О теориях имен и ссылок в формальных языках и последствиях для функционального и объектно-ориентированного программирования

А.И. Адамович[1], Анд.В. Климов[2]

1 Институт программных систем им. А.К. Айламазяна РАН
2 Институт прикладной математики им. М.В. Келдыша РАН

Аннотация. Обсуждается давняя проблема адекватной формализации понятия локальных имен в математических формулах и семантики ссылок в объектно-ориентированных языках, взятых «в чистом виде» без объектов. Объясняется, почему существующие подходы нельзя считать подходящими решениями. Дается введение в сравнительно свежие работы по теориям имен и ссылок группы, которую возглавляет Andrew Pitts. Разбирается понятие референциальной прозрачности, в которой вместо обычного равенства значений используется контекстуальная эквивалентность. Это основное свойство, на которые опираются указанные теории: оно сохраняется при расширении чисто функционального языка именами и ссылками как данными. Утверждается, что это свойство вместе со многими другими может быть сохранено и для объектов с ограниченно изменяемыми состояниями. Это приводит к модели вычислений между функциональной и объектно-ориентированной, которая позволяет детерминированную параллельную реализацию.

Ключевые слова: связанные переменные, локальные имена, ссылки на объекты, номинальная теория множеств, номинальные методы, референциальная прозрачность, контекстуальная эквивалентность, монотонные объекты

On theories of names and references in formal languages and implications for functional and object-oriented programming

A.I. Adamovich[1], And.V. Klimov[2]

1 Ailamazyan Program Systems Institute of RAS
2 Keldysh Institute of Applied Mathematics of RAS

Abstract. The long-standing problem of adequate formalization of local names in mathematical formulas and the semantics of references in object-

oriented languages taken “as is” without objects is discussed. Reasons why the existing approaches cannot be considered suitable solutions are explained. An introduction to the relatively recent works on the theories of names and references of the group headed by Andrew Pitts is given. The notion of referential transparency, in which contextual equivalence is used instead of the usual equality of values, is analyzed. This is the main property on which these theories are based: it is preserved when extending a purely functional language with names and references as data. It is argued that this property, along with many others, can be preserved for objects with limited mutable states. This leads to a model of computation between functional and object-oriented ones, which allows a deterministic parallel implementation.

Keywords: bound variables, local names, references to object, nominal set theory, nominal methods, referential transparency, contextual equivalence, monotonic objects

1. Введение

В статье рассматривается старая проблема адекватной формализации локальных имен, связанных переменных в формальных языках и ссылок в объектно-ориентированных языках. В математике и программировании за многие годы выработался некий стиль работы с именами и ссылкам, и кажется, что проблем нет. Однако в разделе 2 мы покажем, что это вовсе не так: представленные в учебниках логики (например, [**11**]) способы работы с формулами с кванторами содержат неприятные тонкости, артефакты выбранного способа формализации. Другое решение, представленное Н. Бурбаки в построении базового языка Трактата [**5**] в виде графового представления математического текста, отнюдь не лучше, хотя и поучительно. Его разберем в разделе 3.

Недостающие теории имен уже почти тридцать лет развиваются группой под руководством Andrew Pitts’a [**12**][**12**][**13**][**14**][**17**]. Укажем на эти работы в разделе 4, а затем в разделе 5 разберем идею предмета исследования и анализа этими теориями — чисто функциональный язык, области данных которого расширены понятием имен. Впоследствии, в области объектно-ориентированного программирования, они превратятся в ссылки на объекты, но здесь имена и ссылки рассматриваются сами по себе без того, что они именуют и на что ссылаются. Для математиков этот язык нетривиален, поскольку в нем нарушается *референциальная прозрачность* (*referential transparency*).

В разделе 6 исправим ситуацию и покажем, что если в понятии референциальной прозрачности привычное равенство значений заметить на *контекстуальную эквивалентность* (*contextual equivalence*, *observational equivalence*), то такая *модифицированная референциальная прозрачность* выполняется для функционального языка с именами и ссылками. Это понятие является «краеугольным камнем», на который

опираются теории типа тех, что разработаны группой A. Pitts'a. Какие свойства при этом сохраняются, а какие нарушаются, разберем в разделе 7.

В разделе 8 перечислим шаги, которые можно (и нужно) сделать дальше от языка с именами и ссылками «в чистом виде» к объектам с состояниями — сначала неизменяемым (immutable), а потом с ограниченным изменением, но так, чтобы сохранялась референциальная прозрачность на основе контекстуальной эквивалентности.

Здесь же укажем на практическое значение данных работ в области параллельного программирования: сохранение контекстуальной эквивалентности результатов при любом порядке вычислений означает детерминированность. В тех случаях, когда параллельная программа обладает детерминированностью как общего результата, так и промежуточных результатов вычислений вызовов функций, ее намного легче отлаживать, она надежна и предсказуема.

Основной вклад данной работы в том, что мы показываем связь проблем и решений по адекватной формализации имен и ссылок с построением расширенной функциональной или ограниченной объектно-ориентированной модели вычислений, в которой сохраняются многие ценные свойства функциональных программ, а области данных, и тем самым классы решаемых задач, расширяются с представления деревьев на обработку произвольных графов, почти столь же эффективную как в объектно-ориентированных языках.

2. Проблема имен и ссылок как данных в метаматематике и программировании

Начнем издалека. В языках программирования понятие *имени* появляется в двух, казалось бы совсем разных обличьях: *идентификаторы переменных* в тексте программ и *ссылки* на объекты. (Ссылки и указатели возникли в программировании и имеют смысл до полного оформления понятия объекта в объектно-ориентированных языках, и этого нам достаточно).

