Аппликативный компьютинг: попытки установить природу вычислений*

Вольфенгаген В.Э.

Аннотация На ранних стадиях программирование представляло собой вид искусства, когда программист писал программу для решения определенной задачи и сопровождал ее более или менее подробно составленной документацией, то теперь создана мощная индустрия программирования с сопутствующей ей инженерией программирования. В настоящее время в исследованиях по программированию или в сфере компьютерных наук, как правило, поддерживаются работы, в которых вносится некоторое небольшое улучшение в решение уже хорошо известной проблемы. Вместе с тем из виду упускаются действительно важные и фундаментальные исследования, ведущие к поиску новых концепций вычислений на компьютере и недостаточное внимание уделяется накоплению знаний в области программирования.

В настоящей работе основное внимание уделено вычислениям с объектами, удельный вес и роль которых в данной области все более возрастает, превращаясь в доминирующую тенденцию.

Об авторе. Вольфенгаген В. Э., д.т.н., профессор (vew@jmsuice.msk.ru). Заведующий кафедрой перспективных компьютерных исследований и информационных технологий (ПКИ и ИТ) в Институте «ЮрИнфоР-МГУ». Область его научных интересов составляют компьютерные науки и информационные технологии, включая аппликативные вычислительные системы, λ-исчисление, комбинаторную логику, системы типов. Проекты РФФИ 93-01-00943-а (ЛАМБДА), 96-01-01923-а (КООАМ), 05-01-00736-а.

1 Аппликативный компьютинг

Для компьютинга центральным вопросом является установление того, что и как может быть эффективно автоматизировано. Это нередко связывают в применением компьютеров, осуществляющих те или иные вычисления. Термин “вычисление” предполагает, что речь пойдет об использовании чисел, об оперировании числами, получении числовых результатов.

В случае же аппликативных систем применим термин “компьютинг”, поскольку в них среди исходных понятий *нет* чисел. Для них самое общее толкование компьютинга, по-видимому, все еще работает, хотя в этом и нет абсолютной уверенности. Выражаясь более точно, с появлением аппликативных систем вопрос о границах представления о компьютинге требует нового изучения. Как оказывается, причин тому имеется несколько.

Если все же соглашаемся говорить о вычислениях, то нужно отдавать себе отчет в том, что аппликативное вычисление не похоже на обычное. Оно выполняется на переплетении цепочек возможных путей вычислений, которые представляют собой связи конвертируемости, отражающие трансформации объектов. При этом одни объекты могут редуцироваться к другим, либо подвергаться экспансии до других объектов.

Рассматриваемые объекты не похожи на те сущности, которыми оперируют в объектно-ориентированном подходе (см. [4]). В процессе конверсии участвует *пара объектов*, первый из которых играет роль *редекса*, а второй — *контракта*. Процесс конверсии является двунаправленным, представляя собой *редукцию* в одном направлении и *экспансию* в другом. Таким образом, пара объектов ‘редекс-контракт’ участвует в процессе ‘редукции-экспансии’, обеспечивающем взаимные переходы между редексами и соответствующими им контрактами. Комплексирование редекс-контракт подчиняется законам аппликативных вычислений.

Объекты, участвующие в вычислении, не обязательно находятся в нормальной форме. Они могут иметь вид редекса, который может быть заменен на соответствующий контракт, т.е. участвовать в процессе редукции. В то же время каждый контракт может быть заменен на его редекс,

* Настоящая работа поддержана Российским Фондом Фундаментальных Исследований, проект 06-07-99005-с

участвуя в процессе экспансии. Представлением для этих объектов может быть сам двунаправленный процесс редукции-экспансии, причем равновесие этого процесса при необходимости может быть смещено в ту или иную сторону. Таким образом, для этих объектов, участвующих в процессе конверсии, имеются представления сущностями об-систем, а последние обладают необходимыми математическим свойствами.

Шаг “вычисления” состоит в следующем. Выбирается пара объектов, один из которых определяется как соответствующий виртуальной функции, порожденной процессом конверсии, а второй — как соответствующий виртуальному аргументу, который также порожден конвертированием. Затем первый объект аппликируется, или применяется к другому. Результат аппликирования рассматривается как представляющий значение, полученное на шаге вычисления. В результате этого процесса возникают пары объектов, первым в которых является ‘аппликация’, а вторым — ‘результат аппликирования’, то есть это пары *аппликация-означивание*.

Цепочки конверсий, в которых участвуют компонентные объекты, сплетаются. Вычисления, в общем случае, могут оказаться бесконечными и никогда не завершаться. В этом случае ответ на вопрос о получении значения оказывается непростым.

В этой общей ситуации лучшим терминологическим решением будет говорить об ‘аппликировании’ вместо ‘вычисления’, а когда речь заходит о результате аппликирования, то есть о ‘значении вычисления’, то здесь можно воспользоваться термином ‘означивание’. Пара ‘аппликирование-означивание’ точнее отражает суть процессов, происходящих в аппликативных системах.

2 Осуществление аппликативного компьютинга

Обсуждение *осуществления* компьютинга можно вести двояких образом. Во-первых, можно исходить от обычного компьютера, основанного на арифметике, а тогда отправным пунктом является система вычислений, в которой можно реализовать погружение аппликативной вычислительной системы. Во-вторых, располагая системой аппликативного компьютинга, в ней можно сформировать погруженную, или встроенную числовую систему, скажем, арифметику, для которой напрямую и подходит термин ‘вычисление’.

