*Содержание*

1 Введение.

1.1. Позиция в отношении к предмету.

1.2. Составляющие обучения.

2 Постановка задачи: о связном предъявлении теории информатики и практики программирования в теме ИСПОЛНЕНИЯ для теоретического мышления.

3 Аппликативный стиль. Денотационная семантика. (класс Прагматика).

3.1 Проблемная ориентированность (программистский аспект).

3.1.1 Сравнение языков программирования стиля.

3.1.2 Неподвижная точка как подходящее обобщение проблемной ориентированности.

3.2 Редукционные реализации (математический аспект).

3.3 Модели теорий и типовые языки (системологический аспект).

3.3.1 О моделях λ-алгебры или λ-теории.

3.3.2 Общие аспекты, связанные с неподвижной точкой.

3.4 Базовое обучение на основе аппликативной среды информационного моделирования в учебнике М. Броя “Информатика. Основополагающее введение.” Часть I.

3.4.1 Пример базового обучения на основе аппликативного стиля.

3.4.2 Некоторые замечания.

4 Класс обучаемых Универсал: предъявление понятия Выразимость.

4.1 Программистский аспект.

4.2 Математический аспект.

4.2.1 Èíèöèàëüíàÿ àëãåáðà.

4.2.2 Óïîðÿäî÷åííî-ñîðòíûå òåîðèè.

4.3 Системологический аспект.

4.4 Элементы среды саморазвития.

5 Реализация задания практикума для аппликативного стиля.

5.1 Макеты задания.

5.2 Понятия.

6 Заключение.

7 Результаты работы.

8 Литература.

9 Приложения.

А. Ранние требования к предметной области системы (до введения уровня спецификаций). 1

B. Несколько слов об энергичных и ленивых интерпретаторах. 2

C. SECD-машина. 3

D. Среда саморазвития для идеи интерпретации. 5

E. Бестиповое лямбда-исчисление. Определения. 15

F. Задание практикума. 16

# 1 Введение.

Дипломная работа выполнена в рамках проекта “Компьютерный комплекс обучения основам информатики” (грант 93 -01-01-047 РФФИ, научный руководитель проф. Трифонов Н. П., ответственный исполнитель н.с. Громыко В. И.) и соглашения о совместной научной работе между факультетами математики и информатики FSU-Jena (Prof. G. Wechsung), Германия и факультетом ВМК по теме “Учебные пособия и обучающие системы в области информатики” с консультационным участием Prof. M. Broy, Prof. A. Blaser и н.с. А. Г. Симакина (Институт психологии).

## 1.1. Позиция в отношении к предмету.

Важно определиться в главном - оценить происходящие изменения в человеческом виде. Исходим из признания факта, что меняется подсознательный стиль мышления. Такие периоды сопровождаются активной деятельностью рационального в нашем мышлении. Конкретно, связываем изменения с развитием компьютерного дела, требующего от рационализма обрести ту же общность, которую давно демонстрирует гуманитарное знание. Естественно, что рационализм решает эту задачу на своем пути, выводя рациональное знание на более сложный уровень - *синтезирующего научного знания*. Этому определяюще способствует информатика, обеспечивающая существование общества в век систем, и являющаяся сензитивной деятельностью Человека как вида.

Уже прошло много лет после того, когда “мы и в самом деле преуспели в преобразовании программирования в науку, причем замечательно простым способом - решив называть ее computer science (компьютерной наукой)” [2]. С тех пор она действительно обрела нужный статус. Но настало время, когда это ее свойство должно стать заметным и для будущего профессионала в информатике и для любого человека, уже существующего в условиях информационного общества. Поэтому та же проблема (синтезирующей науки), смещенная в область образования - синтезирующего знания с навыком созидания в программировании, явление актуальное: информационный рационализм должен стать составляющей нашего интеллекта. Каким бы банальным не показалось утверждение, повторим его еще раз - базовое обучение в информатике должно сосредоточиться на *синтезирующих основах предмета в аспекте их роли для конструирования в программировании (созидания).* К сожалению, образование, в силу консервативности, всегда обременено “культурной отсталостью”. К тому же усложнение средств информационного моделирования потворствует примитивной позитивистской интерпретации программирования как искусства и тем самым необязательности, ненадобности (зла) научного восприятия.