Идентификаторы и ссылки «живут» в разных временах: идентификаторы переменных — в *static time,* «статическое время» от того момента, когда программа пишется, до того, когда она преобразуется компилятором в исполнимый код; ссылки — в *run time*, период исполнения, когда программа в основном не меняется, а порождаются и обрабатываются данные. Эти времена пересекаются, когда речь идет о программах, анализирующих и преобразующих программы, о *метавычислениях*, где интерпретаторы и компиляторы — самый простой, базовый вид таких систем. Здесь переменные обычно представляются ссылками на некоторые объекты, хранящие необходимую информацию. Возникает вопрос: такое отображение переменных в ссылки — это чисто программистский «трюк» или есть чего-то глубинно общее?

А что в математике? Пользуясь языком математики в повседневной научной деятельности мы по-разному воспринимаем *переменные*, *буквы* как элементы текста на бумаге, с которым работает человек («*static time*») и области *данных*, *значений*, по которым переменные «пробегают» в некоем идеальном «*run time*». Однако они пересекаются в *метаматематике*, в логике, когда язык математики становится объектом исследования, описания, порождения, анализа и преобразования. И затем в программировании — в системах компьютерной алгебры, когда уже не человек, а машина по предписанным программам манипулирует с формулами, с «кодом» на математических языках.

Если программисты могут ограничить свое мышление приемами реализации, не задумываясь о глубинных взаимосвязях, даже если бы их обнаружение могло бы упростить программные системы, то хорошие математики не только (и не столько?) решают задачи, как наводят порядок в строгих понятиях и формальных языках, «машут» бритвой Оккама направо и налево, чтобы разложить объект изучения на части так, чтобы после сборки назад мир стал проще. Давайте посмотрим, как сейчас выглядит математический мир там, где переменные в формулах становятся *объектом* определения, изучения, манипулирования.

В основе математического языка лежит фундаментальное понятие *свободных и связанных* переменных. Например, в формуле $\forall x\, \exists y\, P(x, y, z)$ переменные x и y — связанные кванторами $\forall$ и $\exists$ соответственно, а z — свободная переменная, параметр всей формулы. Кажется, всё просто, но открываем классический учебник [**11**, стр. 56] и видим, например, такое заумное (всего лишь вспомогательное!) понятие:

> Терм t называется *свободным для переменной* x в формуле A, если никакое свободное вхождение x в A не лежит в области действия никакого квантора $\forall y$, где y — переменная, входящая в t.

Эта фраза всего лишь означает, что терм t можно подставить вместо x в формулу A без каких-либо проблем. Другой способ преодолеть те же «проблемы» — в определении подстановки сделать оговорку о переименовании:

> При подстановке терма t в формулу A переменные под кванторами в формуле A переименовываются по надобности так, чтобы не возникло конфликтов со свободными переменными в t.

Например, при подстановке $f(x)$ вместо y в $\forall x\, \mathrm{P}(x, y)$ нужно заменить x, скажем, на z:

$$\forall z\, \mathrm{P}\big(z, f(x)\big)$$

Кроме того, в теоремах и доказательствах о формальных языках постоянно встречаются обороты типа «с точностью до переименования переменных», «с точность до α-эквивалентности», которые загромождают

код в системах машинной проверки доказательств (*proof assistants*) типа Coq и Agda.

Однако откуда проблемы? Наше интуитивное восприятие кванторов вроде не содержит таких «фокусов». Смотря на связанную квантором переменную, мы сразу воспринимаем ее по-другому, чем свободную, — не просто как букву. Мы представляем, что z в вышеприведенной формуле — это совсем другая сущность, чем буква свободной переменной x в ней же:

> Буква связанной переменной, стоящая под квантором и в местах ее вхождения в область действия квантора, изображает некую сущность, которая *заведомо отличается* как от букв всех переменных, так и всех принимаемых ими значений.

Интуитивно мысля так, видим, что не требуется никаких переименований при подстановке терма вместо такой «сущности». Эти «сущности» по определению все разные под всеми кванторами во всех формулах, с которыми идет работа и вообще со всеми формулами и кванторами в мире.

Таким образом, и понятие терма, свободного для данной переменной, и требование переименования при подстановке — это лишь *артефакты* конкретного способа формализации в привычных нам языках математики в общепринятых учебниках по логике, а к существу представления о кванторах и связанных переменных отношения не имеют.

3. Решение Н. Бурбаки

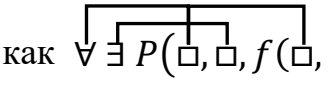
Конечно, это не новость. Достаточно вспомнить, какую формализацию кванторов дали Н. Бурбаки в своем Трактате [**5**]. Она не несет на себе этих артефактов и ближе к нашему наивному пониманию.

По Н. Бурбаки, кванторы ∀ и ∃ с переменными — это запись на бумаге для человека формул на базовом языке математики, введенном в Главе 1 [**5**]. При этом базовый язык оказался не одномерным и даже более сложным, чем «деревянистый». Заметим, что термы, формулы, коды программ и т.п. в традиционной построении формальных языков — это *деревья*, а переход от деревьев к графам (например, на стадии разрешения имен в компиляторах) — это совсем не тривиальный шаг со сменой представления.