Обычные системы вычислений Многие из этих систем вычислений уже осуществлены, а их возможности в большей или меньшей степени освоены на практике, другие находятся в стадии разработки. Для них характерно то, что они находятся в процессе постоянных доработок и усовершенствований, в погоне за дополнительными возможностями. В ряде систем возникает переизбыток возможностей, освоить которые на практике за разумный период времени не представляется возможным.

В результате обычные системы вычислений и соответствующие им языки программирования постоянно разрастаются в объеме, но их выразительные возможности при этом не изменяются. В их основе лежат модели не только далекие от совершенства, но содержащие изъяны, а попытки их преодоления дают нагромождение деталей, усложняя восприятие вычислительных идей, затрудняя реализацию и снижая ее эффективность. Наиболее распространенный стиль вычислений состоит в обработке одного слова в данный момент, разворачивание последовательности моментов приводит к последовательной обработке “слово за словом”. Семантика таких вычислений опирается на переходы состояний, что приводит к разделению вычисляемых конструкций на два класса: *выражения* и *команды*, или операторы. Это само по себе не дает возможности развивать мощные приемы комбинирования имеющихся “вычислительных блоков” для формирования из них более крупных новых блоков. Кроме того, обычные системы вычислений основываются на достаточно сложной и запутанной математической основе, которой не только недостает концептуальной ясности (см. J. Backus, [1]), но и относительно которой никак не возникает согласия в среде специалистов. Их разногласия выражаются в многочисленных публикациях

в виде статей, написанных специальным языком, доступным немногим посвященным, а это со стороны пользователей не добавляет доверия к предлагаемым новым сериям математических и технических ухищрений.

Системы аппликативного компьютинга Некоторые из аппликативных систем осуществлены, но не в полном объеме. При их разработке и применении накоплен известный опыт, но он явно недостаточен, а принципиальные для аппликативных вычислений механизмы так и не удалось реализовать в полной мере и без урезаний и излишних упрощающих ограничений.

Аппликативный компьютинг предполагают комбинационное построение вычисления как относительно самостоятельного 'блока', пользуясь уже имеющимися 'блоками' вычислений. Традиционно в состав аппликативных вычислительных систем, или, сокращенно, АВС, включают системы исчислений объектов, основанные на комбинаторной логике и ламбда-исчислении. Единственное, что существенно разрабатывается в этих системах – это представление об *объекте*. В комбинаторной логике единственный метаоператор – *аппликация*, или, по иной терминологии, *приложение* одного объекта к другому. В ламбда-исчислении два метаоператора – *аппликация* и функциональная *абстракция*, позволяющая связывать одну переменную в одном объекте.

Возникающие в этих системах объекты ведут себя как функциональные сущности, имеющие следующие особенности:

число аргументных мест, или арность объекта заранее не фиксируется, но проявляет себя постепенно, во взаимодействиях с другими объектами;

при конструировании составного объекта один из исходных объектов – функция, – применяется к другому объекту – аргументу, – причем в других контекстах они могут поменяться ролями, то есть функции и аргументы рассматриваются как объекты на равных правах;

разрешается самоприменимость функций, то есть объект может применяться сам к себе.

Вычислительные системы с таким наиболее общими и наименее ограничительными свойствами оказываются в центре внимания современного сообщества computer science. Именно они в настоящее время обеспечивают необходимые метатеоретические средства, позволяя исследовать свойства целевых прикладных теорий, дают основу построения семантических средств языков программирования и обеспечивают средства построения моделей данных/метаданных в информационных системах.

Аппликативные системы программирования Представление об аппликативной вычислительной системе поддерживается аппликативными системами программирования, которые образуют алгебры программ. В них переменные замещаются 'программами', а операции комбинируют одни программы с другими. В этой алгебре решаются уравнения, в которых неизвестными являются программы, а сам процесс решения конструктивно ведет к 'синтезу' требуемой программы как вычисления с заданными свойствами – 'комбинаторными характеристиками'. Комбинаторные, или комбинационные формы оказываются простыми по своей структуре и вместе с тем обладают высокими выразительными возможностями. Теорема о комбинаторной полноте и некоторые другие фундаментальные теоремы, характеризующие законы аппликативных вычислений, задают детали поведения и условия выполнения и завершения для целых классов программ. Семантика аппликативных вычислений слабо связана с состояниями, на относительно большой объем вычислений приходится небольшое число 'изменений состояний', что характеризует их как *не фон Нейманновские вычисления*.

3 Об-системы

В ходе производимого вычисления “аппликативный вычислитель” оперируется цепочками возможных конверсий, меняя местами элементарные “частицы” вычислений — обы. Обы существуют не сами по себе, но наделены специальной структурой, образуя *индуктивный класс*. Таким образом, обы имеют разную сложность. Построение их индуктивного класса начинается с нижнего уровня, на котором находятся элементарные, или *атомарные* обы — константы и переменные. Из них посредством применения *правил образования* строятся более сложные агрегации, которые тоже являются обами, но не элементарными.

