В поисках математических оснований языков программирования:

Адекватна ли теория множеств для формализации имен, ссылок и объектно-ориентированного программирования?

Андрей В. Климов

Институт прикладной математики им. М.В. Келдыша РАН
г. Москва

План

Введение

3. О чем речь? В математике – о проблеме имен и ссылок
4. О чем речь? В программировании – о значениях и ссылках на объекты
5. Классы языков программирования

Основания языков программирования – роль теории множеств

6. Какие есть способы формального описания сущностей языков программирования?
7. Зачем «простому программисту» теория категорий?
8. Зачем уходить из теории множеств? Поначалу всё хорошо
9. Что в объектно-ориентированных языках?
10. Надо уходить из теории множеств: споткнулись на именах, ссылках на объекты

Эквивалентность по Лейбницу расширяет возможности теории

11. От равенства к эквивалентности по Лейбницу

12.13. Лейбниц-прозрачность и денотационная семантика со ссылками

14. Лейбниц-прозрачность изменяющихся объектов

Программистские задачи: между функциональной и объектно-ориентированной моделями

не успели на семинаре

15. Программистская цель: вот зачем городим огород
16. Монотонные объекты и классы
17. Иерархия и объемы понятий
18. Практическая проблема и тестовая задача: обработка графов
19. Резюме по модели вычислений и обработке графов на ФЯ
20. Публикации по программированию с монотонными объектами

О чем речь?
В математике – о проблеме имен и ссылок

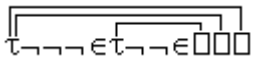
- **В мире формул есть «проблема» связанных переменных, локальных имен**
  - $\forall x.P(x)$ и $\exists y.P(x)$ – «наши родные» кванторы всеобщности и существования
  - $\tau_x P(x)$ и $\epsilon x.P(x)$ – τ–символ Бурбаки и ϵ–символ Гильберта
  - $\lambda x.T(x)$ – лямбда-термы
    - $(\lambda x.\lambda y.xy)y \not\Rightarrow \lambda y.yy$

- **Проблема проявляет себя в «лишних» фразах и оговорках в математических рассуждениях и формальных выводах**
  - «с точностью до альфа-эквивалентности»
  - «класс эквивалентности при всевозможных переименованиях переменных»
  - «терм, свободный для переменной x»
  - в правилах вывода условия вида: «x – свежая переменная», «x fresh», «x new»

- **Бурбаки в томе 1 «Теория множеств» «решили» проблему, введя «графовый» синтаксис**
  - терм следующего вида

    $\tau_x \ \ldots \ x \ \ldots \ x \ \ldots$
  - является «человеческой» записью следующей конструкции

  - например:  $= \tau_x \neg\neg\neg\left(\left(\tau_y \neg\neg(y \in x)\right) \in x\right) \approx \tau_x\left(\tau_y(y \in x) \notin x\right) = \emptyset$

О чем речь?
В программировании – о значениях и ссылках на объекты

- **Значения**
  - обладают равенством «такой же»
  - описываются классической математикой, теорией множеств
  - значения не изменяются в каком-либо времени, существуют в «статическом» мире
  - термы, вырабатывающие значения, обладают «ссылочной прозрачностью», «referential transparency»:
    - термы и принимаемые ими значения взаимозаменяемы
    - вместо терма можно подставить его значение (точнее, константу его изображающую)

- **Имена, ссылки на объекты**
  - обладают равенством «тот же самый»
  - семантика в терминах теории множеств дается «с трюками», неадекватна
  - объекты, на которые указывают ссылки, могут «жить» во времени и изменяться
    - отношение «тот же самый» на ссылках не зависит от состояния объектов
  - терм **new**, порождающий новые ссылки на новые объекты, не обладает ссылочной прозрачностью

$$\textbf{new} \neq \textbf{new}$$

$$\textbf{let}\ \ x \coloneqq \textbf{new};\ y \coloneqq \textbf{new}\ \ \textbf{in}\ \ x \neq y$$