Итак, во многом, изменению парадигмы в образовании - от унаследования к овладению знанием - мешает позитивистский миф об искусстве программирования. Плохо, что мифу неявно потворствуют такие авторитеты как Дональд Кнут (его трехтомник /74 г./ называется “Искусство программирования для ЭВМ”). Неявно, так как его понимание искусства в программировании правильно фиксирует внимание на созидающем аспекте в программировании, а также правильно выделяет научный характер такой деятельности. На этом основании следует лучше признать синтезирующий характер профессиональной деятельности в век систем, которая, в связи с программированием, наполнена конструирующей особенностью. Впрочем, обстоятельства сильнее нас, иногда трехтомник Кнута называют “Дело программирования”. В этой же связи обратим внимание на другой авторитет - Дэвида Гриса и его книгу “Наука программирования”, который, в конце концов, заявил /81 г./: ”Из-за нашего невежества мы потратили очень много времени на то, чтобы бить по воде, вместо того, чтобы плавать. Задним числом я могу сказать, что мне лучше всего было бы пройти хороший курс логики 10 лет назад”. Также надлежит обратиться к авторитету Эдсгера Дейкстра с его книгами “Краткое введение в искусство программирования” /71 г./ и затем “Дисциплина программирования” /76 г./. Он говорит: “... я чувствую себя сродни преподавателю композиции в консерватории. Он не учит своих подопечных тому, как следует писать конкретную симфонию, но он должен помочь им обрести собственный стиль и объяснить, что под этим подразумевается”. Но уже приходится считаться, что время требует в обучении большего - сосредоточиться на развитии творческих особенностей интеллекта. Таким образом, предыдущее обучение - “преподаватель композиции” становится только необходимой составной частью. Теперь нуждаемся в “преподавателе сочинительства”.

Синтезирующей особенности созидающей деятельности человека подчинено Развивающее Обучение (РО) [3]. Оно особый упор делает на приобщение к теории предмета, развитие теоретического мышления, но посредством развития свойства теоретической самоорганизации интеллекта. Поэтому так своевременна позиция Дж. Р. Хиндли, которая хорошо ориентирует в отношении нашего предмета: ”Программисты-теоретики сталкиваются с теми же вопросами, над которыми логики (особенно принадлежащие к различным школам конструктивной математики) ломают голову уже около 80 лет. Например: что такое математическое мышление и как его формализовать? Поэтому, - если, конечно, логики не совсем сбились с пути, - их исследования могут служить, по крайней мере, полезным источником идей” [9]. Исходя из этого, для целей базового обучения, информатику следует воспринимать как математику, сосредоточенную на метаматематических вопросах предъявления знаний, связанных с пониманием его существенной зависимости, возможности от выбираемых языковых средств. Тогда фундаментальными вопросами становятся: возможности и границы аксиоматического описания (основных математических структур), а также возможности и границы эффективных методов, то есть алгоритмов. (Сравним с позицией Ю. Манина, выраженной в книгах “Доказуемое и недоказуемое”, “Вычислимое и невычислимое”). В этом ключе проблемную ориентированность языковых средств программирования следует воспринимать как определение (пользовательское) истинности для соответствующего стиля. На главную роль в обучении уже претендуют среда информационного моделирования также, как раньше, - учебные вычислительные машины или позже, языковые вычислители золотого фонда языков программирования. Таким образом, приходим к необходимости рассматривать в базовом обучении информатике *современный аксиоматический метод.* Он включает в себя язык, теории, определение истинности, представление моделей, разрешение или выводимость (в основном, в ракурсе конструктивных предъявлений). Возможность обучения на этой основе следует из зрелости предмета информатики, выраженной в существовании пользовательской семантики фундаментальных абстракций, а главное - в наличии ясных и разработанных ориентиров нового интеллектуального качества - {конструктивности (прагматическая деятельность), эффективности (профессиональная деятельность), выразимости (мировоззренческое восприятие)}, а также столь же ясных прагматических экстенсиональных обобщений средств информационной обработки - {исполнение, типизация, система, язык}. Возможность усвоения на младших курсах абстрактных (главных) ориентиров обучения во многом связана с конструктивным воплощением изучаемого в компьютере, что позволяет обеспечить необходимую практическую (осязаемую) деятельность обучаемого.