Бестиповые вычисления Вычислительная среда комбинаторной логики (см. [10]) является высоко симметричной в том смысле, что если взять два оба M и N, то можно говорить о результате вычисления, когда M применяется, или апплицируется к N, что записывается посредством $(M\,N)$. В этой записи M играет роль левого объекта, а N — правого. Такой об $(M\,N)$ бинарен по своей природе, по построению, а индуктивные классы обов могут быть топологически представлены бинарными деревьями. В этом вычислении первый об M считается функцией, которая применяется к N как к аргументу и, применительно к образам обычной математики, речь идет о формировании значения M “в точке” N. Но если в классической математике поменять местами M и N, то конструкция ‘N от M’ в понимании ‘значение N в точке M’ лишена смысла. В аппликативной же системе такое вычисление не только не запрещается, но и имеет вполне определенный смысл, поскольку с вычислительной точки зрения обы абсолютно одинаковы.

На рис. 1 представлены вычислительные характеристики комбинаторов S и K. Если их за-

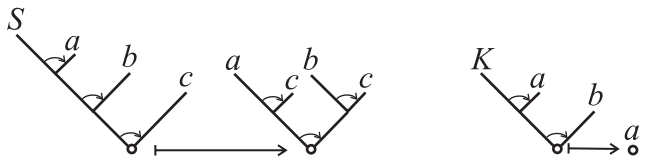

Рис. 1. Характеристики комбинаторов S и K.

писать эквационально, с применением равенства, то получится

$$Sabc = ac(bc),$$
$$Kab = a.$$

На рис. 2 показан процесс “выращивания” дерева вычислений для оператора композиции. Композиция двух функций a и b определяется равенством

$$a \circ b(c) = a(b(c)),$$

что в аппликативной записи превращается в конверсию

$$a(b(c)) \equiv a(bc) = Babc.$$

В исходной об-системе нет комбинатора B с такими свойствами. Но, как оказывается, такой комбинатор можно получить. Процесс генерации состоит в следующем.

Построим дерево вычислений, ассоциированное с композицией (дерево 1). Результирующее дерево вычислений показано в позиции 7. Если бы оно существовало в об-системе, то переход к

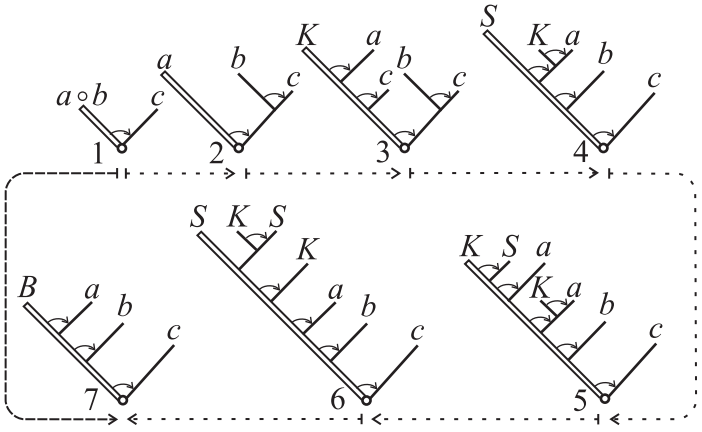

Рис. 2. Характеристика комбинатора композиции B и выращивание дерева вычислений для B, пользуясь деревьями для комбинаторов S и K. { *Пояснение.* Переход от композиции $a \circ b$ (дерево 1) к представляющему комбинатору B (дерево 7) получается при обходе против часовой стрелки. Для этого потребуется шесть шагов, если имеются только деревья вычислений для S и K. }

нему можно выполнить против часовой стрелки. Посмотрим, можно ли сформировать виртуальную композицию, указав соответствующее дерево вычислений, которое строится комбинированием уже известных деревьев вычислений. Это дерево можно вырастить поэтапно, переходя от дерева к дереву по часовой стрелке. Переход 1-2 представляет собой просто применение определения композиции. Переход 2-3 соответствует конверсии

$$a(bc) = Kac(bc),$$

он – осуществим в соответствии с вычислительной, или комбинаторной характеристикой комбинатора K. Переход 3-4 соответствует конверсии

$$Kac(bc) = S(Ka)bc.$$

Аналогично, переходы 4-5 и 5-6 соответствуют конверсиям

$$S(Ka)bc = KSa(Ka)bc, \qquad KSa(Ka)bc = S(KS)Kabc.$$

Но об $S(KS)Kabc$, расположенный последним в цепочке из виртуальных обов, соответствует дереву вычислений с "каноническим" последовательным расположением обов a, b и c вдоль ветвей (показаны на рисунке одинарной линий). Вместе с тем виртуальный об $S(KS)K$ оказался в "вершине" кроны дерева вычислений, формируя его "ствол" (показан на рисунке двойной линий). Другими словами, этот виртуальный об имеет в точности те вычислительные характеристики, которые требуется от оба B, представляющего композицию (дерево 7). Это завершает обход контура конверсий по часовой стрелке: путь от оба $a(bc)$ к обу $S(KS)K$ найден.

Таким методом можно строить деревья вычислений и для многих других математических объектов, предъявляя для них подходящие виртуальные обы и соответствующие их свойствам деревья вычислений.

Ти́повые вычисления Но в аппликативных системах с типами все не так просто: свойства обов описываются функцией приписывания типа, которая определяет их возможности как *процессов преобразования*. Отметим, что обы являются чистыми процессами преобразования, для которых не ставится вопрос о существовании области определения и области значения. Эти области

могут быть пустыми, тогда индивиды в них “не обитают”. Со временем индивид может начать свое существование в одной из областей, находится там в течение определенного периода времени, а потом исчезнуть, прекратив свое существование. Но это не более, чем удобная словесная интерпретация, попытка привнести смысл в высоко абстрактные вычисления. С позиций же аппликативной вычислительной системы речь идет просто о применениях одних обов к другим, а если так проще думать, то области определения и значения можно также идентифицировать обами.