В базовом язык математики Н. Бурбаки [**5**, глава 1] кванторы ∀ и ∃ не являются собственными понятиями, а определяются через термы вида $\tau_x P(x)$ (с неформальным смыслом «такой x, что верно $P(x)$»), которые в свою очередь раскрываются в конструкции без буквы x. Не опускаясь на уровень τ-термов, покажем бурбаковский способ исключения переменных, как если бы он был применен непосредственно к кванторам.

Кодируя кванторы в стиле Н. Бурбаки, заменим вхождения связанных переменные на знаки и проведем линии к соответствующим

кванторам ∀ и ∃, изображая их *связь*. Свободные переменные останутся буквами. Например,

$\forall x\, \exists y\, P\big(x, y, f(x, z)\big)$ кодируется как  .

Что в этом подходе поучительного? Несомненно, цель была сделать базовый уровень языка математики как можно более простым. Тем не менее, они сделали его преставление не линейным, а «графовым»: код на базовом языке — это не линейная последовательность знаков, а граф — далеко не тривиальное понятие само по себе. Такие решения случайными не бывают.

Бурбаки переложили сюда сложность из другого места. Из какого? Из операции подстановки. Теперь подстановка любого терма вместо любой связанной переменной определяется без каких-либо условий типа «терм свободен для» и проверок на совпадение переменных, борясь с «захватами» свободных переменных связанными.

Более того, такое представление отражает нашу интуицию, о которой говорилось выше: переменная, связанная квантором, — это совсем другая сущность, чем свободная переменная; каждая связанная переменная отличается от всех других переменных в мире, поскольку поддерживает связь с уникальным вхождением квантора в математический текст. Можно предположить, что именно это было решающим аргументом, почему Н. Бурбаки переложили сложность на этот уровень.

Вернемся к программированию. Как мы закодируем бурбаковское графовое представление на каком-либо привычном объектно-ориентированном языке программирования? Самое простое решение — в представлении каждого вхождения связанной переменной сохраним ссылку на объект, представляющий квантор, точнее, на узел дерева абстрактного синтаксиса, соответствующий квантору. Так оно обычно и делается в компиляторах после фазы разрешения имен: каждое вхождение идентификатора несут в себе ссылку на его декларацию.

Однако заметим, что это решение не реализуемо в чисто функциональных языках, имея в виду, что реализация должна иметь ту же сложность $O(1)$ для скачков от узла к узлу по дугам, что и в объектно-ориентированных языках, и как это интуитивно ожидаются. Функциональные языки имеют принципиальный недостаток: они не предоставляют средств для эффективного представления графов общего вида, максимум — деревья. Представляя граф, можем разве что завести списки вершин и дуг, а для этого надо присваивать вершинам, а может и дугам, уникальные идентификаторы — еще одно усложнение. Обращение к таким спискам имеет сложность больше, чем $O(1)$. Принципиально более простых решений нет.

В математике понятие графа именно так и определяется. Апологеты чисто функциональных языков любят их именно за это — за тесное

соответствие классическим разделам математики с теорией множеств в качестве основы. А объектно-ориентированные языки из этой математической картины выпадают, хотя именно они предоставляют эффективные средства для адекватных представлений структур данных, включая графы общего вида. Это указывает на недорешенные проблемы в основаниях семантик языков программирования.

Возвращаясь к графовому представлению базового языка Н. Бурбаки, видим занятную картину. В основание кладется нетривиальное понятие графа, считающееся в данном случае интуитивно понятным. А после построения теории множеств, оказывается, что ее понятий не хватает для адекватного самоописания. (Отметим, что полнота/неполнота теорий к этому вопросу отношения не имеет. В рассуждениях о не/полноте допускается кодирование одной теории в другой любой сложности; важна лишь принципиальная возможность отображения. А мы под «адекватностью» подразумеваем сохранение сложности.)

В программировании эти вопросы всплывают, когда мы занимаемся анализом и верификацией программ, преобразованиями, сохраняющими их семантику. Для этого семантика должна быть формально определена, да причем так, чтобы было удобно и эффективно для реализации в системах верификации, *proof assistants*. Простого математического созерцания бурбаковского представления со словами: «интуитивно ведь понятно, что это работает» — недостаточно.

4. Работы по номинальным методам

К счастью, 30 лет назад математик Andrew Pitts не только увидел эти проблемы (он был не один такой, см., например, [**6**][**10**], а из более поздних работ, указывающих на проблемы [**16**, сноска 2 и раздел 6.2]), но и начал их решать на основе своего математического опыта по теории категорий и основаниям математики [**15**]. За прошедшее время вместе с учениками он построил богатую теорию имен — теорию *номинальных множеств*, *nominal set theory* [**17**][**14**][**12**][**12**], в более широком контексте также называемую *nominal methods*, *nominal techniques* [**13**], да не в одном варианте — в виде моделей в теории множеств и в теории категорий.

В 2019 году Andrew Pitts и Murdoch Gabbay получили премию ACM имени Алонзо Черча за *their ground-breaking work introducing the theory of nominal representations* [**1**]. Тем самым, их достижения были высоко оценены экспертами в логике и теории вычислений, а теперь дело за тем, что они вошли в широкую практику методов верификации программ и семантики языков программирования. Они должны лечь в основу теории ссылок и объектов в объектно-ориентированных языках.

5. Функциональный язык с генератором ссылок и его свойства

Чтобы конкретно представить предмет исследования и увидеть источники проблем, рассмотрим чисто функциональный язык, расширенный генератором имен, ссылок. В работах группы A. Pitts'a такой язык назван FreshML [**14**][**12**] по имени оператора генерации имен ***fresh***. Здесь мы будем использовать более привычное объектно-ориентированным программистам название такого оператора — $\boldsymbol{new}$.