- **Статья с говорящим названием о связи отношения «тот же самый» с изменениями объекта**
  - Henry G. Baker. 1993.
    Equal rights for functional objects or, the more things change, the more they are the same.
    SIGPLAN OOPS Mess. 4, 4 (Oct. 1993), 2–27. DOI: https://doi.org/10.1145/165593.165596

Классы языков программирования

- **Чисто-функциональные языки**
  - данные – только значения
  - результаты вычислений, семантика описываются без понятия времени и изменений
    - чистый Lisp – без операций изменения списков и прочих побочных эффектов
    - чистый Рефал – без операций изменения чего-либо и ввода-вывода
    - языки семейства ML – их чисто-функциональные подмножества
    - Haskell – единственный практический чисто-функциональный язык
    - из многих современных языков можно выделить чисто-функциональное подмножество
- **Императивные языки**
  - языки типа Algol-60 – есть изменяемые массивы и глобальные переменные
  - идеологически лежат между чисто-функциональными и объектно-ориентированными
- **Объектно-ориентированные языки**
  - данные включают объекты, идентифицируемые ссылками, с изменяющимися состояниями
    - Lisp с равенством EQ и изменяемыми списками
  - другие «требования» к объектно-ориентированным языкам нас сегодня не интересуют
    - Simula-67 – первый объектно-ориентированный язык, гениальное изобретение
    - Smalltalk – первый популярный объектно-ориентированный язык
    - Java, C#, Python...
- Во всех классах языков нас интересуют языки с семантикой, определяемой без апелляции к фон-неймановской памяти компьютера
  - Pascal, C/C++ и т.п. – «плохие» в этом смысле
  - Algol-68 – «хороший», но не дотянул до объектно-ориентированного

Какие есть способы формального описания сущностей языков программирования?

Какие виды формальной семантики и логики языков программирования наиболее используются в жизни computer science?

доминирующие сейчас

- **Теория-множеств (ТМ)**
  - денотационная семантика использует ТМ для описания областей данных и приписывает конструкциям языка области данных и функций в терминах ТМ
  - Dana Scott [1971] проинтерпретировал лямбда-исчисление в ТМ: построил область $D = D \to D$
- **Аксиоматико-дедуктивные системы**
  - разнообразные логики (включая модальные, временные, интуиционистские и т.п.)
  - корректность типизации (и других свойств) программ описывается правилами вывода
  - операционная семантика в виде правил вывода, Natural Semantics: вывод ≈ вычисления

наступающие

- **Теория зависимых типов**
  - свойства программ и доказательства в терминах типов и программ (соответствие Карри-Ховарда)
  - типизация будущих языков (а пока для теоретиков: Coq, Agda, Idris)
- **Теория категорий (ТК)**
  - многие математики рассматривают ТК как альтернативу ТМ в основаниях математики
  - в близком соответствии с теорией зависимых типов

- **Экзотика на будущее?**
  - Конструктивные направления в математике
  - Интуиционизм Брауэра (не путать с интуиционистскими логиками)
  - Кибернетические основания математики Валентина Турчина

Зачем «простому программисту» теория категорий?

- **Страшно отстать от мировой науки**
  - Теория категорий используется как рабочий инструмент во всё большем числе статей
  - Хочется хотя бы понимать, о чем они пишут
  - (Вспомнить байку про Келдыша и Штаркмана)
- **Теория категорий формально проста (математика на уровне мат-школьника)**
  - Но говорит она о математических явлениях и структурах многое
  - Из-за непривычки к новым понятиям трудна для восприятия и использования
  - (Вспомнить байку про Колмогорова и теорию категорий)
  - Это теория базовых математических шаблонов – этим плодотворна
  - Активно использует «метасистемные переходы» [В.Ф.Турчин, «Феномен науки»] – этим трудна
    - самоприменение шаблонов
    - лестницы метасистемных переходов
    - разрастание предпоследнего уровня (в удивительном многообразии применений понятий)
- **Всё более осознаваемые потребности в Computer Science, незавершенность ее оснований**
  - Теории множеств не хватает (можно, но трудно)
  - Пример: ссылки в объектно-ориентированных языках и имена сами по себе
- **Нынешнее состояние Computer Science позволяет «стоять на плечах гигантов»**
  - Есть обширная литература по теории категорий
  - Теория Nominal Sets and Techniques школы Andrew Pitts'а [1993-...]
    - Pitts, A. 2013. **Nominal Sets: Names and Symmetry in Computer Science**. Cambridge Tracts in Theor. Comp. Sci., vol. 57. Cambridge: Cambridge University Press. DOI:10.1017/CBO9781139084673