Конкретизация РО для базового обучения информатике, которое выстраивается на прагматике предмета - программировании, проводится с учетом его современных проблем. На этом пути привлекательны конструктивностью позиции Н. Вирта и Ч. Э. Р. Хоара. Первый призывает осмысливать собственные информационные проблемы посредством моделирования подходящего языка программирования (послужной список созданных им языков впечатляет: Эйлер, Алгол-W, Паскаль, Модула, Модула-2, Оберон). Тем самым он фиксирует внимание на необходимости поднимать прагматические проблемы любой деятельности по конструированию в программировании до высот их обобщенного, абстрактного восприятия. Второй неоднократно демонстрировал конструктивную мощь *взаимодополняющих семантик:* реализаторской, пользовательской (проблемно-ориентированной) и авторской (выражающей “идеальную”, идейную суть предметной области). Состояние дела обучения уже предлагает нам удачные образцы синтезирующего предъявления семантик. Книга А. Филда, П. Харрисона “Функциональное программирование” (1988г., у нас 1993г.) решает проблему связного изложения семантик для функциональных языков. Конечно же, в новом не все получается с первой попытки. Например, не удалось в качестве авторской семантики предложить модели λ-исчислений и комбинаторных алгебр. Но вопрос этот ставится и обсуждается на основании доверия к сообщенным математическим фактам. Приведем еще один пример новаторства - учебник (в четырех частях - ≈1200 страниц) “Информатика. Основополагающее введение” профессора М. Броя (M. Broy), лауреата премии Лейбница Германской академии наук за 1993г., одного из лидеров немецкой теоретической школы, преемника и ученика Ф. Л. Бауэра (F. L. Bauer). Учебник написан в 1992-1995 гг., у нас первая часть вышла в 1996 г. Он продолжает традицию концептуального взгляда на предмет, заложенную в известном учебнике Ф. Л. Бауэра и Д. Гооза (G. Goos) “Информатика” (в Германии были 3 переработанные издания книги, на русском языке был перевод со 2 издания в 1976г., с 3 издания 1984 года - в 1990г.). В первой части предлагается введение в информатику на основе аппликативной среды информационного моделирования. В остальных частях рассмотрены вопросы устройства современных базовых вычислителей, устройства языков программирования, эффективности вычислений. В целом, наша позиция исследования отличается от приведенных двумя моментами: ориентацией на привлечение к обучению разных сред информационного моделирования; еще большей концептуальностью (современный аксиоматический метод) в отношении предмета информатики, но и особой заботой о восприятии обучаемым предлагаемого (логика открытия).

Так как наиболее реально выражены результаты происходящих изменений в предмете, то фиксирование сущности базового обучения информатике начнем с констатации позиции в отношении предъявления предмета: ***на уровне среды информационного моделирования связывание прагматики программирования с теорией предмета мы проводим на основании взаимодополняющих семантик в ракурсе современного аксиоматического метода.***

## 1.2. Составляющие обучения.

Практической целью работы является повышение эффективности базового обучения информатике за счет предоставления её теоретических образующих. Противоречие между наукоемкостью теоретических понятий и неподготовленностью обучаемого (студента-первокурсника) решается методом РО. Метод состоит в подключении творческих возможностей интеллекта обучаемого при овладении предметом и для его овладения.