Понятие объекта — составное, его можно разложить на 2 части — понятие ссылки и понятие состояния объекта, причем о ссылках можно говорить отдельно от состояний объектов, но не наоборот. Более того, чтобы говорить об изменяющихся состояниях, надо еще ввести понятие времени вычислений. А на уровне разговора о ссылках понятие времени не требуется. Хотя мы будем говорить об «актах вычисления», все результаты не зависят от порядка вычислений, то есть понятие времени еще не будет играть существенной роли. Сейчас мы забудем, что объекты имеют состояния, изменяющиеся во времени. Займемся лишь семантикой имен и ссылок как таковых, «в чистом виде».

Возьмем любой чисто функциональный язык и расширим его оператором $\boldsymbol{new}$ с такой операционной семантикой: каждый акт вычисления $\boldsymbol{new}$ порождает атомарное данное, единственное свойство которого — быть *равным самому себе* (*рефлексивность* равенства) и *быть неравным всему остальному*: всем значениям в областях данных языка и всем данным, порожденным другими вызовами операторов $\boldsymbol{new}$. Будем называть данные, порождаемые оператором $\boldsymbol{new}$, *ссылками*, имея в виду дальнейшую цель (за рамками данной работы) — формализация понятия объекта. В работах группы A. Pitts'a используются термины «*имена*» и «*атомы*». Они прикладывают свою теорию к описанию формальных языков с именами — локальными (связанными) и глобальными (свободными), — языка математика и расширенного функционального языка, а объектно-ориентированными языками не занимаются.

Рассмотрим примеры. Будем использовать знак $\coloneqq$ в нашем функциональном языке в операторе $\boldsymbol{let}$ для введения локального имени с однократными вычислением значения правой части и для определения функций, а также в метаязыке в качества знака «равно по определению». В остальном будем использовать привычную математическую нотацию, надеясь, что она понятна без долгих объяснений.

Следующие выражения в функциональном языке с оператором $\boldsymbol{new}$ вычисляются до True:

$$\boldsymbol{new} \neq \boldsymbol{new}$$

$$\boldsymbol{let}\ x \coloneqq \boldsymbol{new},\ y \coloneqq \boldsymbol{new}\ \boldsymbol{in}\ x \neq y$$

$$\boldsymbol{let}\ x \coloneqq \boldsymbol{new},\ y \coloneqq x \quad \boldsymbol{in}\ x = y$$

Видим, что повторные вычисления оператора $\boldsymbol{new}$ порождают неравные ссылки, а копирование сохраняет равенство.

Еще пример. Следующая функция f вырабатывает пару из заведомо неравных элементов, поскольку при каждом вычислении $f(x)$ первым элементом пары будет новая ссылка, по определению неравная никаким другим значениям:

$$\mathrm{f}(x) := \langle \boldsymbol{new}, x \rangle$$

Повторное вычисление $f(x)$ всегда дает неравные результаты:

$$\forall x \; f(x) \neq f(x)$$

6. Референциальная прозрачность и контекстуальная эквивалентность

Приведенные примеры указывают на первую (и главную) проблему функционального языка с оператором $\boldsymbol{new}$: *нарушается референциальная прозрачность*. (Мы не будем говорить «ссылочная прозрачность», чтобы не путалось с понятием ссылки).

Референциальная прозрачность — это свойство термов (выражений) данного языка, заключающееся в том, что замена вхождений терма на один раз вычисленное его значение является эквивалентным преобразованием. В частности, для референциально прозрачного языка выполняются такие свойства для всех термов t и функций f, определенных на языке:

$$t = t$$

$$x = y \implies f(x) = f(y)$$

Референциальная прозрачность настолько глубоко встроена в язык математики, математическую логику, что, кажется, трудно подступиться к построению теории языка, в котором она не выполняется, при этом не свалиться в операционную семантику языков программирования с понятием времени вычислений. В частности, денотационная семантика чисто функциональных языков, отображающая определения функций на языке в функции теории множеств, существенно опирается на референциальную прозрачность.

Тем не менее, как показали работы группы A. Pitts'a, такая теория возможна, интересна и плодотворна. Этот успех основан на следующем наблюдении: значения и термы в языке с оператором $\boldsymbol{new}$ обладают *контекстуальной эквивалентностью* (*contextual equivalence, observational equivalence, Leibniz equivalence*).

Два значения x и y некоторого типа D являются *контекстуально эквивалентными* (обозначим $x \approx y$), если они не различимы любым предикатом c, то есть функцией типа $D \to Bool$, определенном на данном языке L:

$$x \approx y \;\coloneqq\; (\forall c \in L)\; c(x) = c(y)$$

Теперь рассмотрим *модифицированную референциальную прозрачность*, в определении которой равенство значений, присущее языку, *заменено на контекстуальную эквивалентность*. Оказывается, что функциональный язык с оператором $\boldsymbol{new}$ обладает этим свойством: для всех термов t, функций f и значений x и y:

$$t = t$$

$$x = y \implies f(x) = f(y)$$

и более того:

$$x \approx y \implies f(x) \approx f(y)$$

Отметим, что контекстуальную эквивалентность невозможно проверить внутри языка, нет такого вычислимого отношения как $\approx$, в отличие от обычного равенства значений $=$. Контекстуальная эквивалентность является понятием теории, математического метаязыка.