Все это в точности и означает, что обы являются обитателями функциональных пространств, образованных системами высших порядков. Например, наиболее вероятно, что на роль функции в апллицировании одного оба к другому будет взят об из более “высокого” функционального пространства, чем об, претендующий на роль аргумента.

Пара обов характеризуется метаоператором аппликации $(\cdot\,\cdot)$, который с чисто абстрактных синтаксических позиций эту пару обов преобразует в новый об. Тогда, если два оба меняются местами, то результирующий об интерпретируется иначе, теперь роль функции отводится второму обу — тому, который в начале играл роль аргумента, — а роль аргумента отводится первому обу — тому, который в начале играл роль функции. Фактически, при этом все характеристики обов в об-системе остаются прежними, если обы рассматривать *по отдельности*. Однако после перестановки они с другими обами будут взаимодействовать иначе. Отметим, что это новое взаимодействие проявит себя как встроенная, или *погруженная* система типов.

Пары обов имеют тот смысл, что первый играет роль функции, а второй выступает в роли аргумента. Интерпретируемая пара имеет смысл, если вычислитель сможет установить ее значение, будь то действительное, возможное или виртуальное. Для интерпретируемости важно, чтобы первый об имел более высокий потенциальный тип, чем второй. Изменение потенциального типа одного из обов может привести к нарушению интерпретируемости и выводимому несоответствию логическим законам.

Согласно имеющимся на сегодня теория типов не все обы типизируемы. В аппроксимационных решетках, дающих вычислительную модель бестипового ламбда-исчисления [7], есть и другая возможность: рассматривать бесконечные типы. При этом справедливым оказывается изоморфизм между специально построенным представлением типа и отображением этого типа в самого себя. Тогда с вычислительной точки зрения оправдывается *самоприменимость*: объект можно апплицировать к самому себе, что с точки зрения обычной математики недопустимо и лишено смысла. Сам метод работает в топологических пространствах и позволяет отыскивать *неподвижные точки* отображений, что дает практически реализуемую и приемлемую среду для организации циклических вычислений. Обы в аппликативной вычислительной системе рассматриваются как отображения и, как оказывается, для всякого оба можно отыскать неподвижную точку: для этого есть систематический способ. Он достаточно прост, сводится к упаковыванию в пару парадоксального комбинатора Y (см. [9], [10]) и оба, а результат такой упаковки и рассматривается как абстрактное представление неподвижной точки этого оба, который интерпретируется как отображение.

Таким образом, обы при перестановке могут привести к несогласованию некоторых из систем типов, но этого может и не произойти. Для того, чтобы рассогласование не происходило, применяют правила образования и правила преобразования обов, к которых учитываются схемы типов. Удовлетворительными с этой точки зрения вычислительными свойствами обладают аппроксимационные решетки. Обы в этих топологических структурах при интерпретации проявляют полезные логические свойства, что немаловажно для семантических применений. В частности, при этом для об-системы строится модель вычислений, обладающая полнотой и непротиворечивостью.

4 Обы в системе типов

Пары обов, упакованные метаоператором аппликации, существуют в виртуальном мире значений функций. Особая роль при этом отводится тождественным преобразованиям – единичным обам. Дело в том, что с вычислительной точки зрения их можно отождествить с преобразуемыми обами. Тогда обы проявляют заложенную в них вычислительную способность к *композированию*, то есть к участию в метаоперации композиции, имеющей обычный математический смысл.

Процессор списков В системе обов имеются специальные исходные обы-комбинаторы, пользуясь которыми можно производить новые обы, в том числе и комбинаторы. Среди комбинаторов существует производный об-комбинатор, называемый комбинатором пары. При упаковывании комбинатора пары вместе с двумя обами возникает виртуальный об, проявляющий математические свойства упорядоченной пары. В этой системе также существует производный об-комбинатор, называемый композитором. При упаковывании композитора вместе с двумя обами возникает виртуальный об, проявляющий математические свойства композиции функций. Упорядоченная пара – это пример конечной последовательности длины два. Конечные последовательности большей длины формируются композицией частичной упакованных упорядоченных пар, когда комбинатор пары упакован только с первым обом, а "подстановочное место" для второго оба – свободно и ожидает прихода какого-либо оба для своего заполнения. Эти виртуальные объекты, проявляющие свойства конечных последовательностей, можно оснастить семейством виртуальных обов, проявляющих математические свойства проекций над конечными последовательностями. Такое комбинирование обов порождает виртуальную вычислительную систему, которая реально существует в программировании – это система обработки списков *Lis*t *P*rocessor, или, сокращенно, Lisp. Этот процессор списков был реализован Дж. Маккарти (J. McCarthy, [5]) в 1960 г., исходя из несколько иных посылок и представлений. Но вскоре возможности этого процессора стали осознаваться именно в связи с природой аппликативных вычислений. Вместе с тем на практике полных выразительных возможностей об-системы по разным причинам достигнуть так и не удалось. Однако имевшаяся система программирования Lisp получила дальнейшее развитие и приобрела большую популярность в области искусственного интеллекта именно как "ассемблер знаний".

