УДК 519.681.2:512.579 + 004.942

АЛГЕБРАИЧЕСКИЙ ПОДХОД К УПРАВЛЕНИЮ

Н.Н. Непейвода

Рассмотрены алгебраические структуры, порождаемые программами моделирования. Намечены их связи с численным моделированием, информатикой, суперкомпьютерами, качественной теорией систем. Сформулированы задачи дальнейших исследований.

Ключевые слова: моделирование, алгебраические системы, обратимые вычисления, группоид.

ВВЕДЕНИЕ

При математическом моделировании сначала строится модель рассматриваемой системы на языке математики, а затем алгоритм и программа, реализующие ее. Полученные результаты анализируются и интерпретируются в рамках как математической модели, так и исходной системы. Традиционно под математической моделью понимается численная модель (дифференциальное уравнение или дифференциально-разностный полудискретный процесс). Если удается, то система обрабатывается точными методами (например, методами качественной теории дифференциальных уравнений). Но чаще всего алгоритмическая модель является приближением к математической (дискретизация непрерывной системы), а программа — приближением к алгоритму. В итоге численные результаты требуют серьезной неформальной обработки, чтобы получить содержательно требуемые качественные выводы о поведении системы.

Дальнейшее развитие средств моделирования путем грубой силы (увеличения количества и быстродействия процессоров и памяти), который был магистральным направлением в последние 40 лет, подошло к трем барьерам. Мощность компьютеров объективно ограничивается скоростью света и тепловыделением, сложность программ — пределом Чейтина. Пока что не видно никаких средств попытаться обойти предел, который ставит скорость света. А вот два других предела можно попытаться преодолеть. Но чтобы это сделать, надо сначала разобраться в их природе.

Алгебраический подход направлен на преодоление двух барьеров: теплового и сложностного. Но развитие в данном направлении потребовало, прежде всего, пересмотра фундаментальных концепций алгебраической теории моделирования.

Алгебры программных систем начали развиваться с 1960-х гг.: Глушков [1], дальнейшее развитие этого подхода см. в работах [2, 3]; Маурер [4], далее в работах [5, 6]; алгебры функциональных программ — прекрасное и полное изложение важнейших результатов дано в работе [7]. Все эти подходы носят на себе печать одного и того же «первородного греха»: в алгебры в том или ином виде непосредственно закладываются операторы языков программирования. Тем самым вместо того, чтобы на базе теории выдать рекомендации для практики, может быть, в новом и неожиданном направлении, ставится задача изучения существующей практики.

В настоящей работе намечается направление, ортогональное традиционному: алгебраические модели на базе теории алгебраических систем [8], и уже затем интерпретация результатов в рамках программных конструкций. Заодно рассматриваются взаимосвязи этого подхода с качественной теорией систем и возможности, которые открываются для преодоления второго и третьего пределов.

1. ТЕПЛО И ЭНЕРГИЯ: ПРЕДЕЛ ЛАНДАУЭРА

Тепловой предел связан с фундаментальным термодинамическом результатом фон Неймана и Ландауэра. В соответствии с принципом Ландау-

эра минимальное выделение тепла на выработку одного бита информации

$$E_{diss} = k_B T \ln 2, \text{ Дж}, \quad (1)$$

где k_B — постоянная Больцмана, T — абсолютная температура в кельвинах [9, 10].

Доказательство принципа Ландауэра не зависит от физической структуры вычислителя. Также не влияют на общую потерю энергии от нереверсивности и сверхнизкие температуры: значение предела Ландауэра (1) формально снижается, но на охлаждение требуется тот же порядок энергии [11]. Эта формула показывает предел производительности традиционных процессоров, т. е. фундаментальный предел производительности идеального классического вычислителя — $2{,}5 \times 10^{26}$ операций/с [12, 13]. А вычислителя, использующего операции с плавающей точкой (принципиально неточные) — 10^{22} операций/с (там же).

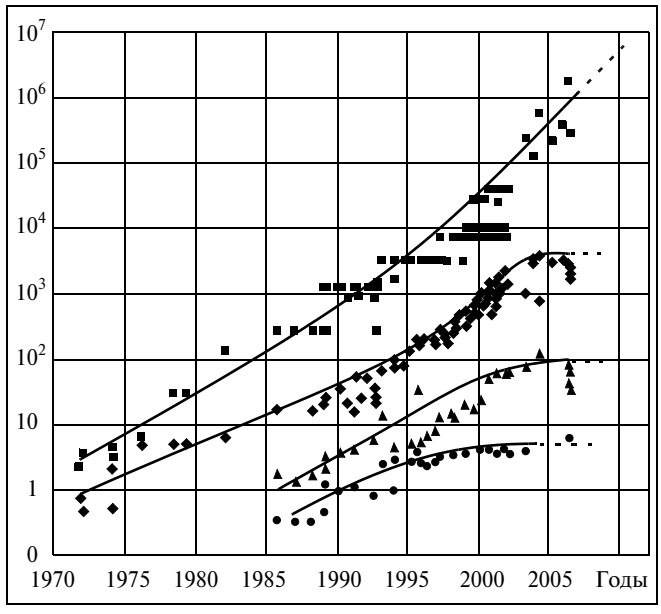
Соображения надежности вычислений указывают на еще более низкие верхние границы [12, 14—16]. Разберем это подробнее. Вероятность ошибки при мощности сигнала E_{sig} равна $e^{-E_{sig}/E_{diss}}$. Для случая электрического сигнала $E_{sig} = CV^2/2$, где C — емкость элемента, V — разность напряжений при переключении. Для надежности нужно $E_{diss} = 100 k_B T$. Этот предел уже достигнут.

Ситуацию иллюстрирует диаграмма (см. рисунок), любезно предоставленная Л.К. Эйсымонтом.

Верхняя кривая на рисунке — рост числа процессоров, вторая сверху — тактовая частота, МГц, третья — выделяемое тепло, Дж, нижняя — производительность одного процессора, операций/с. Видно, что рост идет лишь благодаря грубой силе и технологическим ухищрениям, позволяющим размещать больше процессоров на ограниченном объеме и отводить генерируемое ими тепло.

В работе Ландауэра [10] показан и теоретический метод обхода термодинамического барьера: вычисления должны быть обратимыми, тогда значительная часть выделенного тепла поглощается на месте. Беннет [17] показал, что любая машина Тьюринга может быть смоделирована обратимой машиной Тьюринга, и отметил еще одно важное свойство вычислений, которые могут преодолевать предел Ландауэра: обратимы должны быть не только действия, но и программа.

Чтобы пояснить, что такое обратимость, воспользуемся наиболее распространенной моделью действий. Система представляется как множество состояний, а действия — как функции на этом множестве. Эта модель не является полностью адекватной в нашем случае и ни в коем случае не



Диаграмма, иллюстрирующая положение дел на переднем крае суперкомпьютеров

может рассматриваться как основная, но для демонстрации некоторых примеров на привычном языке она годится. Тогда из анализа доказательств фон Неймана и Ландауэра выявляются следующие свойства обратимости.

1. Никакие два состояния не должны переходить в одно и то же. Для дискретных систем с конечным числом состояний это означает биективность отображений.

2. Имеется возможность после получения результата запустить систему таким образом, что она пройдет в обратном направлении путь от результата к исходным данным через те же промежуточные состояния. Это в совокупности со свойством 1 влечет биективность отображений и для бесконечных систем.

3. Обратное вычисление может быть выполнено до прямого, и в результате их композиции получается тождественное преобразование. На самом деле это следствие свойств 1 и 2, но слишком часто не замечаемое на практике.

Беннет не обратил внимания на то, что выделение памяти и ее освобождение — существенно необратимые действия, а его последователи увлеклись позитивной стороной результатов Беннета. И совершенно было проигнорировано предупреждение Беннета насчет программы. Оно показывает, почему стандартная модель действий неадекватна в данном случае: одним из действий должно быть обращение последовательности действий.