7. Нарушение и сохранение свойств функционального языка

У функционального языка с генератором ссылок $\boldsymbol{new}$ сохраняются многие хорошие свойства чисто функциональных языков, когда равенство $=$ заменяется на эквивалентность $\approx$. Конечно, формулируя эквивалентные преобразования программ, надо следить, какое из этих отношений играет в них роль.

Например, перестает быть эквивалентным *выделение одинаковых подвыражений* (здесь знак $\Longrightarrow$ означает «эквивалентно преобразуется к», а не «влечет», как выше):

$$\ldots t \ldots t \ldots \implies \boldsymbol{let}\; x \coloneqq t\; \boldsymbol{in}\; \ldots x \ldots x \ldots$$

Такими преобразованиями можно продолжать пользоваться, накладывая ограничения, говорящие о том, что данный фрагмент кода не выходит за рамки чисто функционального подмножества или выходит «не очень сильно». Пример первого случая для статически типизированного языка: определение типа терма t не использует ссылочного типа; пример второго: при вычислении t не порождаются новые ссылки (скажем, если в определяющем его коде нет операторов $\boldsymbol{new}$) или они не переходят в его значение (что можно определять статическим анализом зависимостей данных).

Нарушение эквивалентности выделения одинаковых подвыражений (подтермов) указывает на то, что теперь *не годится* стандартное построение *денотационной семантики* для функциональных языков, которая присваивает каждому терму языка его *денотат* — значение или функцию от значений свободных переменных, которое можно

использовать в качестве результата вычисления всех вхождений текстуально совпадающих термов.

Тем не менее, сохраняются следующие свойства, хотя и в модифицированном виде:

- Можно сконструировать понятие *денотата* терма, которое хоть и не будет готовым значением, но его можно использовать вместо вычисления копий терма со сложностью не более чем копирования денотата. Это следует из работ группы A. Pitts'a в части теоретико-множественных моделей номинальных множеств и т.н. ν-*исчисления* (ν-*calculus*) [**17**]. Идея такого денотата представлена в [**8**].
- На основе этого можно дать денотационную семантику всего языка, сопоставляющую определениям функций множества пар определенного вида, описывающих связь подмножеств аргументов с денотатами значений. Идея этого построения также показана в [**8**].
- Благодаря наличию такой модифицированной денотационной семантики, в реализации расширенного функционального языка остается возможность *мемоизации* или *табуляции*: при каждом обращении к данной функции пара аргумент-значение заносится в специальную таблицу и готовое значение используется при следующих вызовах с аргументом, в некотором смысле похожим на предыдущий. Такой механизм не реализуют для функциональных языков в общем виде, так как для него требуются лишние затраты по времени и большие расходы памяти, но он оправдан для отдельных функций под контролем программиста в тех случаях, когда он может дать выигрыш по эффективности по смыслу задачи.
- И последнее, но практически, быть может, самое важное — сохраняется *детерминированность* параллельных вычислений, присущая чисто функциональным языкам, то есть контекстуально эквивалентны результаты, полученные при любом порядке последовательных или параллельных вычислений вызовов функций. Этот механизм тоже далеко не всегда выгоден в общем виде из-за чрезмерно мелких гранул параллелизма, но может значительно повышать эффективность под контролем программиста, указывающим в программе, какие вызовы выгодно вычислять параллельно.

8. Что сделано и что дальше?

Andrew Pitts, Ion Stark, Murdoch Gabbay и другие [**12**][**13**][**14**][**15**][**17**] разработали теории чисто функциональных языков с генератором имен и ссылок на базе теории множеств, теории категорий, а также на основе операционного подхода, и применили их для построения формальной

семантики локальных имен в формальных языках и их эквивалентных преобразований — более удобных и богатых теорий, чем классические рассуждения об областях видимости и т.п. Также ими были разработаны пакеты для систем автоматизации построения и проверки доказательств (*proof assistants*) Coq и Agda, для построения доказательств в терминах номинальных методов (*nominal methods*).

Следующий шаг, который еще предстоит сделать, — построение полной теории объектов в объектно-ориентированных языках с использованием адекватной теории ссылок. Существующие теории неполны, поскольку оставляют вопросы точной семантики ссылок на наивные или полуформальные рассуждения. Показательна ремарка в [**16**, сноска 2]: «Unfortunately, our theoretical understanding of references is not well developed», — со ссылкой на [**6**][**17**] как на шаги в правильном направлении. Насколько известно автору данной статьи, с тех пор в области объектно-ориентированных языков еще нет существенного продвижения, но должно произойти после того, как премия ACM [**1**] создала рекламу достижениям группы A. Pitts'a.

На этом пути теоретически интересный и практически важный вопрос — построение моделей вычислений, промежуточных между функциональными языками с генератором ссылок и объектно-ориентированными языками общего вида. Они могут строиться, либо со стороны функциональных языков их расширением с сохранением выбранных свойств, либо со стороны объектно-ориентированных языков путем наложения ограничений на языковые конструкции так, чтобы выполнялись какие-то свойства.

Здесь основная задача: максимально продвинуться по этой шкале с сохранением референциальной прозрачности на основе контекстуальной эквивалентности как основополагающего свойства, из которого вытекают другие, хоть и в модифицированном виде по сравнению с классической референциальной прозрачностью. В частности, детерминированность параллельных вычислений правильнее определять через более широкое понятие контекстуальной эквивалентности, а не встроенного в языки отношения равенства, так как содержательный смысл имеет именно неотличимость результатов, полученных при разных порядках вычислениях, а не их равенство.