Зачем уходить из теории множеств?
Поначалу всё хорошо

Функциональные языки (ФЯ) и их теоретико-множественная денотационная семантика

Без статических типов, первого порядка (например, чистые Рефал, Лисп без λ)

– одна область данных D – плоская решетка с $\perp$ – не определено, не завершается

– определения функций на ФЯ $f(x) \coloneqq E[f, x]$, где E — выражение на ФЯ, рассматриваются как уравнения на f вида $f = F[f]$, где $F[f] = \lambda x.E[f, x]$,
и решаются итеративно до наименьшей неподвижной точки:

$f_0 \coloneqq \perp$ — всюду «неопределенная» функция: $(\forall x \in D)\, f(x) = \perp$

$f_{i+1} \coloneqq F[f_i]$, $i \to \infty$

– решение существует на полной (полу)решетке функций
с отношением частичного порядка $\sqsubseteq$ по степени «определенности» функции

$f \sqsubseteq g$, если $(\forall x \in D)\, f(x) \sqsubseteq g(x)$

то есть функции сравниваются «поаргументно» на плоской решетке значений функций

Без статических типов, высшего порядка (например, чистый Лисп)

– проблема: область данных D содержит функции над ней же: $D \supseteq D \to D$

– такая наивная теория множеств противоречива

– Dana Scott [1971] построил кодировку этой области в ТМ («предел» посл-ти $D_{i+1} \coloneqq D_i \to D_i$)

Со статическими типами, не позволяющими определять «противоречивые» типы данных вида $D \supseteq D \to D$ (например, языки семейства ML, Haskell и далее к dependent types)

– ОК с рекурсивными определениями со стрелкой через конструктор

type *D* ≔ *G Ground* | *F* (*D* → *D*)

– здесь конструктор *F* «кодирует» функции в данных (можно считать, по Dana Scott'у)

Что в объектно-ориентированных языках?

Объектно-ориентированные языки (ООЯ)

- содержат ключевое понятие объекта, идентифицируемого ссылкой
- ООЯ невозможны без ссылок в качестве данных — уникальных имен объектов, порождаемых динамически при каждом исполнении оператора **new**
- понятие ссылок «в чистом виде» без объектов имеет смысл как «имен», «атомов», порождаемых оператором **new** — теория Nominal Techniques Andrew Pitts'а [1993–]

Иерархия построения объектно-ориентированных языков

object-based

1. понятие ссылки, имени с оператором **new**
2. понятие объекта, идентифицируемого уникальной ссылкой и обладающего изменяемым состоянием
   - без понятия ссылки невозможно говорить об изменяемых сущностях (или лишь об их конечном числе, определяемом статически программой)
   - понятие процесса исполнения вводится на шаге 2, чтобы изменять состояния

class-based

3. понятие класса, как конструкции языка, описывающей операции над объектами и инкапсулирующей состояние объектов
   - объекты «принадлежат» классу C от их рождения оператором **new** C

object-oriented

4. понятие наследования классов

← Классификация Peter Wegner'а [1987-90]

Надо уходить из теории множеств: споткнулись на именах, ссылках на объекты

Чисто-функциональный язык с именами, то есть со ссылками, но пока без объектов

Определение. Язык обладает ссылочной прозрачностью (referential transparency), если одинаковые подвыражения в одинаковых контекстах выдают равные значения, в частности:

$$x = y \;\Rightarrow\; f(x) = f(y)$$

А в чем проблема в объектно-ориентированных языках?