Не только из соображений тепловыделения возникает обратимость. Общеизвестно, что квантовые вычисления по своей природе обратимы. Поэтому, если когда-нибудь удастся создать квантовый процессор, он будет способен лишь на обратимые вычисления. Далее, вычисления на молекулярном уровне тоже обратимы по своей природе. Вычисления на уровне нанокристаллов (все равно, электрические или световые) тоже часто обратимы. Нанокристалл вычисляет конечную группу. Группы вычислений кристаллов — более широкий класс, чем шубниковские группы [18—21], поскольку групповыми являются также вычисления в кристаллах с упорядоченными дислокациями. Вычисления на сверхпроводниках тоже должны быть обратимыми, если мы не желаем вскипятить жидкий азот.

Стандартным представлением обратимых вычислений в математике являются группы. Поэтому обратимость требует алгебраических, а не численных, моделей.

2. ПАРАДОКС ИЗОБРЕТАТЕЛЯ: ЭФФЕКТ ОРЕВКОВА И ПРЕДЕЛ ЧЕЙТИНА

Рассмотрим теперь вторую трудность: накопление громадных количеств кода, зачастую плохо сделанного. Такой код становится необозримым и неулучшаемым. Попытки улучшить перешедшую некоторый барьер программу приводят к тому, что на каждую исправленную ошибку появляется две новых, что даже рассматривается как некий «объективный закон». Подкрепляет такие убеждения то, что даже в сообществе свободного софта, где каждая программа, а особенно системные программы, подвергается беспощадному анализу на понятность кода, в последние годы начался кризис. Старые программы становятся все менее перестраиваемыми, а приток молодых и энергичных волонтеров, на которых 20 лет развивалось сообщество, резко сократился в связи с почти запредельной сложностью вхождения в существующие проекты [22]. Тем самым надежность существующего программного обеспечения оказалась практически уникально низкой среди всех современных высокотехнологичных систем. Каждая программа с гарантией содержит ошибки.

Это неразрывно связано еще с одним, более «мягким», но тоже объективным барьером: пределом Чейтина [23]. В этой статье установлено существование принципиально непознаваемых объектов, причем предел непознаваемости конечен и зависит от «сложности» познающего субъекта.

Рассмотрим понятие алгоритмической сложности объекта[1]. Это *минимальная длина программы и данных, порождающих искомый объект*. Объект может быть очень большим (например, двоичная запись числа $33^{33^{33}}$), но воспроизводится короткой программой и данными, и поэтому прост.

Понятие алгоритмической сложности, первоначально введенное Колмогоровым [24] для стандартной концепции вычислимости, оказывается корректным для очень широкого класса концепций и уже, чем тьюрингова вычислимость, и шире ее [25]. Теорема Чейтина [23], как показано в работе [25], сохраняется для самых различных понятий сложности и при очень слабых предположениях о формальной системе. Поэтому ее можно сформулировать в качестве методологического закона:

«Для каждого субъекта или сообщества объекты, превосходящие некоторую сложность, являются принципиально непознаваемыми. Предел непознаваемости для каждого индивидуума свой, и область познаваемости может быть неравномерно развита в разных областях».

Таким образом, если системы становятся все сложнее, с некоторого момента они оказываются принципиально непонимаемыми. Длина и структурная сложность программ оказываются сейчас во многих случаях решающим фактором.

Возникает вопрос, является ли такая ситуация объективным законом развития больших формализованных систем? Сразу же виден контрпример: математика. В ней результаты, первоначально получаемые как сложные конструкции на грани понимания, в дальнейшем постепенно свертываются во все более и более короткие, и «разбухания кода» не происходит. Например, теорема полноты классической логики первоначально доказывалась на 200 страницах, а теперь доказательство умещается на двух, одна из которых занята необходимыми определениями. Проанализируем, благодаря чему это достигается.

Еще в 1920-е гг. Д. Гильберт [26] уверенно утверждал, что развитие математики практически невозможно без идеальных высокоуровневых по-

[1] Обратим внимание, что слово «сложность» употребляется в различных контекстах по отношению к совершенно разным характеристикам. Вычислительная сложность принимает во внимание количество операций для решения задачи, но игнорирует характеристики самой программы. Структурная сложность измеряет характеристики структуры объекта, почти игнорируя его величину. Логическая сложность измеряет уровень объекта или высказывания. Алгоритмическая в некотором смысле сочетает характеристики структурной сложности и сложности, измеряемой просто как количество знаков в коде объекта.

нятий. В конце концов на практике нас интересуют реальные ответы, выражающиеся конечными сущностями. Например, численная оценка или область фазового пространства, в которой находится система; последнее конечно ввиду того, что область представляется как элемент дискретного разбиения пространства или же как элемент конечной алгебраической структуры, качественно описывающей подмногообразия пространства состояний системы. Но получить нетривиальные реальные результаты с приемлемыми затратами труда получается, лишь пройдя через идеальные абстрактные понятия и их теоретические преобразования, которые можно рассматривать как интеллектуальные орудия.

Д. Пойа [27] заметил этот эффект на примерах из элементарной математики: чтобы доказать простое утверждение, часто требуются значительно более сложные леммы. Хао Ван [28] подтвердил этот факт для арифметики, построив последовательность примитивно-рекурсивных функций f_n, для каждой из которых доказательство утверждения $\forall x\, f_n(x) = 0$ может быть проведено лишь с помощью математической индукции по формулам не менее чем с n переменами кванторов. В.П. Оревков доказал [29], что аналогичный эффект имеет место уже в логике, где, *в принципе*, от лемм можно избавиться (теорема о нормализации вывода). А именно, он построил последовательность формул, доказательство каждой из которых с промежуточными леммами имеет длину $13n + 7$, а без помощи лемм (прямое) — как минимум $2^{(2^{(\ldots 2)})}$ (n раз). Идею конструкции Оревкова можно пояснить на двух функциональных примерах: числовом и абстрактном (в λ-исчислении).

Пример 1. Пусть у нас есть натуральные числа и одна лишь операция добавления 1. Рассмотрим рекурсивное определение

$$\varphi(x, 0) = x + 1;$$

$$\varphi(x, y + 1) = \varphi(\varphi(x, y), y).$$

Согласно ему, экспонента вычисляется в линейное число шагов.

Если теперь задать определение функции второго порядка

$$\Phi_1(\varphi, x, 0) = \varphi(0, x);$$

$$\Phi_1(\varphi, x, y + 1) = \Phi_1(\varphi, \Phi_1(\varphi, x, y), y),$$

то оно вычисляет уже экспоненту экспонент, третий уровень башни дает

$$\Phi_2(\varphi, x, 0) = \Phi_1(0, x);$$

$$\Phi_2(\Phi_1, x, y + 1) = \Phi_2(\Phi_1, \Phi_2(\Phi_1, x, y), y)$$

и т. д.

Пример 2. В λ-исчислении, которое сейчас стало главным теоретическим аппаратом информатики, есть две базовых конструкции: применение функции f к аргументу ($x\,f$) и образование функции, вычисляющей при каждом заданном x_0 значение выражения t, содержащего переменную x: $\lambda x.t[x]$. Функции и объекты не различаются: функция может быть аргументом и результатом другой функции (почти как в теории множеств, где «всё есть множество»; только здесь «всё есть функция»). Тогда можно построить последовательность определений, уже не рекурсивных:

$$\Phi_1 = \lambda f.\lambda x.((x\,f)f);$$

$$\Phi_{n+1} = \lambda\Psi_n.\lambda\Psi_{n-1}.((\Psi_{n-1}\Psi_n)\Psi_n);$$

$$(x(f(\Phi_1\ldots(\Phi_{n-2}(\Phi_{n-1}\Phi_n^k))\ldots))) = \left(xf^{\left(2^{\left(2^{(\ldots 2)}\right)}\right)}\right)\ (k \text{ раз}).\ \blacklozenge$$

В дальнейшем для определения функций мы свободно пользуемся λ-обозначениями.

Заметим, что в обоих определениях использовались функции над функциями. Это неизбежно, поскольку теоремы Хао Вана, Оревкова и их многочисленные применения и усиления, развившиеся сейчас в целое направление «обратной математики», используют иерархию сложности и уровней объектов. Уровень функции или множества выше уровней аргументов (элементов). Уровень логической формулы определяется в первую очередь уровнями представленных в ней объектов, по которым навешиваются кванторы, а во вторую очередь количеством перемен вложенных друг в друга кванторов.