Общей целью обучения является развитие у обучаемого умения воспринимать, предъявлять и исследовать знания. Метод РО ориентирует на предъявление предметной области (ПО) как среды развития для логики открытия (ЛО) обучаемым предмета, то есть для его творческой деятельности по самостоятельному переоткрытию предмета, и на этом основании формированию универсального качества интеллекта - теоретической адаптивности. Общая цель конкретизируется посредством РО при её разработке в отношении базового обучения информатике. Этому служат: учет системологического подхода к развитию научной культуры от века машин к веку систем; восприятие информатики как дальнейшего развития математического механизма по своевременному приобщению нашего интеллекта к познающему научному разуму века машин и к созидающему разуму века систем; понимание предмета при базовом обучении широко, как симбиоза программирования, математики и информатики. Таким образом, конкретной целью обучения является приобщение обучаемого на предмете к интеллектуальному состоянию, которое свойственно системному аналитику. То есть комплекс должен обеспечить учащегося понятийными средствами об инструментах информационного моделирования для предъявления знания для его исследования, причем сократовским методом обучения (посредством логики открытия).

Актуальность задачи диктуется возрастающим объемом профессиональных граней предмета, к тому же подверженных быстрым изменениям. РО старается привить способность ученику самостоятельно переоткрывать многое на основе последовательного усложнения его видения существенного. Здесь находятся истоки РО, которое озабочено теоретическим мышлением, формируемым посредством логики открытия (обучаемым предмета). Комплексный характер задачи приводит к комплексу обучения. В нем необходимы:

* *учебник*, выстроенный традиционно на основе логики обоснования предмета, который послужит основой для курса лекций; но выбранные темы соответствуют цели - теоретическому кругозору;
* *компьютерный задачник - учебник*, сформированный для логики открытия обучаемым предмета, который занят естественно обучением предмету, но старается использовать теоретическую самоорганизацию интеллекта обучаемого;
* *обучающая система* для самостоятельной работы ученика, преследующая цель помочь ему в его творческой самоорганизации.
* последней составляющей, обеспечивающей комплекс достаточностью в повышении эффективности базового обучения, является традиционная связь *Учитель - Ученик*; учитель корректирует, контролирует ученика в их совместной работе по творческой деятельности ученика (субъект творчества) в его самообучении.

Создание комплекса переводится в реальный план посредством моделей обучаемого, предметной области (ПО), обучения.

* *Модель обучаемого* отражает интегральное видение состояния, перспективу предмета в его отношении к интеллектуальным качествам, в особенности, связанным с самоорганизацией. Поэтому моделирование творческой самоорганизации начинается с фиксирования наиболее общих интеллектуальных состояний на предмете - априорной модели обучаемого, а также подходящего образа конкретной деятельности в программировании. Выделенными состояниями являются три класса и две категории обучаемых, представленные на рис. 1. Часть уточнённой априорной модели обучаемого, относящейся к профессионалу, с точностью до аспектов представлена на второй части рисунка. Аналогично уточняются модели классов универсала и прагматика.

 Рис. 1: Модель обучаемого.

Такой выбор трех классов ("прагматик", "профессионал", "универсал") следует из кантовского анализа научного познания (рассудочное, разумное), которое мы связываем с использованием, устройством и границами применения компьютера для предъявления знания для его исследования. При этом использование мы соотносим с рассудочной деятельностью, а устройство и границы связываем с разумным охватом предмета. Синтезирующий характер предъявления знания и необходимая для этого самоорганизация теоретического разума приводит к выделению категории "личность". Итак, из общей цели обучения и значения концептуального каркаса предмета следует требование - дать обучаемому возможность сбалансировано представить (приобщиться) инструменты информационного моделирования: их применение (прагматик), устройство (профессионал) и границы применения (универсал). Интегральное интеллектуальное состояние на предмете соответствует состоянию системного аналитика, охватывающего предмет от его возможности по предъявлению знания до реализации этого предъявления на компьютере. Выделение категории "начинающий" вызвано требованием метода РО о необходимости предъявлять ориентиры обучения посредством познавательной инверсии. Дело в том, что метод основан на фиксировании доминирующей роли культуры (филогенез мышления) в развитии мышления каждого человека (онтогенез). Поэтому требуется отыскивать доступное представление ориентиров в ПО и посредством их раннего предъявления обучаемому (с точки зрения их позднего появления в культуре) формировать в нем целеполагание для логики открытия. Доступность представления объясняется уточнением А. Северцова закона развития Г. Геккеля: филогенез есть последовательность проявленных онтогенезов. Для предъявления ориентиров посредством познавательной инверсии разрабатываются специальные среды саморазвития.