Еще не говоря об изменяющихся состояниях – проблема в отсутствии ссылочной прозрачности в языках с генератором новых имен **new**

$$\textbf{new} \neq \textbf{new}$$

$$\textbf{let}\ \ x \coloneqq \textbf{new};\ y \coloneqq \textbf{new}\ \ \textbf{in}\ \ \ x \neq y$$

$$\textbf{let}\ \ x \coloneqq \textbf{new};\ y \coloneqq x\ \ \ \textbf{in}\ \ \ x = y$$

$$\textbf{let}\ \ f(x) \coloneqq \textbf{new}\ \ \ \textbf{in}\ \ f(x) \neq f(y)$$

Из-за этого денотационная семантика, построенная для ФЯ, неприменима к **ФЯ+new**

Теория ФЯ+new на базе теории категорий — Andrew Pitts с учениками [1993–]

> The 2019 ACM Alonzo Church Award for Outstanding Contributions to Logic and Computation is given jointly to Murdoch J. Gabbay and Andrew M. Pitts for their **ground-breaking work introducing the theory of nominal representations**
> http://siglog.org/winners-of-the-2019-alonzo-church-award

От равенства к эквивалентности по Лейбницу

Нельзя ли ослабить ссылочную прозрачность?

Определение. Язык обладает ссылочной прозрачностью (referential transparency), если одинаковые подвыражения в одинаковых контекстах выдают равные значения, в частности:

$$x = y \;\Rightarrow\; f(x) = f(y) \tag{1}$$

Надо избавиться от равенства, нарушаемого, когда f возвращает новые ссылки, напр. $f(x) \coloneqq \textbf{new}$

Нет ли эквивалентности, которая выполняется для данных со ссылками в ФЯ+new?

Эквивалентность по Лейбницу = контекстуальная эквивалентность = observational equivalence

$$x \approx y \coloneqq (\forall C : D \to Bool)\; C(x) = C(y) \tag{2}$$

Функции и данные с такой эквивалентностью в **ФЯ+new** обладают ссылочной прозрачностью:

$$x \approx y \Rightarrow f(x) \approx f(y) \tag{3}$$

$$\big((\forall C : D \to Bool)\; C(x) = C(y)\big) \Rightarrow \big((\forall C : D \to Bool)\; C(f(x)) = C(f(y))\big)$$

Сравнение пар $\langle \text{аргумент}, \text{значение} \rangle$ отличается от раздельного сравнения аргументов и результатов:

$$x \approx y \Leftrightarrow \langle x, f(x)\rangle \approx \langle y, f(y)\rangle \tag{4}$$

Правое условие в (4) сильнее, чем в (3), так как в (5) импликация только в одну сторону:

$$\langle x, f(x)\rangle \approx \langle y, g(y)\rangle \;\Rightarrow\; x \approx y \;\&\; f(x) \approx g(y) \tag{5}$$

Например, пусть $x \coloneqq r_1 \coloneqq \textbf{new}$, $y \coloneqq r_2 \coloneqq \textbf{new}$, $f(x) \coloneqq \langle x, \textbf{new}\rangle$, $g(x) \coloneqq \langle \textbf{new}, x\rangle$. Тогда:

$$\langle r_1, \langle r_1, r_3\rangle\rangle \not\approx \langle r_2, \langle r_4, r_2\rangle\rangle \qquad r_1 \approx r_2 \;\&\; \langle r_1, r_3\rangle \approx \langle r_4, r_2\rangle \tag{6}$$

Лейбниц-прозрачность и денотационная семантика со ссылками (1)

Определение. Назовем Лейбниц-прозрачностью модифицированную ссылочную прозрачность с эквивалентностью по Лейбницу (Лейбниц-эквивалентностью)

Ссылочная прозрачность (исходная и модифицированная) – основа построения следующих понятий:

- денотационная семантика
- мемоизация / табуляция при вычислениях
- ленивые вычисления
- детерминированный параллелизм и прочие прелести

Вопрос. Можно ли это сохранить с Лейбниц-прозрачностью?