Выразимость существенно повышается при подъеме на новый уровень. Но одновременно понятия становятся более абстрактными.

Таким образом, сколь угодно сложные рекурсивные процедуры стандартного языка программирования типа C — объекты первого уровня. Классы C или Java — где-то между первым и вторым. А для существенного сокращения текстов и борьбы с объемом программ нужны понятия высших уровней.

Явное введение сложных понятий необходимо, но нужно искать обходные пути, поскольку выразительная сила растет, как минимум, суперэкспоненциально, однако одновременно экспоненциально убывает число людей, способных воспринять понятия соответствующего уровня[2].

Здесь вступает в силу фундаментальное свойство алгебраических систем. Рассмотрим произ-

[2] Автор может пересчитать по пальцам тех, кто способен понять конструкции седьмого уровня, порою встречающиеся в информатике и в качественной теории систем (работы школы В.М. Матросова и С.Н. Васильева).

вольный группоид (тип данных, на котором задана бинарная операция $*$; никакие свойства этой операции, даже ассоциативность, не постулируются). Тогда любой элемент a может одновременно рассматриваться как функция $\lambda x.(x * a)$, называемая *действием* элемента a. Работая с неассоциативными операциями, мы никогда не будем опускать скобки. Продолжая далее, видим, что элемент одновременно является преобразованием функций, и дальше вверх по иерархии уровней.

Таким образом, алгебраические модели дают возможность атаковать и второй барьер.

Пример 3. В частном случае возможность представления функций высших уровней при помощи элементарной алгебраической системы была открыта М. Шейнфинкелем [30] в 1920 г. Но общность этого метода не была осознана.

Комбинаторная алгебра Шейнфинкеля — группоид с операцией $*$ и двумя константами: **S** и **K.** Постулируются следующие свойства этих констант:

$$(x * (y * (z * S))) = ((y * z) * (x * z));$$

$$(x * (y * K)) = y. \blacklozenge \qquad (2)$$

Комбинаторные алгебры обладают следующими свойствами. Как известно, любая алгебраическая система, определяемая тождествами, имеет тривиальную модель из одного элемента. Все нетривиальные модели комбинаторных алгебр [31]:

— бесконечны;

— неассоциативны;

— некоммутативны;

— нерекурсивны;

— позволяют выразить внутри себя натуральные числа и любую вычислимую функцию как действие некоторого элемента на натуральные числа;

— позволяют выразить вычислимые функции над функциями и сколь угодно высоко вверх по иерархии понятий.

Таким образом, алгебраические модели в принципе позволяют атаковать и барьер сложности программ. Это косвенным образом подтверждено в работах по категорному программированию [32].

3. ПРОГРАММНЫЕ АЛГЕБРЫ

В работах по алгебрам программ (в качестве представительных примеров см. работы [1—6]) немедленно предполагается, что программа — это функция, и, соответственно, базовая алгебраическая структура этих моделей — полугруппа. Основанием такого выбора служит классическая теорема о представлении любого пространства функций, замкнутого относительно композиции, как полугруппы и любой полугруппы как полугруппы функций. Даже в книге Митчелла [7], как только переходят к алгебраическим моделям, они рассматриваются на базе полугрупп. Есть еще один принципиальный недостаток всех рассматриваемых алгебраических конструкций. Они ставят телегу впереди лошади: сразу моделируются конкретные конструкции языков программирования и включаются как операторы в алгебраические модели. Таким образом, теоретическое исследование оказывается в плену существующей практики, которая уже не полностью адекватна запросам развития информатики, как показано ранее. Не может быть описано как функция, в частности, действие обращения операции, естественно возникающее в обратимом программировании. В самом деле, пусть действием элемента M является обращение операции. Тогда, поскольку $(a \circ b)^{-1} = b^{-1} \circ a^{-1}$, имеем $a \circ (b \circ M) \neq (a \circ b) \circ M$.

На самом деле аналогичная конструкция возникает на каждом шагу в практической информатике. Рассмотрим операцию отмены последнего действия Undo. Тогда

$$\mathrm{S};\ T;\ \mathrm{Undo} \neq \{\mathrm{S};\ T\};\ \mathrm{Undo}.$$

Таким образом, *действия операторов не обязательно являются функциями*.

Но синтаксическая структура программ, которая важна при их преобразованиях, полугруппой является. А именно, полугруппа дает возможность соединить действия нескольких элементов в действие одного элемента, тем самым позволяя агрегировать блоки модели. Эта операция необходима для декомпозиции и синтеза систем моделирования.

Самый нейтральный способ ввести операцию, свертывающую систему элементов в элемент, аналогичен используемому в комбинаторных алгебрах. Добавим к группоиду константу **B** (композиция) и аксиому

$$((x * f) * g) = (x * (f * (g * B))). \qquad (3)$$

Такие группоиды назовем *композиционными алгебрами*.

Предложение 1. *Для каждой последовательности* $f_1, \ldots, f_n$ *элементов группоида ее действие* $\lambda x.((\ldots(x * f_1)\ldots) * f_n)$ *равно действию элемента* $(f_1 * (\ldots(f_n * B)\ldots))$.

Определение 1. *Обогащение* алгебраической системы — добавление новых операций и констант без изменения носителя и старых операций[3]. *Расширение* алгебраической системы — вложение ее в другую алгебраическую систему. *Факт* — равен-

[3] Тем самым значения новых констант задаются в старом носителе.

ство или неравенство постоянных выражений над алгебраической системой: $t = u$ либо $t \neq u$.

Предложение 2. *Каждый группоид* **G** *может быть расширен до композиционной алгебры.*

Доказательство. Рассмотрим свободный группоид, образующийся добавлением к элементам **G** константы **B**, и хорнову теорию, образуемую всеми позитивными фактами о **G** и тождеством (3). Фактор-алгебра по отношению выводимого равенства будет искомой композиционной алгеброй. ♦

Однако полугруппы действий не получается: композиция в композиционных алгебрах не обязательно ассоциативна. Сделать ее ассоциативной можно, задав аксиому

$$(f * ((g * (h * B)) * B)) = \\ = ((f * (g * B)) * (h * B)). \qquad (4)$$

Итак, приходим к алгебре, являющейся бигруппоидом с двумя операциями. Одна из них — полугрупповая операция композиции $\circ$, вторая — неассоциативная операция применения программы $*$. Получающуюся полугруппу по композиции будем обозначать прописной жирной буквой, ее носитель — прописной буквой; 0 и e — нулевой и единичный элемент соответствующей полугруппы, если они имеются. Операции связаны тождествами:

$$((x * f) * g) = (x * (f * g));$$

$$(0 * x) = (x * 0) = 0;$$

$$(x * e) = x.$$

Как бы мы ни определили понятия композиции и применения, они должны быть связаны приведенными 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).

Первое тождество выполняется всегда, оно является определением последовательного исполнения либо агрегации элементов. Если хоть одна из констант 0 или e присутствует, она сохраняет свой смысл и для операции $*$.

Пусть имеется некоторый гомоморфизм α: $G \to (G \to G)$ нашей полугруппы **G** в полугруппу отображений ее носителя, такой, что

$$(e\ \alpha) = \lambda x.x; \quad (0\ a) = \lambda x.0.$$

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 отображение α изоморфизмом нельзя.

Предложение 3. *Каждая полугруппа* **G** *может быть обогащена до GAPS таким образом, что* $(x * f) = (f\ \alpha)$.

Доказательство. Второе и третье тождества выполнены по построению.

$$((x * f) * g) = ((x(f\ \alpha))(g\ \alpha)) = (x((f\ \alpha) \circ (g\ \alpha))) = \\ = (x((f \circ g)\alpha)) = (x * (f \circ g)).$$

Такой гомоморфизм всегда существует. Достаточно определить $(x\ \alpha) = \lambda y.0$ для всех делителей нуля, и $\lambda x.x$ для всех остальных. Второй тривиальный вариант — просто определить $(x * y) = (x \circ y)$. ♦

Пример 4. Рассмотрим известную игру «колодец, ножницы, бумага». Бинарная операция определения выигравшего в этой игре задает коммутативный неассоциативный группоид КНБ с тремя элементами:

*	к	н	б
к	к	к	б
н	к	н	н
б	б	н	б.