Ядро учебного материала для каждого класса обучаемых готовится на основании пользовательской семантики. Семантической разработке подвергаются средства информационного моделирования, но рассматриваемые в их отношении к главному концептуальному ориентиру класса: для прагматика - к конструктивности, для профессионала - к эффективности, для универсала - к выразимости. Таким способом практика программирования “завязывается” с необходимой для РО теорией предмета. Естественно, что наиболее ёмкий ориентир для личности - это конструктивность. Так как “планкой” обучения является стиль среды информационного моделирования, то конструктивность разрабатывается широко. Во-первых, со стороны программирования она разрабатывается инвариантно относительно языков стиля - программистский аспект. Во-вторых, со стороны информатики привлекаются: математический аспект - спецификация, проясняющая реализацию, и системологический аспект - спецификация, проясняющая предъявление знания. Для классов профессионала и универсала применим тот же подход. Но, если сравнивать с прагматиком, то, вообще говоря, действуем уже, так как рассматриваются отдельные темы - эффективность и выразимость. Эффективность, в некотором роде, является продолжением прояснения вопросов реализации, но инвариантно относительно конкретных базовых вычислителей. Выразимость, в некотором роде, является продолжением рассмотрения вопросов предъявления знания, но инвариантно относительно “реальных” предметных областей, то есть с точностью до абстрагирования информатики. Итак, инвариантность выбранных ориентиров относительно стилей и сосредоточенность на базовом обучении позволяют и требуют действовать таким способом.

* *Модель ПО* должна вобрать ядро с большой творческой потенцией понятий, которые хорошо увязаны с прагматикой предмета. Поэтому ядро формируется на основании *сред информационного моделирования*, воплощенных в прагматике предмета в стилях программирования (по исполнению, по типизации, по системам): императивном, аппликативном, функциональном, логическом, Пролог-стиле, объектно-ориентированном, бестиповом, с сильной типизацией, с АТД. ПО предъявляется в двух согласованных качествах - в виде компьютерного учебника-задачника и в виде традиционного учебника. Согласование проводится на основании принципа РО, который требует пристального внимания к прагматике предмета, отыскивая в ней элементарные составляющие, непосредственно связанные с идеями фундаментальных абстракций предмета. Элементарные составляющие для развитого предмета выбираются естественно на основании его вхождения в общую культуру предмета и системологического видения основной цели обучения. Главными ориентирами выбираются естественные темы для инструментов информационного моделирования: от программирования - исполнение, типизация, система, язык; их математические обобщения - модель, алгоритм (теория), программа (язык); их обобщения в информатике - конструктивность, эффективность, выразимость. Так как в нашем рассмотрении предъявление знания выбрано общей целью обучения, то кажется уже достаточно обоснованным, что нормой анализа выбирается стиль - класс языков, связанных проблемной ориентированностью. Наличие искусственно разработанных идей в математике позволяет создать среды саморазвития для начинающего по внедрению в его интеллектуальный актив главных ориентиров предмета на основании элементарных составляющих. Ориентиры облегчат ему целеполагание в среде развития - работе с компьютерным учебником-задачником и традиционным учебником для собственного переоткрытия предмета.