Денотат функции $f : X \to Y$

- для **ФЯ** в **ТМ**: график = множество пар $аргумент \mapsto значение = \{x \mapsto y \mid x \in X, y = f(x)\}$
  - операционно: график порождается вычислением $f(x)$ при всевозможных $x : X$
  - вычисление $f(x)$ эквивалентно извлечению y из соответствующей пары $x \mapsto y$ графика
- для **ФЯ+new**: мемо = множество пар $шаблон\ аргумента \mapsto шаблон\ значения$
  - мемо порождается вычислением $f(x)$ при всевозможных $x : X$

Лейбниц-прозрачность и денотационная семантика со ссылками (2)

Денотат функции $f : X \to Y$

- для **ФЯ+new**: мемо = множество пар *шаблон аргумента* $\mapsto$ *шаблон значения*
  - мемо порождается вычислением $f(x)$ при всевозможных $x : X$
  - *шаблон аргумента* – значение x при первом обращении $f(x)$, в котором ссылки $r_1 \ldots r_n$ рассматриваются как переменные для отождествления в вызовах $f(x')$ аргумента x' с x

    $$\lambda\, r_1 \ldots r_n .\, (x \mapsto \textit{шаблон значения})$$
  - *шаблон значения* имеет вид $\nu\, r_{n+1} \ldots r_{n+k}\, .\, y$, где y – значение, полученное при первом $f(x)$, а $r_{n+1} \ldots r_{n+k}$ – список ссылок, порожденных при вычислении $f(x)$

    $$\lambda\, r_1 \ldots r_n .\, (x \mapsto \nu\, r_{n+1} \ldots r_{n+k}\, .\, y)$$
  - вычисление $f(x')$ эквивалентно отождествлению x' с x в некоторой паре в мемо, генерации k новых ссылок для $r_{n+1} \ldots r_{n+k}$ и подстановки в y полученных значений ссылок-переменных

определение и вызовы функции	промежуточные вычисления	состояние мемо
$f(x) \coloneqq \langle x, \textbf{new} \rangle$		$\{\}$
$f(\textbf{new}) \Downarrow \langle r_1, r_2 \rangle$	$r_1 \coloneqq \textbf{new}$ $r_2 \coloneqq \textbf{new}$	$\{\lambda r_1 . r_1 \mapsto \nu\, r_2 . \langle r_1, r_2 \rangle\}$
$f(\textbf{new}) \Downarrow \langle r_3, r_4 \rangle$ используем не определение f, а находим пару в мемо	$r_3 \coloneqq \textbf{new}; r_1 \coloneqq r_3$ $r_4 \coloneqq \textbf{new}; r_2 \coloneqq r_4$	$\{\lambda r_1 . r_1 \mapsto \nu\, r_2 . \langle r_1, r_2 \rangle\}$

- Есть тонкость: равные / неравные ссылки в аргументе x

Лейбниц-прозрачность изменяющихся объектов

Теория ФЯ+new построена группой Andrew Pitts'а [1993–]

Главный вопрос

– Насколько эти результаты и свойства языка можно перенести на изменяющееся объекты в объектно-ориентированном языке? Какие ограничения надо наложить на такой **ООЯ**?

– Можно ли сохранить Лейбниц-эквивалентность для **ООЯ** с объектами некоторых классов, типов с ограниченными свойствами? Каковы эти ограничения?