Для того чтобы задать GAPS, содержащую этот группоид, имеется несколько способов. Первый из них — чисто синтаксический. Отождествим элементы группоида с их одноэлементными кортежами, а композиции действий зададим как кортежи из к, н, б; $\circ$ — соединение кортежей. Но даже в таком пополнении имеется много способов задания операции $*$. Проиллюстрируем их на примере кортежей [к, н, б] и [н, к, б]:

$$[к, н, б] * [н, к, б] = [к, н, (((б * н) * к) * б)];$$

$$[к, н, б] * [н, к, б] = [(((к * н) * б) * н), к, б];$$

$$[к, н, б] * [н, к, б] = \\ = [(((к * н) * к) * б), (((н * н) * к) * б), (((б* н) * к) * б)].$$

Теоретически простейший способ получения GAPS — построить свободную алгебру над константами {к, н, б} с бинарными операциями {$*$, $\circ$} и факторизовать ее по отношению равенства «$t = u$ выводимо в теории со следующими аксиомами»:

$$(к * к) = к; \quad (к * н) = к; \quad (к * б) = б;$$

$$(н * к) = к; \quad (н * н) = н; \quad (н * б) = н;$$

$$(б * к) = б; \quad (б * б) = б; \quad (б * н) = н;$$

$$\forall x, f, g((x * f) * g) = (x * (f \circ g));$$

$$\forall x, f, g((x \circ f) \circ g) = (x \circ (f \circ g)).$$

Есть способ получить и конечную GAPS. Для этого сначала пополним КНБ до полугруппы действий последовательностей его элементов. Эта полугруппа оказывается некоммутативной. Приведем все различные действия последовательностей элементов игры. Перед двоеточием стоит последовательность ходов, после него — ее действие:

к: ккб, н: кнн, б: бнб, кн: ккн, нк: ккк, бн: ннн,

нб: бнн, кб: ббб, бк: бкб,кнб: ббн, нбк: бкк, бкн: нкн.

Полугруппа действий состоит из двенадцати элементов. Но даже здесь операция $*$ может быть продолжена, по крайней мере, двумя способами:

$$(\text{кн} * \text{нк}) = \text{кннк} = \text{кнк} = \text{кн};$$

$$(\text{кн} * \text{нк}) = (\text{кн})*(\text{н}*\text{к}) = (\text{кн})\ \text{к} = (\text{к}*\text{к})(\text{н}*\text{к}) = \text{кк} = \text{к}.$$

Пример 5. Некоторые «естественные» предположения о GAPS разрушают систему. Например, из тождества $e * f = f$ следует, что обе операции совпадают:

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

Рассмотрим возникшую ситуацию содержательно: почему 0 может оставаться нулем, а единица при действиях оставаться сама собой не всегда может? Нуль в смысле интерпретации — ошибка. Дальше с нею внутри алгебры делать уже нечего: система вылетела. Константа 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. См. далее пример 8. ♦

Центральная проблема: когда можно некоторую систему преобразований программ соединить с данной синтаксической полугруппой программ, тем самым образовав единый язык метапрограммирования? Эта задача естественно разбивается на три подпроблемы: соединение языка с системой преобразований без изменения языка (т. е. полугруппы); сохранение некоторого подъязыка (подполугруппы) и превращение ранее не интерпретированных или не нужных нам более синтаксических конструкций в действия преобразователей; пополнение существующего языка до метаязыка путем расширения полугруппы без изменения исходной.

Определение 2. Полугруппа **F**, порожденная совокупностью функций F — минимальный тип функций, включающий F и замкнутый относительно композиций:

$$\forall f, g(f \in F \wedge g \in F \supset f \circ g \in F). \blacklozenge$$

Содержательно это означает, что **F** состоит из всех функций, представимых как

$$f_1 \circ \ldots \circ f_n, f_1, \ldots, f_n \in F.$$

Определение 3. Пусть задана некоторая совокупность действий **A** как функции на носителе полугруппы ***G***. Совокупность **A** *допустима*, если имеется GAPS **AG**, которая является обогащением **G**, такая, что каждое действие $g \in A$ представляется как $\lambda x.(x * \alpha)$ для некоторого α. Она *консервативна над подполугруппой* G_0, если она допустима и $a * b = a \circ b$ для всех элементов G_0. Она *слабо допустима*, если имеется полугруппа G_1, подполугруппой которой является **G**, и функции из **A** могут быть продолжены на G_1 таким образом, что полученная A_1 консервативна над **G**. ♦

Очевидно, что самое сильное из этих понятий — консервативность над подполугруппой. Далее идут допустимость и слабая допустимость. Установим критерии возможности этих обогащений и расширений.

Предложение 4. *Совокупность действий* **A** *допустима тогда и только тогда, когда имеется гомоморфизм ее замыкания в полугруппу* **G**.

Доказательство. По предложению 2, если обогащение удалось осуществить, то любой элемент полугруппы, порожденной **A**, представляется как действие некоторого элемента G. Значит, мы гомоморфно вложили замыкание **A** в **G**. ♦

Обратно, если замыкание **A** гомоморфно отображается в **G**, то его образ изоморфен некоторой фактор-полугруппе $\hat{G}$ полугруппы **G** и имеется отображение $\Phi: G \to (G \to G)$, сопоставляющее каждому элементу полугруппы функцию, соответствующую его классу эквивалентности в $\hat{G}$.

Заметим, что важно вложение структуры полугруппы действий, а не самих действий.

Предложение 5. *Совокупность действий* **A** *консервативна над* G_0 *тогда и только тогда, когда имеется такой гомоморфизм* ψ *ее замыкания в* **G**, *что для каждого действия f, такого, что* $(f\psi) \in G_0$, *выполнено* $\forall a \in G(a f) = (a \circ (f \psi))$.

Доказательство. Если такое обогащение удалось осуществить, то действие, попавшее в G_0, обязано по условию сохранения подполугруппы удовлетворять

$$\forall a \in G(a f) = (a \circ (f \psi)). \blacklozenge$$

Обратно, если такой гомоморфизм имеется, то применим построение из предыдущего доказательства.

Пример 5. Например, пользуясь этим критерием, можно проверить возможность обогащения языка до языка метавычислений, рассматривая строки как программы, а подъязык, работающий с числами, оставляя неизменным.

Теорема 1. *Любая совокупность действий слабо допустима над любой* **G**. ♦

Эта теорема доказана в статье [33]. Несмотря на простоту формулировки, доказательство требует методов теории моделей и теории алгебраических систем. Получающаяся алгебра не всегда вычислима, даже если исходная полугруппа и совокупность действий конечны.

Пример 6. Рассмотрим группу вычетов по модулю 3 Z_3 и интерпретируем ее объекты {0, 1, 2} как коды колодца, ножниц и бумаги в игре из примера 4. Кодируя

функции тройками чисел, задаем действия игры как 002, 011, 212. Пополним группоид до двенадцатиэлементной полугруппы. Для согласованности с Z_3 необходимо добавить единичный элемент, который не может быть получен композициями остальных. Возьмем прямое произведение получившегося тринадцатиэлементного моноида C с Z_3 и зададим действия следующим образом:

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

Построенная алгебра содержит Z_3. Видно, что элементы C можно рассматривать как команды, а Z_3 — как данные; (xd) — применение последовательности действий к элементу, закодированному x,и взятие кода получившегося элемента. Команды преобразуются как полугруппа, но их эффект на Z_3 — группоид.

Имеется и другой способ определения GAPS над тем же носителем:

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

Если же в качестве пополнения КНБ возьмем бесконечную GAPS, то и получившаяся система окажется бесконечной. ♦

Дадим полный теоретический критерий пополнимости некоторой системы операций над программами с заданными свойствами операций до GAPS над уже имеющейся у нас программной системой с сохранением свойств операций и программ. Он обобщает теорему из статьи [33], где сохранялись только свойства операций и факты о программах.