Об-система по отношению к этой частной, *индивидной* вычислительной системе ведет себя как *концепт*-система, способная порождать семейства виртуальных вычислительных систем с разными выразительными возможностями. Конечно, обы виртуальной индивидной системы могут взаимодействовать по законам аппликативных вычислений и с произвольными обами, что придает ей свойство *расширяемости*.

Комбинаторная арифметика Попытки строить виртуальные системы предпринимались и до этого, в том числе для анализа математических понятий. Примером может служить комбинаторная арифметика и система нумералов А. Черча. Действительно, в об-системах – будь то комбинаторная логика или λ-исчисление, – среди первичных сущностей *нет* чисел. Это замечательная особенность об-систем сразу привлекает внимание, поскольку в математике именно число относится к первичным фундаментальным понятиям, а из арифметики, по сути, строится вся математика. Удалось установить производные обы-комбинаторы, которые ведут себя как цифры – целые числа, нумералы. Это виртуальные объекты, или, как их еще называют в формализмах, комбинаторные представления нумералов. Разработаны и семейства виртуальных объектов, представляющих арифметические операторы над нумералами (см. анализ проблематики и библ. в [10]). Исследование представлялось и представляется в настоящее время настолько привлекательным, что был построен производный об-комбинатор рекурсии и для виртуальной системы рекурсивных вычислений проверена справедливость теоремы С. Клини о том,

что всякая частично рекурсивная функция может быть (в слабом смысле) представлена обом-комбинатором.

Виртуальная система обов, представляющая рекурсивные вычисления сама по себе может рассматриваться как концептуальная основа процессора списков.

Логика высказываний Среди исходных сущностей об-системы нет не только чисел, но *нет* ни представления об истине или лжи, ни условных конструкций или иных логических связок. Другими словами, об-система может оказаться пригодной и для построения виртуальных систем, представляющих логику. В аппликативной вычислительной системе есть только обы и их приложения, позволяющие упаковывать одни обы вместе с другими в ожидании, когда будет порожден нужный вычислительный контекст и они проявят свои вычислительные свойства. А это значит, что логические рассуждения можно свести к прямому вычислению.

Это сразу было проверено для логики высказываний (см. [10], с. 169) и ожидаемые вычислительные свойства подтвердились. Были установлены виртуальные обы, которые, соответственно являлись представлениями истины и лжи в логики высказываний, которые удалось согласовать с представлениями для условной конструкции, отрицания и конъюнкции. Все эти представления обладали относительно логики высказываний точно таким же поведением, как и "обычные" сущности, которые они представляли. Это вновь заставило задуматься о природе истины и лжи, а также высказываний и рассуждений в терминах высказываний. Действительно, об-системы подчиняются законам аппликативных вычислений, которые на первый взгляд не имеют ничего общего с обычно логикой или арифметикой. Вместе с тем, комбинируя обы определенным образом, удается получить такие упаковывания обов, совместное поведение которых удовлетворяет законам логики или арифметики.

Если взять комбинатор K с характеристикой $Kxy = x$ и комбинатор (KI) с характеристикой $KIxy = y$, то они являются подходящими представлениями для истины $\mathbb{T}$ $(true)$ и лжи $\mathbb{F}$ $(false)$ соответственно в следующем смысле.

Истинностные значения и условные выражения. Сами объекты $\mathbb{T}$ и $\mathbb{F}$ рассматриваются как *метапеременные*, то есть их можно замещать подходящими замкнутыми выражениями, конвертируемыми соответственно к $\mathbb{T}$ и $\mathbb{F}$.

Если b — терм, принимающий значение $true$ или $false$, то условное выражение

$$\texttt{if}\ b\ \texttt{then}\ m\ \texttt{else}\ n$$

имеет тот смысл, что

$$\begin{aligned}\texttt{if}\ true\ \texttt{then}\ m\ \texttt{else}\ n &\to m,\\ \texttt{if}\ false\ \texttt{then}\ m\ \texttt{else}\ n &\to n.\end{aligned}$$

В этом случае для обозначения отношения редукции использован символ '$\to$'.

Выполняя погружение условной конструкции в λ-исчисление или комбинаторную логику, положим

$$BMN,$$

где B, M, N — соответственно представления b, m, n. Таким образом,

$$\begin{aligned}\mathbb{T}MN &\to M,\\ \mathbb{F}MN &\to N,\end{aligned}$$

а если воспользоваться конструкцией *упорядоченной пары*

$$[x, y] \equiv \lambda r.rxy \equiv Dxy,$$

то предыдущие выражения переписываются как

$$[M,N]\mathbb{T} \equiv (DMN)\mathbb{T} \to M,$$
$$[M,N]\mathbb{F} \equiv (DMN)\mathbb{F} \to N.$$

Логические связки При построения условных конструкций в программировании, как известно, разрешается применение логических связок, среди которых остановимся на NOT, которую будем обозначать через $\neg$, и AND, которую обозначим через $\&$. Положим по определению:

$$\neg \equiv \lambda b.\lambda x.\lambda y.byx,$$
$$\& \equiv \lambda b.\lambda v.\lambda x.\lambda y.b(vxy)y.$$

Непосредственными вычислениями, пользуясь правилами λ-исчисления, можно убедиться, что представления таблиц истинности для этих связок корректны:

$$\neg\mathbb{F} \to \mathbb{T},$$
$$\neg\mathbb{T} \to \mathbb{F},$$

и

$$\&\mathbb{F}\mathbb{F} \to \mathbb{F},$$
$$\&\mathbb{F}\mathbb{T} \to \mathbb{F},$$
$$\&\mathbb{T}\mathbb{F} \to \mathbb{F},$$
$$\&\mathbb{T}\mathbb{T} \to \mathbb{T}.$$

Можно заметить, что $\&$ дает "короткозамкнутое" вычисление в том смысле, что

$$\&\mathbb{F}a \to \mathbb{F}$$

даже если объект a не имеет нормальной формы.