*Компьютерный учебник-задачник* формируется на основании задания практикума, предназначенным для выполнения обучаемым. Сопутствующий заданию учебный материал соотносим с интеллектуальным состоянием, зафиксированным в модели обучения в виде класса прагматика. Для задания специально отыскиваются задачи, предоставляющие возможность предъявить разнообразие материала и не фальсифицировать предмет. Из этого следует необходимость их нетривиальности и межстилевого характера. Значит, задачи непосредственно должны быть связаны с особой ролью инструментов информационного моделирования - в нашем понимании, с предъявлением знания на компьютере. Поэтому задания, в первую очередь, обслуживают главные ориентиры обучения от программирования - исполнение, типизацию, системы, язык. Во-первых, общность обеспечивает единый взгляд на сравнение стилей; во-вторых, позволяет естественно развить ориентиры классов профессионала и универсала; Кроме того, наличествует разрыв между общностью главных ориентиров обучения и узостью элементарных составляющих с сопутствующими абстрактными представлениями в идеях, удается заполнить промежуточными формами, так называемыми наследуемыми ориентирами, обогащенными конструирующей ролью. Это позволяет учебный материал программистского аспекта прагматика наполнить теоретическим видением (понятиями). Наследуемыми ориентирами являются: структурное и доказательное (инвариант цикла) программирование, связанные с примитивностью "как делать" императивного стиля; смешанные вычисления нетипизированных аппликативных вычислителей, метод индукции, языковые расширения в направлении предъявления "что делать"; высокая верифицируемость комбинаторных вычислителей, алгебра программ, фиксирование "что делать"; для логического стиля много моментов - порождающее и вычисляющее программирование функций, управление для эффективного вывода, исчисление выше первого порядка, динамические расширения исчисления, фиксация исчислением "что делать". Важно, что наследуемые ориентиры через задание практикума существуют интегрально. Во-первых, это обеспечит активность обучаемого в творческом проявлении. Во-вторых, удержит разработчиков ПО от соблазна профессиональной ориентации учебного материала.

Связность учебного материала организуется межстилевым характером задания, а также наличием связующих факторов для разных интеллектуальных состояний - классов. Эти связи показаны на рисунке 2.

**Error! Not a valid link.**Например, задание практикума для темы Исполнение имеет следующий вид: написать программу, которая моделирует алгебраическую группу и позволяет решать уравнения вида E1\*x\*E2=A, где E1, E2, A - произвольные термы (выражения), x - неизвестное для заданных интерпретаций. Задание необходимо выполнить в нескольких стилях, например, в императивном, аппликативном и логическом. Дополнительные требования должны выявить иерархию в стилях с точки зрения предъявления знания. Во-первых, может быть требование об изменении типа группы добавлением новых законов. Во-вторых, может быть предложено использовать несколько интерпретаций заданной группы. Учебный материал разрабатывается при подготовке ответа и указаний для выполнения задания практикума. Ответ готовится в виде макетов-программ заданий. Среда саморазвития, учебные материалы для универсала и профессионала, выделение понятий спецификаций аспектов прагматика исследуются и предлагаются специально.

Для прагматика выбор материала, во многом, определяется помощью, которую необходимо оказать обучаемому по выполнению задания, привлекая для этого теорию предмета. Именно, программистский аспект прагматика нагружается понятиями и задачами для уяснения необходимых для выполнения окрестностей ЯП. Посредством наследуемых ориентиров уровень требований к характеристике учебного материала повышается и достигает необходимых теоретических обобщений (понятий). Математический аспект прагматика обеспечивается учебным материалом, связанным с рассмотрением вопроса эффективной реализации. Системологический аспект прагматика обеспечивается учебным материалом, связанным с рассмотрением вопросов выразительных возможностей проблемной ориентированности стиля. Для профессионала сосредотачиваемся на вопросах эффективности алгоритмов для компьютерных вычислителей. Для универсала сосредотачиваемся на возможности предъявления знания, которая в информатике обеспечивается современным аксиоматическим методом, то есть адаптацией теории моделей к компьютеру.

*Учебник строится* традиционно, но также с учетом общей и конкретной целей обучения. Для этого симбиоз математики, программирования и информатики предлагается рассматривать в виде, представленном на рисунке 3:



Рис. 3: Модель Предметной Области в учебнике.