Определение 4 [8]. *Факт* — выражение вида $P(t_1, ..., t_n)$ либо $\neg P(t_1, ..., t_n)$, не содержащее переменных, в частности, $t = u$ или $t \neq u$. Факт *позитивный*, если он не начинается с отрицания. *Диаграмма системы* — совокупность всех истинных в ней фактов, при условии, что сигнатура пополнена константами для всех элементов носителя системы. *Тождество* — формула вида $\forall x_1, ..., x_k P(t_1, ..., t_n)$, где x_i — все свободные переменные элементарной формулы. Ограничение формулы A на одноместный предикат P — формула A_P, образующаяся заменами всех кванторов $\forall x B[x]$ на $\forall x(P(x) \supset B[x])$, а всех $\exists x B[x]$ на $\exists x(P(x) \wedge B[x])$. Если Σ — алгебраическая система, то Th^{Σ} — теория Th, пополненная всеми фактами о Σ.

Теорема 2. *Пусть система действий* **A** *описывается теорией* Th_1, *и* Th *— теория, моделью которой является полугруппа* **G**, *в которой выводится аксиома ассоциативности, а также свойства единицы и нуля, если они присутствуют в* **G**, *и одноместный предикат* P *не принадлежит объединению сигнатур.* Th_P *означает теорию* Th, *к сигнатуре которой добавлен предикат* P *без новых аксиом. Тогда* **G** *можно пополнить действиями* **A** *с сохранением их свойств тогда и только тогда, когда теория*

$$Th_1^A \cup Th_P^G \cup \{\forall x, f, g(((x * f) * g) = (x * (f \circ g))), \forall x(0 * x) = (x * 0) = 0, x(x * e) = x\}$$

непротиворечива.

Доказательство. По теореме полноты классической логики, теория имеет модель тогда и только тогда, когда она непротиворечива. Если теория содержит матрицу алгебраической системы Σ, эта система изоморфно вложима в любую ее модель [8, *с. 172*]. Тем самым существующая модель является искомой GAPS. Обратно, если есть такая GAPS, то она является моделью нашей теории. ♦

Теоретически данное решение полное. Но на практике оно редко применимо, поскольку чаще всего расширения оказываются невычислимы. Однако есть один практический критерий, который следует из данной теоремы.

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

Пример 7. Поскольку λ-исчисление лежит в основе современной математический теории полных по Тьюрингу программных систем, построим модель стратифицированного λ-исчисления как GAPS. Напомним некоторые понятия: λ-выражение называется *стратифицированным*, если в нем можно корректно расставить типы всех переменных и констант; λ-выражение имеет нормальную форму, если оно не может быть преобразовано далее. Выражения в нормальной форме считаются окончательными результатами вычислений. Обе возможности: рассматривать полугруппу как множество λ-термов в нормальной форме или как множество отображений нормальных форм t в нормальные формы $(f\, t)$, происходящих при конверсии выражений, не дают решений. При первом подходе нет композиции, поскольку нет операции соединения λ-выражений. При втором композиция естественно определяется, но теряется суть программной системы, поскольку разные программы могут производить одну и ту же функцию преобразования.

Чтобы получить GAPS, перейдем к эквивалентному представлению λ-термов: комбинаторной логике. Комбинаторная логика естественно и взаимнооднозначно формулируется как группоид действий: применение терма к терму $(t\, r)$ записывается в наших обозначениях $(r * t)$.

Определяющие равенства для констант **K** и **S** переписываются в форме (2). В комбинаторной алгебре определяется комбинатор композиции

$$\mathbf{B} = (\mathbf{K} * ((\mathbf{S} * \mathbf{K}) * \mathbf{S})).$$

Преобразуя это выражение согласно определяющим равенствам для **K** и **S**, получаем:

$$(x * (f * (g * (\mathbf{K} * ((\mathbf{S} * \mathbf{K}) * \mathbf{S})))) =$$
$$= (x * (f * ((g * \mathbf{K}) * (f * (\mathbf{S} * \mathbf{K}))))) =$$
$$= (x * (f * ((g * \mathbf{K}) * \mathbf{S}))) = ((x * f) * (x * (g * \mathbf{K}))) =$$
$$= ((x * f) * g).$$

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

Можно было перейти к другому известному базису комбинаторной логики, включающему в себя **B**, но мы проиллюстрируем путь, показывающий способ перехода от теории или системы, в которой композиции как корректно определенной операции не было, к эквивалентной теории или системе, представимой как алгебра GAPS.

Прежде всего, добавим **B** как новую константу с определяющим равенством (3). Такое расширение теории с привычной точки зрения эквивалентно исходной теории, поскольку добавили определимое понятие, но оно избавляет от нежелательного эффекта: как бы мы ни определили **B**, $(f * (g * \mathbf{B}))$ не будет нормальной формой, поскольку последняя константа в определении заведомо применится (даже если она **S**, требующая три аргумента, то, поскольку определение не сводится к самой **S**, третий аргумент будет взят из текста определения). Добавляем равенство (4). Это тождество может быть эффективно реализовано как одностороннее и заведомо фундированное правило переписывания левой части в правую. Носитель алгебры — множество нормальных форм выражений в комбинаторной логике, пополненной комбинатором **B**, нормализованных применением правила, соответствующего равенству (4). Композицией двух выражений считается $(f * (g * \mathbf{B}))$. Это определение корректно, поскольку стратифицированные λ-выражения всегда имеют нормальную форму, и композиция стратифицированных выражений стратифицирована.

Построение завершено. Таким образом, GAPS позволяют описать одну из самых общих теоретических концепций современной информатики. Для бестипового λ-исчисления эта конструкция также работает. Значением выражения, не имеющего нормальной формы, считается 0. В стратифицированной алгебре нуля нет. В построенных GAPS нет единицы. Она может быть добавлена как новая константа с правилами конверсии

$$(t * e) = t,\ (f * (e * \mathbf{B})) = f,\ (e * (f * \mathbf{B})) = f.$$

Итак, представление «конкретных» алгоритмических языков становится излишним: их методы сведения к λ-исчислению детально проработаны и изложены, в частности, в работе [7].

Пример 8. Рассмотрим другой 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. Соединение строк дает композицию функций программ. Пустая строка — ничего не делающая программа. Примером такого языка может служить Brainfuck [34].

Добавим константы 0 и e. Первая из них является значением вылетающей или зацикливающейся программы, вторая — пустая строка. Композиция программ определяется как соединение строк. Действие строки $(a * f)$ состоит в вычислении строки, являющейся значением f на a. Если f выдала ошибку, значением является 0. В противном случае — получившаяся программа. На этом примере видно, что результатом действия над пустой программой может быть любая другая программа.

Пример 9. Еще в 1972 г. теория вплотную подошла к общему понятию алгебры программ [35]. Рассмотрим алфавит {K, S, (,)}. Определим следующую трансляцию его символов в комбинаторы. Выражение, сопоставляемое строке, определяется по рекурсии (a — символ, σ — строка):

$$(\mathrm{K}\varphi) = \mathbf{K};\quad (\mathrm{S}\ \varphi) = \mathbf{S};\quad (\text{`('}\ \varphi) = \mathbf{B};\ ((\text{`)'}\ \varphi) = \mathbf{I};$$
$$(a\sigma\ \varphi) = ((\sigma\ \varphi) * (a\ \varphi)).$$

Для традиционной комбинаторной логики, где функцией считается первый элемент списка и скобки группируются влево, получим еще более простую запись: образы символов пишутся один за другим. Такая интерпретация превращает комбинаторную логику в язык типа Brainfuck. Но получившаяся алгебра не является алгеброй для комбинаторной логики. В нее входят образы синтаксически неправильных конструкций, скажем,))))SK((.

4. АЛГЕБРЫ ОБРАТИМЫХ ВЫЧИСЛЕНИЙ

Обратимые действия составляют группу, и любая группа представляется как группа биективных отображений. Группа сохраняет основное свойство группоида: возможность рассматривать любое данное как функцию. В ней имеется дополнительная операция взятия обратного действия (которая может применяться как до, так и после исходного действия): a^{-1}. Таким образом, кажется, что в данном случае GAPS основана на группе.

