности для действий.

Предложение 11. Выполнено следующее свойство относительной инъективности:

$$\forall a, b, f\,((a \star f) = (b \star f) \equiv\ (a \star (f \circ f^{-1})) = (b \star (f \circ f^{-1}))).$$

Очевидно.
□

Предложение 12.

$$\forall x, f\,((x \star f) = 0 \equiv x \circ f = 0).$$

Доказательство. Пусть $(x \star f) = 0$, $f \neq 0$. Тогда

$$0 = (0 \star f^{-1}) = ((x \star f) \star f^{-1}) = ((x \star (f \circ f^{-1})) = x \circ f \circ f^{-1} = x \circ f \circ f^{-1} \circ f = x \circ f.$$

Повторяя эту последовательность преобразований в обратную сторону, получаем вторую часть доказываемой эквивалентности.
□

Мы видим, что в отношении равенства нулю результата две наших операции совпадают. Как отметил Е. Кочуров в личной переписке: “Раньше длинно говорил «источники наведенных ошибок программы при сборке». Теперь буду говорить коротко и ясно: «делители нуля».”

Теперь методами аналогичными предыдущим параграфам, можно доказать два утверждения

Предложение 13. Каждая инверсная полугруппа с нулем, единицей и хотя бы одним элементом второго порядка, может быть продолжена до APIP.

Предложение 14. Пусть G_0 — подмоноид инверсной полугруппы с нулем $\mathbb{G}$, не совпадающий с нею самой и с $\{0, e\}$ и в $G \setminus G_0$ есть хотя бы один элемент второго порядка. Тогда его можно продолжить до APIP $\mathbb{G}_0$, такой, что для всех $f \in G_0$ $(x \star f) = (x \circ f)$.

Нильпотентная логика как алгебраическая структура (ANP)

Нильпотентная логика была введена как логика существенно необратимых действий, таковы, например, любые действия (даже бездействие), если принимать в расчет текущее время. Рассмотрим понятие строго нильпотентной полугруппы, подобное тому, которое исследовалось в работе А. И. Мальцева [9].

Определение 4. $\mathbb{G}$ — строго нильпотентная полугруппа, если для каждой бесконечной последовательности элементов a_i существует такое n, что

$$\prod_{i=1}^{n} a_i = 0.$$

Нильпотентная программная алгебра (ANP) над $\mathbb{G}$ — GAPS над $\mathbb{G}$, сохраняющая 0.

В строго нильпотентной полугруппе не может быть единицы и идемпотентов, кроме 0 (то есть таких x, что $x \circ x = x$). Все элементы необратимы, даже в полугрупповом смысле. Из определения GAPS следует, что каждая бесконечная цепочка действий также в некоторый момент дает ошибку.

Методами предыдущих параграфов доказываются два утверждения:

Предложение 15. Каждая строго нильпотентная полугруппа с нулем $\mathbb{G}$ может быть обогащена до APIP.

Предложение 16. Пусть G_0 — подполугруппа $\mathbb{G}$. Тогда существут APIP $\mathbb{G}_0$, такая, что для всех $f \in G_0$ $(x \star f) = (x \circ f)$.

Поскольку здесь нет выделенных действий, леммы не имеют ограничений.

Ценность таких алгебр в том, что необратимые действия порождают автоматное программирование, а программные алгебры, соответственно — автоматы высших типов.

Некоторые замечания о связях алгебр с программированием

Прежде всего, об истоках неассоциативности в программировании. В функциональном программировании все очевидно: применить оператор к программе, а затем результирующую программу к данным, не то же самое, что применить обе эти программы друг за другом. Если у нас есть преобразования программ, тоже более или менее ясно: преобразовав программу, мы получаем новое действие. В якобы линейных последовательностях действий с частичной обратимостью (undo) неассоциативность замаскиро-

вана под «медвежьи услуги». Система может неявно объединить несколько действий в блок и отменить блок целиком (так, например, делает Word при отмене операций посимвольного редактирования). Явно блочность и неассоциативность возникает, когда имеются действия типа отката к предыдущей контрольной точке.

Даже простая групповая реализация полностью обратимых действий выходит за пределы группы, поскольку с необходимостью возникает операция обращения сегмента программы, не являющаяся внутренней групповой [8].

Теперь о необходимости сохранять ассоциативную структуру. Во-первых, она гарантирует возможность объединения блока действий в одно действие. Во-вторых, она лучше соответствует структуре текста программы или записи последовательности действий.