В целом предметы понимаются: математика - формирующая деятельность (и, прежде всего, в отношении интеллекта) по образованию рационального единства из многообразия представлений; программирование - часть информатики, нацеленная на конструктивное предъявление знания на основе алгоритмического аспекта математики. Информатика - системологическая деятельность по конструктивному предъявлению неалгоритмического (эвристического) знания. Тогда общая структура учебника имеет иерархию тем, естественную для программирования, и иерархическую организацию внутри тем: математический уровень начального осмысления; программистское толкование рассматриваемого за счет расширений проблемной ориентированности стилей; информационный уровень, замыкающий рассматриваемое со стороны современной математики, решающей проблему предъявления знания на уровне конструктивности, эффективности, выразимости современного аксиоматического метода. Темы учебника естественны: это средства упаковки и распаковки знания посредством исполнения, данных, систем и в целом языка. Что касается изложения материала, то оно строится традиционно на логике обоснования предмета. Подчеркнем, что структура учебника выбрана в соответствии с логикой открытия, то есть учитывает прагматику предмета (выбор тем), но внутренняя иерархия каждой из них соответствует общей цели обучения. Необходимость традиционной организации учебника вызвана пониманием, что логика открытия не исключает рационального метода, а выводит его на более высокий уровень.

Сосредоточенность на базовом обучении проявляется в объединяющем прагматическом начале, которым является понятие - среда информационного моделирования. Это позволяет отследить сложность выбираемых абстракций. Только реальное предъявление материала в компьютерном варианте является признанием (фильтром) его подходящей приспособленности (наличествует пользовательская семантика) для восприятия, а тем самым возможному его включению в учебник. Таким объединяющим началом служат две известные (крылатые слова) парадигмы -от программирования:

Алгоритмы + Структуры данных = Программа;
от информатики:

Универсальная алгебра + Логика = Теория моделей ⊃ АТД.
В парадигмах заметны изменения в прагматике предмета: от “как делать” к “что делать”, причем с большим вниманием к тому “как это делается”.

* *Модель обучения* должна вобрать принципы РО - приготовить предмет для переоткрытия фундаментальных абстракций на основе сократовского способа обучения (познавательная инверсия). Основой модели являются: нацеленность на самостоятельную деятельность обучаемого - нетривиальные, но классические задачи, предъявляемые в заданиях практикума; независимость от педагогических установок - объективность подготовки материала (окрестности понятий, задач); обеспечение логики открытия - среды саморазвития начинающего; невосприятие как противоречия порочных логических кругов - их разрешение проводится посредством примеров; предъявление реальной связи теории и практики - посредством уровня спецификации практического материала по программированию, нацеленного на облегчение созидания в программировании; обеспечение разнообразных связей предмета - посредством проектирования на классы обучаемых составляющих современного аксиоматического метода.

Основой среды развития обучаемого является компьютерный учебник-задачник, формируемый для логики открытия, которая не зависит от конкретной предметной области, так как строится на интеллектуальных проявлениях - индукции, аналогии, наблюдении, эксперименте, гипотезе, обобщении. Она всеобща - как всеобщи перечисленные формы, используемые творческим человеком. РО нацелено на общие задачи теоретической самоорганизации интеллекта и служит логике открытия, что позволяет утвердиться в следующих ответах на ключевые вопросы базового обучения информатике: Зачем учить? - обеспечить состояние познающего интеллекта уровня воспроизводства научного знания; Кого учить? - творческую личность системно-информационного периода науки, продвигающуюся в состояние созидающего интеллекта; Чему учить? - информационному моделированию по предъявлению знания для исследования; Как учить? - тренажируя теоретическую адаптивность интеллекта.

Модель обучения реально представляется системой обучения FLINT [4]. Архитектура, функциональная схема системы, база знаний подчинены функции помощи обучаемому по переоткрытию основ предмета. Этому служат блоки системы, работающие на основе эвристик, которые должны способствовать - логике открытия, теоретическому мышлению, раннему целеполаганию на основе познавательной инверсии. Адаптивность системы к обучаемому достигается посредством его динамической модели.