Трудности

– В **ФЯ+new/ООЯ** с объектами произвольных классов (типов), определенных в объектно-ориентированном языке, Лейбниц-прозрачность не имеет места

– Кроме того, требуется превращение **ФЯ** в императивный с понятием процесса(ов), поскольку понятие изменяющегося состояния не имеет смысла без порядка вычисления

Определение

Назовем Лейбниц-классами классы объектно-ориентированного языка, добавление которых в **ФЯ+new**, сохраняет Лейбниц-прозрачность, а их объекты назовем Лейбниц-объектами

Фундаментальная математическая проблема

Охарактеризовать Лейбниц-классы

Трудности и тонкости

Неразрешимо, невычислимо, неперечислимо

Аналогично (эквивалентно?) непротиворечивости математических теорий

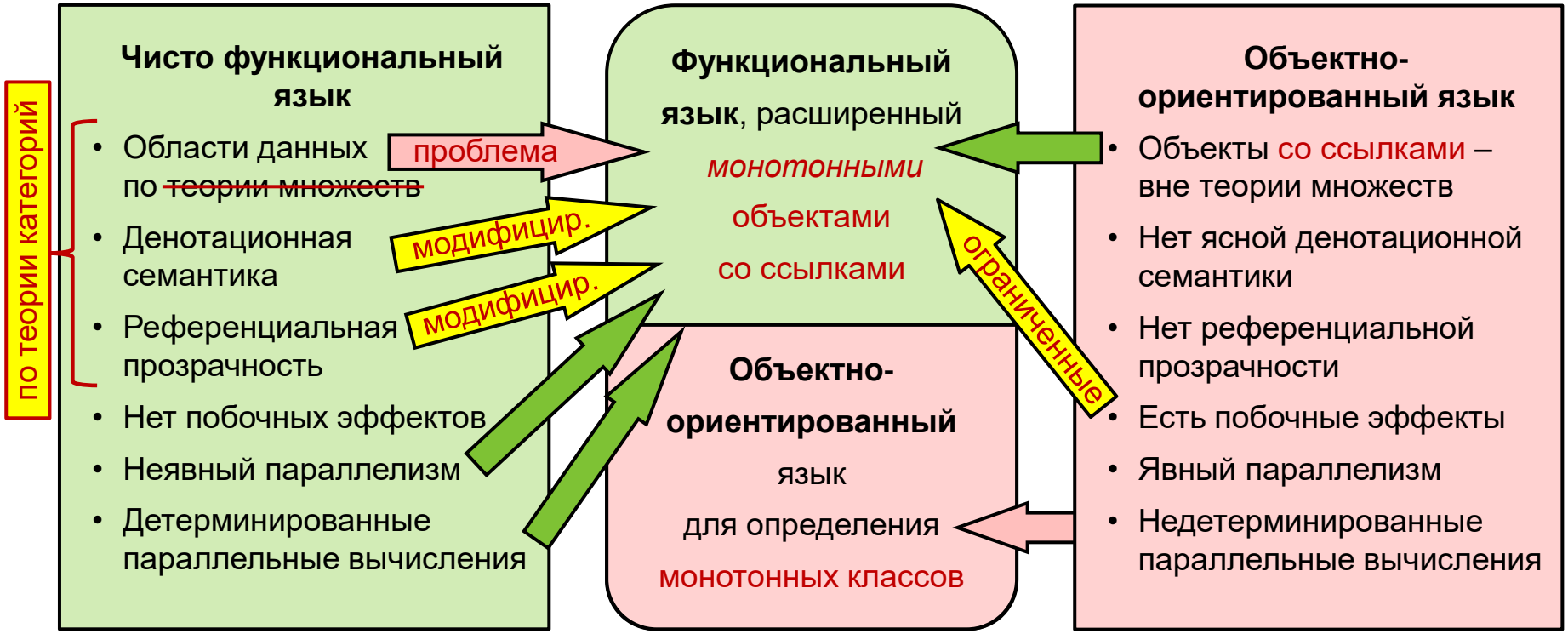
Пример тонкости: классы C_1 и C_2 по отдельности могут быть лейбницовыми, а вместе нет

Программистская цель: вот зачем городим огород

Цель. Построение модели вычислений между функциональной и объектно-ориентированной, сохраняя ценные свойства функциональной и расширяя область ее эффективных приложений