5 Натуральные числа

Отметим, что при построении λ-исчисления среди исходных объектов числа и/или арифметические операции не были использованы вовсе. Это означает, что среди замкнутых термов λ-исчисления можно попытаться отыскать такие из них, которые являются подходящими представлениями натуральных чисел.

Нумералы в системе λ-конверсии Для образов натуральных чисел выберем замкнутые термы, называемые *нумералами Черча*:

$$\begin{aligned}
Z_0 &\equiv \lambda f.\lambda x.x && \equiv \lambda f.\lambda x.f^0x,\\
Z_1 &\equiv \lambda f.\lambda x.fx && \equiv \lambda f.\lambda x.f^1x,\\
Z_2 &\equiv \lambda f.\lambda x.f(fx) && \equiv \lambda f.\lambda x.f^2x,\\
&\dots && \dots,\\
Z_i &\equiv \lambda f.\lambda x.\underbrace{f(f\dots(f}_{i\text{ раз}}x)\dots) && \equiv \lambda f.\lambda x.f^ix,\\
&\dots && \dots.
\end{aligned}$$

Для того, чтобы нумералы Z_i образовали натуральный ряд, требуется добавить операцию 'следования за', или *прибавления единицы*, обозначаемую через σ:

$$\sigma \equiv \lambda n.\lambda f.\lambda x.f(nfx).$$

Действительно,

$$\begin{array}{lll} \sigma Z_0 \to \lambda f.\lambda x.f(Z_0 f x) \to \lambda f.\lambda x.f x & \equiv Z_1, \\ \sigma Z_1 \to \lambda f.\lambda x.f(Z_1 f x) \to \lambda f.\lambda x.f(f x) & \equiv Z_2, \\ \quad \ldots \qquad\qquad\quad \ldots \qquad\qquad & \ldots, \\ \sigma Z_i \to \lambda f.\lambda x.f(Z_i f x) \to \lambda f.\lambda x.f(f^i x) & \equiv Z_{i+1}, \\ \quad \ldots \qquad\qquad\quad \ldots \qquad\qquad & \ldots. \end{array}$$

то есть $\sigma Z_i = Z_{i+1}$, и функция σ на нумералах Черча определяет функцию прибавления единицы.

Нумералы в системе комбинаторов Приступим к определению представления числа средствами комбинаторов. Для любых термов X, Y будем использовать сокращение

$$X^n Y \equiv \begin{cases} \underbrace{X(X(\ldots(X}_{n} Y)\ldots)) & \text{для } n > 0; \\ X^0 Y \equiv Y & \text{для } n = 0. \end{cases}$$

Определение [нумералы]. Каждому натуральному числу n поставим в соответствие терм

$$Z_n \equiv (SB)^n(KI).$$

Эти термы назовем нумералами.

Пример. Таким образом,

$$\begin{array}{rcl} Z_0 & = & KI, \\ Z_1 & = & SB(KI), \\ Z_2 & = & SB((SB)(KI)), \\ & \ldots, & \\ Z_n & = & SB((SB)^n(KI)), \\ & \ldots & \end{array}$$

Имеются и другие последовательности, которые можно использовать вместо Z_n, но принятое определение имеет то техническое преимущество, что

$$(Z_n) \qquad\qquad Z_n F X \rhd F^n X$$

для любых F и X (сравните с нумералами Черча [2]).

Тест нуля Для работы с последовательностью нужно ввести распознающую функцию, или предикат $isZero$ проверки нумерала на свойство быть нулем:

$$isZero \equiv \lambda n.\lambda x.\lambda y.n(\lambda z.y)x.$$

Действительно,

$$\begin{array}{lll} isZero\, Z_0 \to \lambda x.\lambda y.Z_0(\lambda z.y)x & & \\ \qquad\qquad \to \lambda x.\lambda y.x & \equiv \mathbb{T}, & \\ isZero\, Z_{i+1} \to \lambda x.\lambda y.Z_{i+1}(\lambda z.y)x & & \\ \qquad\qquad = \lambda x.\lambda y.\sigma Z_i(\lambda z.y)x & & \\ \qquad\qquad \to \lambda x.\lambda y.(\lambda z.y)(Z_i(\lambda z.y)x) & & \\ \qquad\qquad \to \lambda x.\lambda y.y & \equiv \mathbb{F}. & \end{array}$$

Построение арифметики Пользуясь этими представлениями и добавив к ним *оператор неподвижной точки* Y, действительно можно выразить произвольную вычислимую на натуральных числах функцию. Отложим это для последующего рассмотрения, а вместо этого введем представления для арифметических операторов *сложения* $\oplus$ и *умножения* $\otimes$:

$$\oplus \equiv \lambda m.\lambda n.\lambda f.\lambda x.mf(nfx),$$
$$\otimes \equiv \lambda m.\lambda n.\lambda f.m(nf).$$

Пример. Рассмотрим выполнение сложения двух нумералов Z_2 и Z_3:

$$\begin{aligned}
\oplus Z_2\, Z_3 &\to \lambda f.\lambda x.Z_2 f(Z_3 fx)\\
&\to \lambda f.\lambda x.Z_2 f(f^3 x)\\
&\to \lambda f.\lambda x.f^2(f^3 x)\\
&\equiv \lambda f.\lambda x.f^2\ \ (f(f(fx)))\\
&\equiv \lambda f.\lambda x.f(f\ \ (f(f(fx)))\ \ )\\
&\equiv \lambda f.\lambda x.f^5 x\\
&\equiv Z_5,
\end{aligned}$$

что совпадает с интуитивно ожидаемым результатом.