Преобразования программ также должны быть обратимы, согласно принципу Беннета. Отсюда вытекает конструкция алгебр полностью обратимых программ (AFIP — algebras of fully invertible programs) как GAPS с дополнительной константой M (обращение программ) и дополнительными аксиомами:

$$(x \circ (x * M)) = e,\ ((x * M) \circ x) = e. \qquad (5)$$

Отсюда следует, что $(M * M) = M$; $(M \circ M) = e$.

Для практических нужд обычно достаточно более слабого свойства: тождественности действий (слабые алгебры обратимых программ, weak AFIP, WAFIP):

$$((y * x) * (x * M)) = y,\ ((y * (x * M)) * x) = y. \quad (6)$$

Выражение (6) следует из выражения (5), но не наоборот. В статье [33] изучались AFIP. Здесь ос-

новным понятием являются WAFIP, которых достаточно для представления обратимых программ и их аппаратных моделей. Из аксиом WAFIP автоматически следует биективность операции применения программ $*$:

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

Доказательство. Инъективность. Пусть $(x * f) = (y * f)$. Тогда $((x * f) * (f * M)) = ((y * f) * (f * M))$, и $x = y$ по выражению (6).

Сюръективность. Найдем для каждого x и f такое y, что $(y * f) = x$: $x = ((x * (f * M)) * f)$. ♦

В WAFIP полугруппа не обязательно становится группой, так как биективность каждого действия не означает различия всех действий как функций.

Пример 10. Рассмотрим обогащение Z_6 до AFIP:

$$\lambda x.(x * 3) = \lambda x.(x * 1) = \lambda x.(x * 5) = \lambda x.x^{-1},$$

$$\lambda x.(x * 0) = \lambda x.(x * 2) = \lambda x.(x * 4) = \lambda x.x.$$

В качестве M нужно взять 3. Если мы возьмем другое нечетное число, то получится WAFIP.

Аналогичным образом можно пополнить до WAFIP группу целых чисел по сложению:

$$\lambda x.(x * (2 \cdot n + 1)) = \lambda x.x^{-1},\ \lambda x.(x * (2 \cdot n)) = \lambda x.x.$$

Но здесь до AFIP не продолжишь, поскольку нет такого числа, не равного 0, что $n + n = 0$. ♦

Теперь рассмотрим циклический моноид, порождаемый a и определяемый равенством $a^4 = a^8$. На его носителе всем нечетным степеням a^{2n+1} сопоставим $\lambda a^n.a^{-n}$, где обратный элемент берется по модулю 8, четным — тождественное преобразование. Но данная алгебра не является WAFIP, потому что степени, большие 3, уже не могут быть обращены до степеней, меньших 3.

И завершим тривиальным примером. Любая полугруппа правых единиц, т. е. такая, что $\forall x, y \circ y = x$, может быть сделана WAFIP, а в качестве M можно взять любой ее элемент.

В последней WAFIP полугруппа группой не является. Но, тем не менее, получен следующий результат:

Предложение 7. *Действия элементов* WAFIP *образуют группу.* ♦

Действие $(x * M)$ является обратным к действию x.

В статье [33] доказано, что групповая операция может образовывать AFIP в том и только в том случае, когда все элементы группы имеют порядок 2, т. е. группа является прямым произведением групп Z_2. Следовательно, если операция в полугруппе образует WAFIP, то действия ее элементов образуют группу порядка 2.

Предложение 8. *Каждая полугруппа* **G**, *разлагающаяся в прямое произведение* $Z_2 \times G_0$, *где действия элементов* G_0 *взаимно однозначные и для каждого действия есть элемент, производящий обратное действие, может быть обогащена до WAFIP.*

Доказательство. Сначала установим несколько более абстрактное утверждение о полугруппах. Пусть **G** — носитель полугруппы. Зададим для M в качестве значения какой-нибудь элемент, такой, что действие M^2 тождественное и M не является квадратом никакого элемента. Пусть **GG** — группа всех взаимно однозначных отображений G. Пусть φ — гомоморфизм **G** в **GG**, такой, что $(M\,\varphi) = \lambda x.x^{-1}$. Определив $(x * f) = (x(f\varphi))$, получаем искомую алгебру. ♦

Теперь построим такой гомоморфизм: $\langle 1, e\rangle = M$, $(M\,\varphi\rangle = \lambda x.x^{-1}$, $(\langle 0, a\rangle\varphi) = \lambda x.x$, $(\langle 1, a\rangle\varphi) = \lambda x.x^{-1}$.

Необходимое условие обогащаемости полугруппы взаимно однозначных отображений до WAFIP заключается в наличии элемента, который:

— не является квадратом никакого другого элемента;

— имеет бесконечный порядок или является корнем нечетной степени из самого себя.

Установление необходимых и достаточных условий обогащаемости группы либо полугруппы до WAFIP, подобно тому, как было сделано для AFIP в статье [33], остается открытой проблемой.

Пример 11. Рассмотрим общую и важную для приложений конструкцию. Пусть есть произвольная полугруппа **G**, действия элементов которой обратимы, и для каждого действия есть обратное. Пополним ее до алгебры $Z_2 \times G$ с сохранением G.

Положим $\langle x, a\rangle * \langle 0, b\rangle = \langle x, a \circ b\rangle$, $\langle x, a\rangle * \langle 1, e\rangle = \langle x, a^{-1}\rangle$, где e — произвольный элемент, действие которого тождественно. Продолжим на остальные элементы следующим образом:

$$(\langle x, a\rangle * \langle 1, b\rangle) = (\langle x, a\rangle * \langle 1, e\rangle \circ \langle 0, b\rangle)) =$$
$$= ((\langle x, a\rangle * \langle 1, e\rangle * \langle 0, b\rangle) = \langle x, a^{-1}\rangle * \langle 0, b\rangle = \langle x, a^{-1} \circ b\rangle.$$

Полученное решение может быть интерпретировано как простейшее управление. Элемент Z_2 является управляющим сигналом, изменяющим направление работы системы с прямого на обратное и наоборот.

Заметим, что не всегда обратимость действий означает обратимость алгоритма и самой задачи. Рассмотрим два характерных случая.

Пример 12. В задаче сборки кубика Рубика каждое действие обратимо. Но сама задача принципиально необратима, поскольку во время решения теряется информация об исходном состоянии кубика и о промежуточных состояниях.

Задача упорядочения массива принципиально необратима, так как при ее решении массив приводится к стандартному состоянию и невозможно восстановить его исходное состояние. В данном случае формально она может быть сделана обратимой, если каждый элемент x, стоящий на i-м месте, представлять как пару $\langle x, i\rangle$, но

алгоритм может быть все равно необратим, поскольку теряется информация о промежуточных состояниях массива.

5. НАБРОСКИ АЛГЕБРАИЧЕСКОГО ПОДХОДА К МОДЕЛЯМ СИСТЕМ

В предыдущих параграфах подводился итог исследованиям в данной области. Далее в основном приводятся примеры формализаций, которые обосновывают постановку задач для дальнейшей работы.

Пример 13. Рассмотрим условную простейшую модель беспилотника с зарядом, нацеленного на некоторый объект. Во время полета беспилотник можно рассматривать как недиссипативную динамическую систему, и в любой момент ему можно отдать приказ на возвращение назад, а возвращающемуся — на возобновление атаки (конечностью горючего пренебрегаем). После атаки состояние беспилотника уже не может быть изменено (он действует как камикадзе). На стартовой позиции его состояние не меняется, пока не будет отдан приказ на взлет.

Таким образом, фазовое пространство системы делится на три подпространства: подпространство старта, подпространство полета, подпространство завершенной миссии. Управляющих воздействий в реальности три: команда на взлет, команда на прекращение атаки, команда на возврат к атаке. Но формально мы можем свести их к двум: назначение база, назначение цель.