Сейчас просматриваются следующие круги, этапы ослабления ограничений и добавления понятий:

- За основу берем чисто функциональный язык с генератором ссылок, но без понятия объекта с состоянием. Этот уровень представлен выше.
- Добавляем понятие *объектов с неизменными состояниями* (*immutable objects*). С виду практическая польза от этого небольшая,

хотя есть: ссылочное равенство более эффективно, проверяется значительно быстрее, чем сравнение содержимого (конечно, при условии, что оно подходит по смыслу задачи). Однако есть и фундаментальное, теоретическое и практическое значение равенства по ссылке: оно может быть введено для *всех* данных, включая те, которые ранее не имели разумного отношения равенства по своему смыслу. Например, такими являются функциональные значения, вырабатываемые лямбда-выражениям, — *замыкания*. В некоторых языках (например, в Haskell'е) запрещают сравнивать замыкания, мотивируя это тем, что равенство функций алгоритмически неразрешимо. Однако их можно считать эквивалентными, если они являются копиями результата одного и того же акта вычисления лямбда-выражения — аналогично тому, как равные ссылки восходят к одному акту вычисления оператора ***new***. Особая роль равенства по ссылке неслучайна: оно является *инициальным* в смысле теории категорий, то есть самым сильным отношением, которое влечет любое другое отношение эквивалентности.

- Разрешаем *изменения состояний* объектов, но только такие, которые с точки зрения «наблюдателя» (то есть программного кода на данном языке) незаметны, то есть при любом порядке вычислений и изменений состояний порождаются *контекстуально эквивалентные результаты*. Открытая проблема: можно ли теоретически охарактеризовать такие классы объектов, определенные в объектно-ориентированном языке, которые сохраняют контекстуальную эквивалентность для всех (под)выражений на языке? Общие ответы на эти и подобные вопросы, по-видимому, отрицательны, так как мы здесь сталкиваемся с алгоритмической неразрешимостью. Но как обычно делается в математике, можно выделять и изучать интересные частные случаи, для которых можно сказать что-то содержательное.
- Таким богатым частным случаем оказываются объекты, которые можно описать, как изменяющиеся *монотонно* на некоторой (*полу*)*решетке*. Оказывается, что модифицированная референциальная прозрачность на основе контекстуальной эквивалентности сохраняется при определенной дисциплине операций над такими *монотонными объектами*. Это вопрос исследуется в работах [**9**][**2**][**3**][**4**]. Открытая проблема: научиться описывать соответствующие (полу)решетки по декларациям классов на объектно-ориентированном языке или в каком-то общем виде.
- В терминах *монотонных объектов* надо научиться *решать практические классы задач*, которые не реализуются на функциональных языках столь же эффективно, как на объектно-

ориентированных. В первую очередь надо преодолеть главный недостаток чисто функциональных языков, заключающийся в том, что на них можно обрабатывать только деревья, а графы общего вида приходится кодировать менее эффективными структурами данных по сравнению с объектно-ориентированными языками, где вершины естественно представляются объектами, а движение от вершины к вершине по дуге является дешевой атомарной операцией «переход по ссылке». Оказывается, эта задача вполне разрешима [**4**].

- Далее надо извлечь максимальную пользу из моделей вычислений, двигающихся в сторону объектно-ориентированных, но сохраняющих ценные свойства функциональных. *Анализ, преобразования, верификация, распараллеливание программ* и вообще *метавычисления* — все эти виды деятельности над программами более результативны для языков с отсутствующим или ограниченными побочными эффектами. Это хорошо видно на больших или меньших успехах специализации программ , таких как *частичные вычисления* и *суперкомпиляция*. В частности, как показано в работе [**К91**], механизм табуляции (мемоизации) можно использовать для реализации *динамических частичных вычислений.*

Автор данной статьи участвует в работах по построению такой модели вычислений, промежуточной между функциональной и объектно-ориентированной, и Java-подобного языка, удовлетворяющим указанными выше свойствам с практической целью реализации системы детерминированного параллельного программирования, которая освободит программиста от мук отладки недетерминированных параллельных программ на достаточно широком классе задач [**2**][**3**].

9. Заключение

Мы рассмотрели старую проблему адекватной формализации локальных имен в формальных языках — в языке математики и в языках программирования, и проблему формализации семантики ссылок в объектно-ориентированных языках. Рассматривая историю вопроса, мотивировки и решения, мы пришли к выводу, что это одна и та же проблема, а кажущаяся разница происходит от того, что эти понятия относятся к разным видам, временам, периодам деятельности, которые в программировании имеют устоявшиеся названия: с именами в формальных текстах мы имеем дело в «статическом времени» (*static time*), когда математический текст или программа пишется и обрабатывается, например, компилятором; а работа со ссылками происходит в период исполнения (*run time*) программы на компьютере или математического утверждения в некоем идеальном мире. Эти периоды и виды деятельности совмещаются при механической обработке математических текстов на

компьютере, системах компьютерной алгебре, логике и метаматематике, и в системах глубокого анализа и преобразования программ, метавычислениях. Тогда и проявляется нехватка адекватной теории имен и ссылок.