Подход. Входной язык состоит из двух подъязыков:

- «Верхний» язык – чисто-функциональный, расширенный ссылками и вызовами операции создания нового объекта **new** и операций над объектами, определенных в «нижнем языке»
- «Нижний язык» – обычный объектно-ориентированный язык, например Java



Монотонные объекты и классы

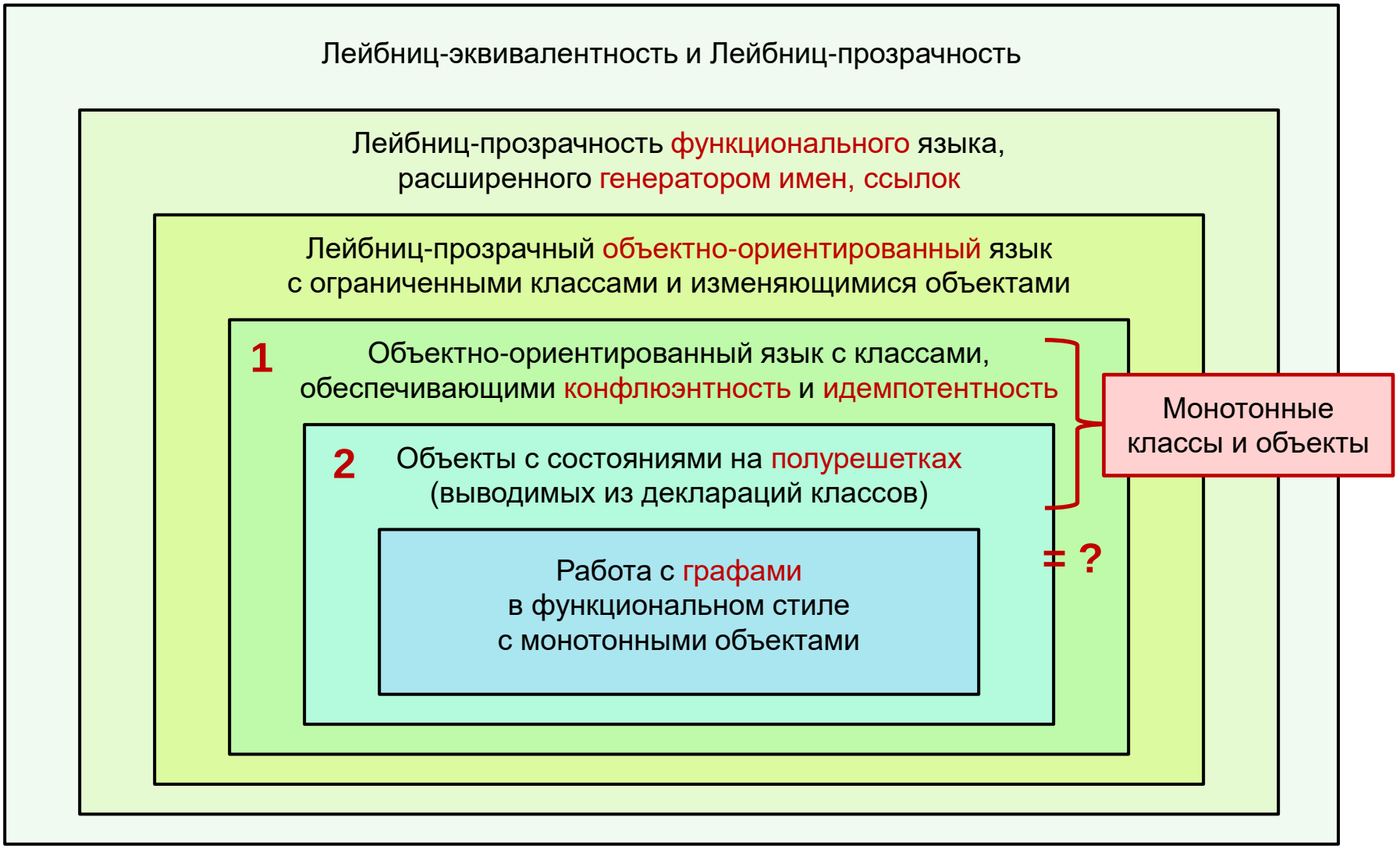
Определение. Монотонными называются такие оо-классы и их объекты, что любая использующая их программа на функциональном языке удовлетворяет следующим свойствам:

1. **Операционные свойства** (формально)
   - Конфлюэнтность параллельных вычислений (св-во Чёрча-Россера)
     - результаты, полученные в любом порядке вычислений любого выражения, Лейбниц-эквивалентны
   - Идемпотентность:
     - повторное вычисление копии любого выражения дает Лейбниц-эквивалентный результат, а побочный эффект неотличим (из функциональной программы)
2. **Желаемые свойства** (пока неформально)
   - Существование полурешетки, по которой состояния объектов изменяются монотонно
     - выводимой из деклараций монотонных классов
   - Наличие денотационной семантики
     - обобщающей семантику функциональных языков
     - не использующей порядок вычислений (поэтому параллельной)

Фундаментальные вопросы

- Следует ли 2 из 1?
- Монотонные классы являются лейбницовыми, а что между ними?
- Являются ли монотонные классы «стабильными»? (*подскажите термин!*)
  т.е. их добавление к лейбницовым консервативно, то есть «лейбницовость» сохраняется

Иерархия и объемы понятий



Практическая проблема и тестовая задача: обработка графов

Основное практическое ограничение чисто функционального программирования

- Функциональные языки позволяют строить и эффективно обрабатывать данные, представимые деревьями
- Объектно-ориентированные языки позволяют эффективно обрабатывать произвольные графы, представляя отношения между вершинами и дугами ссылками на объекты

Тестовая задача для всей этой теории

- Построение циклических структур данных в функциональном языке с монотонными объектами с сохранением его ценных свойств

Пример статьи по обработке графов на ФЯ с неудовлетворяющим нас решением

- Bruno C.d.S. Oliveira and William R. Cook. 2012.
  Functional programming with structured graphs.
  SIGPLAN Not. 47, 9 (September 2012), 77–88.
  DOI: https://doi.org/10.1145/2398856.2364541

Резюме по модели вычислений и обработке графов на ФЯ

- Сделаны первые шаги построения новой модели вычислений между функциональной и объектно-ориентированной парадигмами
- Она реализуется двухуровневым языком программирования:
  - Верхний уровень – функциональный язык (или ему подобный)
  - Нижний уровень – объектно-ориентированный язык
- Основная идея – ограничить методы классов (называемых монотонным), используемых на функциональном языке, так чтобы сохранялись следующие свойства функциональных программ, вместе обеспечивающие детерминированность программ:
  - Конфлюэнтность параллельных вычислений
  - Идемпотентность побочных эффектов и результатов вычислений
- Ключевая решаемая проблема – преодоление главного ограничения функционального программирования, что данные могут быть только деревьями
- Приведены примеры монотонных классов, используя которые функциональные программы могут создавать циклические структуры данных, в которых каждый объект доступен по уникальной ссылке, как в обычно объектно-ориентированном программировании

Публикации по программированию с монотонными объектами

Авторы – Алексей И. Адамович и Андрей В. Климов

1. **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.

2. **Подход к построению системы детерминированного параллельного программирования на основе монотонных объектов**

   Вестник СибГУТИ. 2019. № 3. С. 14–26.

3. **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.

4. **О детерминированной параллельной реализации метода ветвей и границ на монотонных объектах**

   Научный сервис в сети Интернет: труды XXI Всероссийской научной конференции (23–28 сентября 2019 г., г. Новороссийск).
   М. : ИПМ им. М.В. Келдыша, 2019. С. 3–18.

PDF-файлы статей доступны на https://pat.keldysh.ru/~anklimov/pub