Динамическая система без диссипации порождает группу сдвигов. В данном случае группа становится полугруппой, поскольку движения системы ограничены. Пусть d — подлетное время к цели, x и y меняются в диапазоне $[0, d)$. Определим операции:

$$x + + y = \begin{Bmatrix} x+y, x+y<d \\ 0, \quad x+y \geq d \end{Bmatrix},$$

$$x - - y = \begin{Bmatrix} x-y, x-y>0 \\ 0, \quad x-y \leq 0 \end{Bmatrix}.$$

Теперь определим алгебру управления беспилотником:

$$(\langle 0, 0\rangle * \langle a, x\rangle) = \langle a, 0\rangle,$$

$$(\langle 1, x\rangle * \langle a, y\rangle) = \langle a, x + + y\rangle,$$

$$(\langle 0, x\rangle * \langle a, y\rangle) = \langle a, x - y\rangle \ (x > 0, x < d),$$

$$(0 * z) = (z * 0) = \langle a, 0\rangle = 0.$$

Получившаяся алгебра может быть дискретизирована. Если подлетное время разбить на целое число равных интервалов, значения x и y могут быть сделаны натуральными числами, и получается конечное приближение к непрерывной алгебре управления.

Этот пример показывает принципиальные возможности формализации в алгебрах логико-динамического управления. Для того, чтобы подойти к этой проблеме с несколько более общих позиций, рассмотрим еще пару примеров.

Пример 14. Пусть у нас есть недиссипативная динамическая система, фазовые траектории которой не пересекаются. Тогда, дискретизировав пространство состояний в момент времени 0, получаем дискретно-непрерывную группу, качественно характеризующую область, в которой находится система, и количественно — время ее движения от начальной точки. Дискретизировав еще и время, получаем двумерную дискретную группу приближенных характеристик системы. Ограничив время, получаем полугруппу. Ограничивая пространство начальных состояний, получаем подгруппу (соответственно, подполугруппу) ранее построенной группы (полугруппы). Ограничивая область в фазовом пространстве, в которой может находиться система, сохраняя при этом связность оставшихся частей фазовых траекторий, получаем полугруппу, в которой, начиная с некоторого момента, $x \circ y = x$. Любая из этих (полу)групп может быть пополнена до алгебры управлений добавлением бинарного управляющего элемента, переключающего движение с прямого на обратное; тернарного элемента, включающего приостановку системы; элемента с четырьмя состояниями, где четвертое состояние означает ликвидацию системы (0 в примере 13); и т. д.

Подмечено, что идемпотенты (элементы, для которых $x \circ x = x$) играют в алгебрах роль аналогов условий завершения или переключения процессов в традиционном описании [36].

Пример 15. Пусть имеется диссипативная система. Тогда исходная алгебра — полугруппа, и операции обращения нет. Другие управляющие операции по-прежнему законны.

Пример 16. Пусть имеется логико-динамическая система (диссипативная или нет), поведение которой описывается набором операторов на фазовом пространстве и условий переключения между различными операторами. Для иллюстрации метода превращения такой системы в алгебру пусть у нас система описывается двумя операторами без диссипации и одним условием переключения. Тогда каждый из операторов задает группу сдвигов. Обозначим эти группы G_1 и G_2. Определим полугруппу, которая может рассматриваться как частичная прямая сумма двух групп. Их элементы в большинстве случаев совпадают, потому что они действуют над одним и тем же фазовым пространством, но операции разные.

$$G = (\{1\} \times (G_1 \backslash \{e\})) \cup (\{2\} \times (G_2 \backslash \{e\})) \cup \{0, e\}.$$

Таким образом, задаем размеченное объединение двух групп, склеивая единицы и добавляя ноль. Будем считать, что $\langle 1, e\rangle = \langle 2, e\rangle = e$,

$$\langle a, x\rangle \circ \langle a, y\rangle = \langle a, x \circ y\rangle,$$

$$\langle 1, x\rangle \circ \langle 0, y\rangle = \langle 0, y\rangle \circ \langle 1, X\rangle = 0,$$

$$0 \circ z = z \circ 0 = 0.$$

Произведение элементов из разных групп считается ошибкой.

Действия можно определить, по крайней мере, двумя способами. При первом мы должны переключиться в

точности на поверхности переключения, иначе система ломается. При втором — на поверхности переключения система задерживается до следующей команды. Получаем две различных GAPS над одной и той же полугруппой. В G_{err} действие задается следующим образом:

$(\langle a, x\rangle * \langle a, y\rangle) = \langle a, x \circ y\rangle$ — траектория не проходит через точку переключения;

$(\langle 0, x\rangle * \langle 0, y\rangle) = \langle 1, x \circ y\rangle$ — траектория приходит в точку переключения;

$(\langle 1, x\rangle * \langle 1, y\rangle) = \langle 0, x \circ y\rangle$ — траектория приходит в точку переключения;

0 — в остальных случаях.

В G_{stp} определение следующее:

$(\langle a, x\rangle * \langle a, y\rangle) = \langle a, x \circ y\rangle$ — траектория не проходит через точку переключения

$(\langle 0, x\rangle * \langle 0, y\rangle) = \langle 1, x \circ z\rangle$;

$x \circ z$, z — первая точка переключения на траектории $(\langle 1, x\rangle * \langle 1, y\rangle) = \langle 0, x \circ z\rangle$;

$x \circ z$, z — первая точка переключения на траектории;

0 — в остальных случаях.

Из этих алгебр корректно дискретизируется только вторая, поскольку в первой требуется переключение точно в заданный момент. Это соответствует тому, что в конструктивном анализе переключение точно в заданный момент нереализуемо.

Таким образом, мы приближаемся к другому вычислительному подходу к логико-динамическим системам, проанализированным, в частности, в работах [37, 38]. ♦

Качественные состояния даже необратимых систем с бесконечным числом особых точек в нынешней теории систем сплошь и рядом описываются комплексами групп. В качестве примера весьма изощренных конструкций такого рода, в совокупности с развиваемой здесь техникой дающих возможность описать системы как алгебры, можно привести как описание метода [39] и как его приложение [40].

ЗАКЛЮЧЕНИЕ: ВЫВОДЫ И НАПРАВЛЕНИЯ ДАЛЬНЕЙШИХ ИССЛЕДОВАНИЙ

Отметим, что при общем высоком уровне понятий информатика принципиально отставала от теории динамических систем в уровне абстрактности описания систем. Представляете себе, если бы в теории систем рассматривались не общие уравнения и операторы над фазовыми пространствами, а системы из рычагов, шестеренок и т. п. Переход к GAPS позволил избавиться в теории программирования от «шестеренок»: конкретных операторов алгоритмических языков.

Далее, GAPS показали свою способность описывать классы программных систем, различающиеся по свойствам их функционалов. В статье [33] охарактеризованы классы обратимых программ, полуобратимых программ (возможно взятие назад действия, но не его предотвращение), завершающихся за конечное время автоматных программ. Тем самым алгебраический аппарат еще раз продемонстрировал свою общность и гибкость.

Развитие алгебр программ потребовало исследования ранее не изучавшихся алгебраических структур, прежде всего, неассоциативных алгебр с композицией. Их применение к конкретным классам программ и первые опыты по описанию динамических систем поставили несколько проблем.

Прежде всего, в чистом виде алгебры обратимых вычислений описывают ограниченный класс систем. В реальных программах обязательно присутствуют необратимые действия, например, задание исходных данных (инициализация системы) и чтение получившихся результатов. Далее, класс описываемых систем резко расширяется и становится перспективным для описания многих реальных задач, если рассматривать почти всюду обратимые вычисления, которые обратимы на больших участках, а на стыках между ними могут делаться необратимые действия и приниматься необратимые решения. В этом случае тепловой предел все равно обходится, а выразительные возможности принципиально усиливаются. Но почти всюду обратимые алгебры — еще один неисследованный класс алгебраических систем. Их исследование — актуальная и практически важная задача.