Пример. Рассмотрим выполнение умножения двух нумералов Z_2 и Z_3:

$$\begin{aligned}
\otimes Z_2\, Z_3 &\to \lambda f.Z_2(Z_3 f)\\
&\to \lambda f.\lambda x.(Z_3 f)^2 x\\
&\to \lambda f.\lambda x.Z_3 f(Z_3 fx)\\
&\equiv \lambda f.\lambda x.Z_3 f(f^3 x)\\
&\equiv \lambda f.\lambda x.f^3(f^3 x)\\
&\equiv \lambda f.\lambda x.f^3\ \ (f(f(fx)))\\
&\equiv \lambda f.\lambda x.f(f(f\ \ (f(f(fx)))\ \ ))\\
&\equiv \lambda f.\lambda x.f^6 x\\
&\equiv Z_6,
\end{aligned}$$

что совпадает с интуитивно ожидаемым результатом.

Вопросы эффективности реализации этой арифметики нуждается в дополнительном изучении. Кроме того, как оказывается, ряд представлений для рассмотренных сущностей имеет случайные свойства, которые не дают закономерности. В частности,

$$Z_0 \equiv \mathbb{F}.$$

Другой пример дается вариантом представления для функции прибавления единицы:

$$\sigma' \equiv \lambda n.\lambda f.\lambda x.nf(fx),$$

что приводит для σ и σ' к различным результатам, когда эти функции применяются к объектам, не являющимся нумералами.

Отметим, что в этом варианте программирования в λ-исчислении требуется нормально-порядковая редукция. В частности, для условных выражений требуется, чтобы $\mathbb{T}ab$ редуцировалось к a, даже если b не имеет нормальной формы. Кроме того, как оказалось, выражение нужно редуцировать к нормальной форме, но не обязательно к каноническому виду, чтобы установить результирующий нумерал или истинностное значение (см. примеры 5, 5).

Более того, нумерал Z_1 имеет две нормальные формы: $\lambda f.\lambda x.fx$ и $\lambda f.f$, которые связаны отношением η-редукции:

$$\lambda f.\lambda x.fx \equiv \lambda f.(\lambda x.fx) \to_\eta \lambda f.f,$$

поскольку $\lambda x.fx \to_\eta f$.

Таким образом, вычисления с нумералами Черча основаны на $\beta\eta$-редукции, то есть на выражениях, которые не содержат ни β-, ни η-редексов.

6 Композиционные структуры обов

Композиции обов и возникающие при этом структуры вызывают повышенный интерес в одной специальной системе обов. В этой системе для произвольного оба A требуется выполнением равенства

$$A = A \circ A,$$

которое характеризует A как *домен*, а для обов f, A, B — равенства

$$f = B \circ f \circ A,$$

которое характеризует f как *отображение*, функциональную сущность.

Далее потребуется развить достаточно специальное представление. Функцию h из A в T записываем посредством $h : A \to B$. Рассмотрим функции “транзакций” g, g' и функции “клонирования” f, f', для которых

$$g : T \to T', \quad g' : T' \to T''$$

и

$$f : A \to B, \quad f' : B \to C.$$

Их будем рассматривать совместно с функциями $h : A \to T$, $h : B \to T'$ и $h'' : C \to T''$. Поведение этих отображений представлено на рис. 3. Особенность этой композиционной структуры

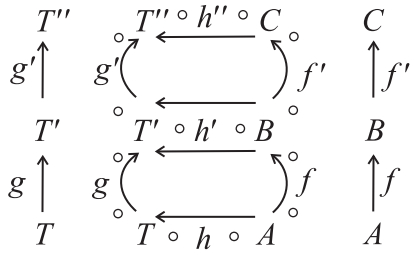

Рис. 3. Пример g, g'-транзактированной, f, f'-клонированной структуры обов.

можно пояснить следующим образом. Пусть B характеризует “настоящее”, тогда A — это возможное “будущее”, а C — “прошлое”. “Разворачивание” событий происходит вдоль эвольвент f, f'’. Обращаем внимание, что события разворачиваются именно от C к B и от B к A. Пусть также об T под воздействием “транзакции” g превращается в об T', а об T' под воздействием g' — в об T''.

Прошлое характеризуется обом

$$T'' \circ h'' \circ C,$$

а возможное будущее соответствует обу

$$T'' \circ g' \circ g \circ h \circ f' \circ f \circ A.$$

Таким образом, под воздействием эвольвент f, f' и транзакций g, g' об h'' из прошлого C трансформируется в виртуальный об $g' \circ g \circ h \circ f' \circ f$ из будущего A. Сходные представления были развиты Д. Скоттом (D. Scott, [8]) применительно к функторным конструкциями доменов.