В последнее время намечается необходимость перехода от примата численных данных к данным нечисленным. Это означает переход от численных функций, математического анализа и дифференциальных уравнений к алгебраическим и логическим структурам. Даже в современной качественной теории дифференциальных уравнений основные преобразования алгебраические, но по традиции исходные данные и конечные результаты «для лучшего понимания» представляются в численной форме. Тем самым теоретическая математика в принципе подготовлена к переходу от аппарата численных моделей к аппарату моделей алгебраико--логических. Самое главное, что уже накоплен багаж методов перевода задач из одной формы в другую.

Алгебраические вычисления представляются гораздо более радикальным шагом в супервычислениях, чем «параллелизм». Даже в идеальном случае распараллеливание дает линейное ускорение программы, а переход к более адекватным высокоуровневым структурам данных может давать выигрыш в башню экспонент раз и по длине программы, и по скорости ее исполнения. Алгебраические вычисления «распараллелены» с самого начала, что делает излишней задачу преобразования к параллельной форме программы, изначально написанной в терминах последовательных действий. С точки зрения системного анализа последняя задача была просто неправильно поставлена, что диагностирует неизлечимую генетическую болезнь вечно молодого и перспективного направления параллельных вычислений.

В программах обычно целесообразно выделять команды и данные. В традиционных формализациях команды являются константами и программа совершает пробег по этим константам. В обратимом и полуобратимом программировании, а также в программировании, где действия могут влиять на команды, такой подход недостаточен. Команды в обратимом программировании должны также работать полностью обратимо. В других случаях команды должны обрабатываться более аккуратно, чем данные. Этим условиям удовлетворяет конструкция полупрямого произведения, известная для групп (см., напр., [**?**]) и обобщаемая здесь на полугруппы и программные алгебры.

Определение 5. Полупрямое произведение полугруппы команд $\mathbf{C}$ *и полугруппы данных* $\mathbf{D}$ *есть полугруппа, носитель которой* $C \times D$*, а операция умножения определяется следующим образом. Задан гомоморфизм* $\varphi : \mathbf{C} \to \mathrm{Hom}(\mathbf{D}, \mathbf{D})$ *полугруппы команд в полугруппу гомоморфизмов полугруппы данных.*

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

Полупрямое произведение программных алгебр $\mathbb{C}$ *и* $\mathbb{D}$ *определяется аналогично с до-*

полнительным условием на $\star$*:*

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

Полупрямое произведение является естественным методом структурирования алгебры на программу и данные. Оно, в частности, возникает при моделировании условных операторов и циклов в обратимых вычислениях, базирующихся на группах (смотри, например, [8]). Оно часто позволяет совместное исполнение в широких пределах (его критерием является, в частности, равенство единичному отображению $(c\ \varphi)$ для некоторого подкласса команд).

Как ни странно, казалось бы, самая «безобидная» из алгебраических конструкций — факторизация — может полностью разрушить структуру вычислений, например, их обратимость или полуобратимость. Если нужна конструкция, ее нужно собрать из допустимых «кирпичиков» или получить отрезанием лишнего.

И, наконец, исключительная гибкость алгебраических понятий (они могут интерпретироваться самыми разными способами) открывает перед алгебраическим программированием невиданные для обычного операторного перспективы.

Основной трудностью при переходе к алгебраическим моделям вычислений является полная смена всей парадигмы описания систем и программирования. В некотором смысле мы возвращаемся к затоптанным грубой силой гибридным машинам: алгебраический специализированный процессор и тупая числовая и булева мельница, как управляющий.

Автор благодарен за неоднократные стимулирующие обсуждения и нахождение неточностей Р. Глюку, С. Мешвелиани, А. Непейвода.

СПИСОК ЛИТЕРАТУРЫ

1. Глушков В. М. Синтез цифровых автоматов. — М.: Физматгиз, 1962. — 476 с.
2. Maurer W. D. A theory of computer instructions. Journal of the ACM, 13(2): 226–235, 1966.
3. Глушков В. М., Цейтлин Г. Е., Ющенко Е. Л. Алгебра, языки. программирование. — Киев, Наукова думка, 1978. — 320 с.
4. Мальцев А. И. Алгебраические системы. М.: Наука, 1970. — 392с.
5. Митчелл Дж. Основания языков программирования. М.— Ижевск, РХД, 2009 (Mitchell J. C. Foundationfor programming languages, MIT, 1996.)
6. Непейвода Н. Н. Реверсивные конструктивные логики. Логические исследования, **15**, 150–169 (2008).
7. Axelsen H. G., Glück R. What do reversible programs compute? FOCSSACS 2011, LNCS 6604, pp. 42–56, 2011
8. Nepejvoda N. N. Reversivity, reversibility and retractability. Third international Valentin Turchin workshop on metacomputation. Pereslavl: УГП, 2012. pp 203–215.
9. Мальцев А. И., 'Уч. зап. Ивановского гос. пед. ин-та', 1953, т. 4, с. 107 – 118;