Без углубления в детали было отмечено серьезное продвижение по построению таких теорий за последние десятилетия в работах группы Andrew Pitts'a [**12**][**12**][**13**][**14**][**17**], недавно отмеченные премией ACM [**1**]. Они построили модели *номинальных множеств* (*nominal sets*), то есть областей данных с именами или ссылками, в рамках теории множеств, дали их формализацию в теории категорий, а также предложили чисто функциональный язык, расширенный именами как данными, названный FreshML [**14**][**12**], и исследовали его семантику в рамках своих теорий. Можно быть уверенным, что эти результаты послужат теоретическими основаниями построения полной формальной семантики объектно-ориентированных языков и разработки методов верификации программ с изменяющимися объектами и ссылками на них.

Было продемонстрировано фундаментальное понятие, на котором базируются успехи построения этих теорий и расчет на их плодотворное будущее развитие вплоть до изменяющихся объектов, — *модифицированная референциальная прозрачность*, в которой вместо обычного равенства значений используется *контекстуальная эквивалентность*. В частности, для определения детерминированности параллельных вычислений по существу требуется вовсе не равенство, а контекстуальная эквивалентность результатов при разных порядках вычислениях. Опираясь на такую ослабленную референциальной прозрачность, получаем возможность расширять чисто функциональные языки ссылками и затем (ограниченно) изменяющимися объектами с сохранением многих ценных свойств функциональных языков.

В последнем разделе приведены возможные круги расширения чисто функциональных языков сначала ссылками в чистом виде без объектов, как в работах группы A. Pitts'a, а затем на изменяющиеся объекты при определенных ограничениях, гарантирующих сохранение модифицированной референциальной прозрачности и вытекающих из нее свойств. Часть этих работ уже ведется с определенными результатами, другая часть — открытые проблемы на будущее. Самое ценное, что здесь получается в конечном развитии, — появление модели вычислений и языков между функциональными и объектно-ориентированными. В этой модели и языках расширяется круг эффективно и адекватно решаемых задач по сравнению с чисто функциональными языками, но с сохранением ценных свойств, включая детерминированные параллельные вычисления. Среди этих задач важнейшая — обработка графов, в то время как в чисто функциональных языках эффективные области данных — это максимум деревья.

Литература

1 ACM Special Interest Group on Logic and Computation. Winners of the 2019 Alonzo Church Award. — 2019. — URL: https://siglog.org/winners-of-the-2019-alonzo-church-award

2 Адамович А.И., Климов Анд.В. Как создавать параллельные программы, детерминированные по построению? Постановка проблемы и обзор работ // Программные системы: теория и приложения. — 2017. — Т. 8. — № 4 (35). — С. 221–244. — DOI: 10.25209/2079-3316-2017-8-4-221-244

3 Адамович А.И., Климов Анд.В. Подход к построению системы детерминированного параллельного программирования на основе монотонных объектов // Вестник СибГУТИ. — 2019. — № 3. — С. 14–26. — URL: http://vestnik.sibsutis.ru/uploads/1570089084_1278.pdf

4 Adamovich A.I., Klimov And.V. Building Cyclic Data in a Functional-Like Language Extended with Monotonic Objects // X Workshop PSSV: Program Semantics, Specification and Verification: Theory and Applications (Novosibirsk, Akademgorodok, Russia, July 1–2, 2019) : Abstracts. — Novosibirsk : A.P. Ershov Institute of Informatics Systems, 2019. — P. 11–19. — URL: https://persons.iis.nsk.su/files/persons/pages/tezisy_seminara_pssv.pdf

5 Бурбаки Н. Теория множеств / пер. с фр. — М. : Мир, 1965.

6 Hoare C.A.R. Record Handling // ALGOL Bulletin. — 1965. — Issue 21. — P. 39–69. — DOI: https://dl.acm.org/doi/10.5555/1061032.1061041

7 Klimov And.V. Dynamic specialization in extended functional language with monotone objects // ACM SIGPLAN Notices. — 1991. — Vol. 26, No. 9. — P. 199–210. — DOI: 10.1145/115865.376287

8 Klimov And.V. On Semantics of Names in Formulas and References in Object-Oriented Languages // Computer algebra: 4th International Conference Materials. Moscow, June 28–29, 2021 / Ed. S.A. Abramov, L.A. Sevastyanov. — M. : MAKS Press, 2021. — P. 73–76. — DOI: 10.29003/m2019.978-5-317-06623-9

9 Kuper L., Newton R.R. LVars: lattice-based data structures for deterministic parallelism // Proceedings of the 2nd ACM SIGPLAN workshop on Functional high-performance computing (FHPC '13). Association for Computing Machinery, New York, NY, USA. — P. 71–84. DOI: 10.1145/2502323.2502326

10 MacLennan B.J. Values and objects in programming languages // SIGPLAN Not. — 1982. — Vol. 17, No. 12. — P. 70–79. — DOI: 10.1145/988164.988172

11 Мендельсон Э. Введение в математическую логику / пер. с англ. — М. : Наука, 1976.
Mendelson E. Vvedenie v matematichesluyu logiku = Introduction to Mathematical Logic / per. s angl. — M. : Nauka, 1976.