7 Системы аппликативного программирования

К настоящему времени аппликативные вычислительные системы начинают рассматриваться как основа для проектирования компьютеров, систем *квантового* компьютинга и квантовых языков программирования (см., например, P. Selinger and B. Valiron, [6]). До сих пор этого не произошло из-за массового распространения компьютеров с фон Нейманновской архитектурой и сложившейся индустрией производства для них программного обеспечения. Аппликативная система вычислений, в отличие от фон Нейманновской, является высоко симметричной и концептуально ясной, имеющей хорошо обоснованное математическое представление.

Вместе с тем препятствие для их реализацией на нынешнем массово распространенном и внедренным в сознание технологическом уровне наталкивается на две трудности – в об-системе нет “памяти” и изначально отсутствует чувствительность к “предыстории вычислений”, – а это краеугольные камни нынешнего фон Нейманновского компьютинга. Другая причина – в большинстве аппликативных систем используются ламбда-абстракции, апплицирование которых к объектам приводит к необходимости использовать правило подстановки в наиболее общем и наименее ограниченном виде, как принцип *неограниченного* свертывания. Такая мощная возможность вычислений, дающая неограниченные выразительные возможности, все еще не реализована из-за нынешней концепции компьютера и компьютинга, а также из-за инерции мышления большинства специалистов, прекрасно образованных в области традиционного компьютинга, но не обладающих должной подготовкой и развитой интуицией в аппликативном компьютинге и вычислениях в об-системах.

При реализации аппликативной системы, прежде всего, пытаются оснастить ее памятью и повысить ее эффективность для организации вычислений на фон Нейманновских компьютерах. Все это привело к непрекращающимся попыткам рассматривать аппликативную систему как ‘встроенную систему’, но встраивание, или погружение ведется в бóльшую, объемлющую систему фон Нейманновского типа. Примером такого рода могут служить реализации Lisp-систем, в которые изначально заложена идея именно аппликативных вычислений в об-системе. Однако Lisp-система обычно рассматривается как подмножество расширенной среды, которая оснащена возможностями взаимодействия с вычислительными системами – будь то аппаратное или программное обеспечение, – фон Неймана. В результате возникает сложная и громоздкая система, утратившая свою математическую лаконичность и которую весьма сложно осмыслить разработчику компьютеров нового типа.

Список литературы

1. Backus J. *Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs.* – Commun. ACM, Vol. 21, №8, 1978. – pp. 613–641.
   DOI http://doi.acm.org/10.1145/10.1145/359576.359579.
   Русский перевод: Бэкус Дж. *Можно ли освободить программирование от стиля фон Неймана? Функциональный стиль и соответствующая алгебра программ.* – Пер. с англ. Мартынюка В. В. – В кн.: *Лекции лауреатов премии Тьюринга за первые двадцать лет 1966-1985.* – Под ред. Р. Эшенхерста. – М.: Мир, 1993. – с. 84-158. 2
2. Church A. *The calculi of lambda conversion.* – Prinston University Press, Prinston, 1941. 5
3. Curry H. B. *Functionality in combinatory logic.* – Proc. National Academy of Sciences of the USA, Vol. 20, 1934. – pp. 584–590.
4. Date C. J., Darwen H. *Foundation for future database systems. The Third Manifesto. A detailed study of the impact of type theory on the relational model of data, including a comprehensive model of type inheritance.* – Addison-Wesley Publishing Co, 2-nd edition, 2000.
   Русский перевод: Дейт К., Дарвен Х. *Основы будущих систем баз данных. Третий манифест. Детальное исследование влияния теории типов на реляционную модель данных, включая полную модель наследования типов.* – Пер. с англ. Кузнецова С. Д. и Кузнецовой Т. А. Под ред. Кузнецова С. Д. – 2-е издание, М.: Янус-К, 2004. – 656 с.
   См. также: Darwen H., Date C. J. *The Third Manifesto.* – 2002. http://www.thethirdmanifesto.com/ 1
5. McCarthy J. *Recursive functions of symbolic expressions and their computation by machine, Part I.* – Commun. ACM, Vol. 3, №4, 1960. – pp. 184–195.
   DOI http://doi.acm.org/10.1145/367177.367199 4

6. Selinger P. and Valiron B. *A lambda calculus for quantum computation with classical control.* — Mathematical Structures in Computer Science, 16(3), 2006. — pp. 527-552. 7
7. Scott D. S. *The lattice of flow diagrams.* — Lecture Notes in Mathematics, 188, Symposium on Semantics of Algorithmic Languages. — Berlin, Heidelberg, New York: Springer-Verlag, 1971, pp. 311-372. 3
8. Scott D. S. *Relating theories of the lambda calculus.* — Hindley J., Seldin J. (eds.) To H.B.Curry: Essays on combinatory logic, lambda calculus and formalism.- N.Y.& L.: Academic Press, 1980, pp. 403-450. 6
9. Вольфенгаген В. Э. *Комбинаторная логика в программировании. Вычисления с объектами в примерах и задачах.* — М.: МИФИ, 1994. — 204 с.; 2-е изд., М.: АО «Центр ЮрИнфоР», 2003. — 336 с. 3
10. Вольфенгаген В.Э. *Методы и средства вычислений с объектами. Аппликативные вычислительные системы.* — М.: JurInfoR Ltd., АО «Центр ЮрИнфоР», 2004. — xvi+789 с. 3, 3, 4, 4