Далее, выразительные средства алгебр мощны для решения сложнейших качественных задач (как показано, например, в работах [39, 40]). Но строящиеся в теории комплексы групп непрерывны. Нет (не требуется для чистой теории) никаких методов их дискретизации и сведения к конечным структурам. Построение конечных приближений непрерывных алгебр — столь же принципиальная задача, какой явилось в свое время осознание топологического характера приближений непрерывных пространств дискретными сетками. Поэтому теория приближений групп, полугрупп и программных алгебр — новое направление, открываемое алгебраическим подходом.

И, наконец, операции конструирования алгебр из простых компонентов не сводятся к широко применяемым в теоретической алгебре прямым, подпрямым и полупрямым произведениям. А переход к фактор-структурам слишком часто разрушает вычислимость. Развитие новых способов пос-

троения алгебр (прежде всего, конечных, но и бесконечные тоже важны) — еще одно направление, исследования в котором необходимы.

Предварительные варианты данной работы докладывались на конференции РАСО'2012 [41], школе молодых ученых ИПУ РАН и на семинаре ИПМ РАН.

ЛИТЕРАТУРА

1. *Глушков В. М.* Синтез цифровых автоматов. — М.: Физматгиз, 1962. — 476 с.
2. *Глушков В. М., Цейтлин Г. Е., Ющенко Е. Л.* Алгебра, языки, программирование. — Киев: Наукова думка, 1978. — 320 с.
3. *Иванов П. М.* Алгебраическое моделирование сложных систем. — М.: Наука, 1996. — 274 с.
4. *Maurer W. D.* A theory of computer instructions // Journal of the ACM. — 1966. — Vol. 13, N 2. — P. 226—235.
5. *Alban Ponse and Mark B. van der Zwaag.* An Introduction to Program and Thread Algebra. — LNCS 3988. — 2006. — P. 445—488.
6. *Bergstra J. A., Bethke I., Ponse A.* Program Algebra and Thread Algebra. — Amsterdam, 2006. — 114 p.
7. *Митчелл Дж.* Основания языков программирования. — М. — Ижевск: РХД, 2009. (Mitchell J.C. Foundation for programming languages, MIT, 1996.)
8. *Мальцев А.И.* Алгебраические системы. — М.: Наука, 1970. — 392 с.
9. *Von Neumann J.* The Role of High and of Extremely High Complication, Fourth University of Illinois lecture, in Theory of Self-Reproducing Automata / A.W. Burks (ed.). — Champaign, Illinois: University of Illinois Press, 1966. — P. 64—73.
10. *Landauer R.* Irreversibility and Heat Generation in the Computing Process // IBM J. Res. Develop. — 1961. — Vol. 5, N 3. — P. 183—191.
11. *Halliday D. and Resnick R.* The Fundamentals of Physics / Extended Third Edition. — N.-Y.: John Wiley and Sons, 1988. — 947 p.
12. *De Benedictis E.* Will Moore's Law be Sufficient // Proc. of the 2004 ACM/IEEE Conference on Supercomputing, 2004.
13. *De Benedictis Erik P.* How to Reach Zettaflops, SAND2008—2492C, Sanda National laboratories, 2008.
14. *Izydorczyk J.* Three steps to the thermal noise death of Moore's law // IEEE Trans. on Very Large Scale Integration (VLSI) Systems. — 2010. — Vol. 18, N 1. — P. 161—165.
15. *Frank M.P.* Introduction to Reversible Computing: Motivation, Progress, and Challenges // Proc. of the 2nd Conference on Computing Frontiers. — P. 385—390.
16. *Frank, M.P.* Towards a More General Model of Reversible Logic Hardware. Superconducting Electronics Approaching the Landauer Limit and Reversibility (SEALeR) Workshop, Annapolis, MD, March 15—16, 2012. Invited talk.
17. *Bennett C.H.* Logical reversibility of computation // IBM J. Res. Develop. — 1973. — Vol. 17, N 6. — P. 525—532.
18. *Шубников А.В., Копцик В.А.* Симметрия в науке и искусстве / Изд. 2-е, перераб. и доп. — М., 1972.
19. *Егоров-Тисменко Ю.К., Литвинская Г.П., Загальская Ю.Г.* Кристаллография. — М.: Изд-во МГУ, 1992.
20. *Егоров-Тисменко Ю.К., Литвинская Г П.* Теория симметрии кристаллов. — М.: ГЕОС, 2000.
21. *Копцик В.А.* Шубниковские группы. — М.: Изд-во МГУ, 1966.
22. *Якушин А.А.* Кризис среднего возраста? Свободное программное обеспечение в высшей школе. — М.: Альт-Линукс, 2013. — С. 17—19.
23. *Chaitin G.J.* Information-theoretic limitations of formal systems // Journal of the ACM. — 1974. — Vol. 21, N 3. — P. 403—424.
24. *Колмогоров А.Н.* Три подхода к определению понятия «количество информации» // Проблемы передачи информации. — 1965. — Т. 1, вып. 1. — С. 3—11.
25. *Nepejvoda N.N.* Abstract Incompleteness Theorems and Their Influence in Methodology // Studia Humana. — 2012. — Vol. 1: 3/4. — P. 43—58.
26. *Гильберт Д.* Избранные труды: в 2 т. / Под ред. А. Н. Паршина. Т. 1: Теория инвариантов. Теория чисел. Алгебра. Геометрия. Основания математики. — М.: Факториал, 1998. — 575 с. — ISBN 5—88688—029—1.
27. *Пойа Д.* Как решать задачу. Пособие для учителей. — М.: Учпедгиз, 1959. — 208 с.
28. *Wang Hao.* Undecidable sentences generated by semantic paradoxes // Journal of Symbolic Logic. — 1955. — Vol. 20. — P. 31—43.
29. *Оревков В.П.* Нижние оценки увеличения сложности выводов после устранения сечений. — Зап. научн. сем. ЛОМИ. — 1979. — Т. 88. — С. 137—162.
30. *Schoenfinkel M.* Uber die Bausteine der mathematischen Logik // Mathematische Annalen — 1924. — B. 92. — S. 305—316.
31. *Барендрегт Х.* Ламбда-исчисление. Его синтаксис и семантика. — М.: Мир, 1985. — 606 с.
32. *Косиков С.В.* Информационные системы: категорный подход. — М.: АО «Центр Юр ИнфоР», 2005. — 92 с.
33. *Непейвода Н.Н.* Абстрактные алгебры различных классов программ // Аппликативные вычислительные системы. — М.: ЮрИнфоР-МГУ, 2012. — С. 103—128.
34. *URL:* http://www.muppetlabs.com/~breadbox/bf/ (дата обращения 23.10.2013).
35. *Buhm 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.
36. *Атаманов В.В.* Алгебраические модели программ, не содержащие условий и операторов. — URL: https://edu.botik.ru/mod—erl/upload/a2eb05b1b73c797396e9262f09e9d40f.pdf (дата обращения 23.10.2013).
37. *Васильев С.Н., Жерлов А.К., Федосеев Е.А., Федунов Б.Е.* Интеллектное управление динамическими системами. — М.: Физматлит, 2000. — 352 с.
38. *Васильев С.Н., Косов А.А.* Анализ динамики гибридных систем с помощью общих функций Ляпунова и множественных гомоморфизмов // Автоматика и телемеханика. — 2011. — № 6. — С. 27—47.
39. *Krasil'shchik I.S., Lychagin V.V., Vinogradov A.M.* Geometry of Jet Spaces and Nonlinear Partial Differential Equations. — N.-Y.: Gordon and Breach, 1986.
40. *Lychagin V., Yumaguzhin V.* Cohomological uniqueness of the Cauchy problem solutions for the Einstein equation // Journal of Geometry and Physics. — 2012. — Vol. 62. — P. 2099—2120.
41. *Непейвода Н.Н.* От численного моделирования к алгебраическому. — Тр. конф. РАСО'2012 / ИПУ РАН. — М., 2012. — Т. 1. — С. 93—103.

Статья представлена к публикации членом редколлегии академиком РАН С.Н. Васильевым.

Николай Николаевич Непейвода — д-р физ.-мат. наук, гл. науч. сотрудник Института программных систем им. А.К. Айламазяна РАН, г. Переславль-Залесский, ☎ (48535) 6-80-68, ✉ nepejvodann@gmail.com.