12 Pitts A.M. Nominal Logic, a First Order Theory of Names and Binding. Information and Computation. — 2003. — Vol. 186, No. 2. — P. 165–193. — DOI: 10.1016/S0890-5401(03)00138-X
12 Pitts A.M. Nominal Sets: Names and Symmetry in Computer Science // Cambridge Tracts in Theoretical Computer Science. — 2013. — Vol. 57. — DOI: 10.1017/CBO9781139084673
13 Pitts A.M. Nominal techniques // ACM SIGLOG News 3, 1 (January 2016). — P. 57–72. — DOI: 10.1145/2893582.2893594
14 Pitts A.M., Gabbay M.J. A Metalanguage for Programming with Bound Names Modulo Renaming // Lecture Notes in Computer Science. — 2000. — Vol. 1837. — P. 230-255. — DOI: 10.1007/10722010_15
15 Pitts A.M., Stark I.D.B. Observable properties of higher order functions that dynamically create local names, or: What's new? // Mathematical Foundations of Computer Science 1993. MFCS 1993. Lecture Notes in Computer Science. — Vol 711. — Berlin, Heidelberg : Springer, 1993. — DOI: 10.1007/3-540-57182-5_8
16 Reddy U.S. Objects and Classes in Algol-Like Languages // Information and Computation. — 2002. — Volume 172, Issue 1. — P. 63–97. — DOI: 10.1006/inco.2001.2927
17 Stark I.D.B. Categorical Models for Local Names // Lisp and Symbolic Computation. — 1996. — Vol. 9, No. 1. — P. 77-107. — DOI: 10.1007/BF01806033

References

1. ACM Special Interest Group on Logic and Computation. Winners of the 2019 Alonzo Church Award. — 2019. — URL: https://siglog.org/winners-of-the-2019-alonzo-church-award
2 Adamovich A.I., Klimov And.V. How to Create Deterministic by Construction Parallel Programs? Problem Statement and Survey of Related Works // Program Systems: Theory and Applications. — 2017. — Vol. 8, No. 4 (35). — P. 221–244. — DOI: 10.25209/2079-3316-2017-8-4-221-244
3 Adamovich A.I., Klimov And.V. An approach to deterministic parallel programming system construction based on monotonic objects // Vestkik SibGUTI. — 2019. — № 3. — С. 14–26. — URL: http://vestnik.sibsutis.ru/uploads/1570089084_1278.pdf
4 Adamovich A.I., Klimov And.V. Building Cyclic Data in a Functional-Like Language Extended with Monotonic Objects // X Workshop PSSV: Program Semantics, Specification and Verification: Theory and Applications (Novosibirsk, Akademgorodok, Russia, July 1–2, 2019) : Abstracts. — Novosibirsk : A.P. Ershov Institute of Informatics Systems, 2019. — P. 11–19. — URL: https://persons.iis.nsk.su/files/persons/pages/tezisy_seminara_pssv.pdf
5 Bourbaki N. Theory of Sets / per. s fr. — M. : Mir, 1965.

6 Hoare C.A.R. Record Handling // ALGOL Bulletin. — 1965. — Issue 21. — P. 39–69. — DOI: https://dl.acm.org/doi/10.5555/1061032.1061041
7 Klimov And.V. Dynamic specialization in extended functional language with monotone objects // ACM SIGPLAN Notices. — 1991. — Vol. 26, No. 9. — P. 199–210. — DOI: 10.1145/115865.376287
8 Klimov And.V. On Semantics of Names in Formulas and References in Object-Oriented Languages // Computer algebra: 4th International Conference Materials. Moscow, June 28–29, 2021 / Ed. S.A. Abramov, L.A. Sevastyanov. — M. : MAKS Press, 2021. — P. 73–76. — DOI: 10.29003/m2019.978-5-317-06623-9
9 Kuper L., Newton R.R. LVars: lattice-based data structures for deterministic parallelism // Proceedings of the 2nd ACM SIGPLAN workshop on Functional high-performance computing (FHPC '13). Association for Computing Machinery, New York, NY, USA. — P. 71–84. DOI: 10.1145/2502323.2502326
10 MacLennan B.J. Values and objects in programming languages // SIGPLAN Not. — 1982. — Vol. 17, No. 12. — P. 70–79. — DOI: 10.1145/988164.988172
11 Mendelson E. Vvedenie v matematichesluyu logiku = Introduction to Mathematical Logic / per. s angl. — M. : Nauka, 1976.
12 Pitts A.M. Nominal Logic, a First Order Theory of Names and Binding. Information and Computation. — 2003. — Vol. 186, No. 2. — P. 165–193. — DOI: 10.1016/S0890-5401(03)00138-X
12 Pitts A.M. Nominal Sets: Names and Symmetry in Computer Science // Cambridge Tracts in Theoretical Computer Science. — 2013. — Vol. 57. — DOI: 10.1017/CBO9781139084673
13 Pitts A.M. Nominal techniques // ACM SIGLOG News 3, 1 (January 2016). — P. 57–72. — DOI: 10.1145/2893582.2893594
14 Pitts A.M., Gabbay M.J. A Metalanguage for Programming with Bound Names Modulo Renaming // Lecture Notes in Computer Science. — 2000. — Vol. 1837. — P. 230-255. — DOI: 10.1007/10722010_15
15 Pitts A.M., Stark I.D.B. Observable properties of higher order functions that dynamically create local names, or: What's new? // Mathematical Foundations of Computer Science 1993. MFCS 1993. Lecture Notes in Computer Science. — Vol 711. — Berlin, Heidelberg : Springer, 1993. — DOI: 10.1007/3-540-57182-5_8
16 Reddy U.S. Objects and Classes in Algol-Like Languages // Information and Computation. — 2002. — Volume 172, Issue 1. — P. 63–97. — DOI: 10.1006/inco.2001.2927
17 Stark I.D.B. Categorical Models for Local Names // Lisp and Symbolic Computation. — 1996. — Vol. 9, No. 1. — P. 77-107. — DOI: 10.1007/BF01806033