Зафиксируем сущность предлагаемой позиции в отношении к составляющим обучения:

***обучается системный аналитик средам информационного моделирования (для предъявления знания) посредством среды (само)развития для логики открытия обучаемым предмета (сократовское обучение).***

# 2 Постановка задачи: о связном предъявлении теории информатики и практики программирования в теме ИСПОЛНЕНИЯ для теоретического мышления.

Связность предъявления материала, прежде всего, решается на уровне естественного согласования практики и теории предмета. Этому служит дополняющее совместное представление о тенденциях развития предметов программирования и информатики в лоне математики [4]. Имеются в виду следующие цепочки развития:

* программирования: базовый вычислитель 🡺 исполнение 🡺 типизация 🡺 банки данных 🡺 системы 🡺 банки знаний 🡺 экспертные системы;
* информатики (как развития математики): конкретная математика 🡺 аксиоматическая математика 🡺 теория моделей 🡺 конструктивная математика 🡺 дискретная математика 🡺 современный аксиоматический метод 🡺 самоорганизующиеся системы.
Для нас важной является граничная точка (системы), непосредственно касающаяся конструирования в программировании. Вспомним Ч. Э. Р. Хоара [5]: “... разработка программы и разработка ее спецификации должны проводиться параллельно одним и тем же лицом, и должно существовать взаимодействие между этими этапами работы. Недостаточная ясность спецификации является важнейшим признаком недостатков в программе, которую она описывает, и эти две ошибки должны устраняться одновременно, до того, как приступают к проекту”. /\*к реализации \*/

Итак, есть средство программирования, которое воспринимается

* *уровень-1* - через пользовательскую семантику проблемной ориентированности, то есть через уровень знания средства в конкретных языках программирования стиля;

Также оно рассматривается в более общем статусе информационного моделирования:

* *уровень-2* - через абстрактную семантику стиля, то есть через уровень знания его в подходящем языке спецификации, причем, по возможности, в трех ракурсах: с упором на пользовательскую позицию; с упором на суженную семантику реализаторской позиции (с точностью до компьютера); с упором на обобщенную семантику авторской позиции (с точностью до предъявления знания).

Поэтому выбранная тема ИСПОЛНЕНИЕ (и другие) разрешается: при разработке программистского материала в рамках среды информационного моделирования; через языки программирования, но как **представителей стилей**: императивного, аппликативного, логического, объектно-ориентированного, параллельного.

Кроме того, различие специфицирующих подходов разных стилей (из-за их проблемной ориентированности) также нуждаются в согласовании. Этому служит современный аксиоматический метод, который мы рассматриваем как синтезирующий подход в общей проблеме предъявления знаний. Через него удается рассмотреть с единых позиций:

* во-первых, проблемно-ориентированные спецификации стилей;
* во-вторых, рассматриваемые темы для разных классов обучаемых (для прагматика - *конструктивное* предъявление знаний, для профессионала - добавить к конструктивному предъявлению необходимую *эффективность*, для универсала - *выразительные* возможности общих средств предъявления знания);
* в-третьих, выделенные **идеи** проблемно-ориентированных стилей для создания *сред саморазвития начинающего*.

Соберем воедино предложенную сигнатуру метапонятий обучения в ключе проблем РО. При разработке учебного материала несколько моментов являются решающими для успеха проекта - по подключению творческих возможностей обучаемого (логики открытия) для переоткрытия предмета.

Во-первых, исходя из роли информатики для обеспечения нового интеллектуального качества информационного моделирования выделяем *главные ориентиры обучения*: исполнение, типизация, системы, язык, модель, теория, программа, конструктивность, эффективность, выразимость, стиль. Но, что еще важнее, необходима связующая позиция рассматриваемых ориентиров. Такой мы выбираем *современный аксиоматический метод*. Это он превратил в успех утерянную определенность математики (вспомним М. Клайна и его книгу “Математика - утрата определенности”), повысив точность и всеобщность теорий с обязательным предъявлением языка и формализацией истинности (вспомним опять же М. Клайна с его книгой “Математика - поиск истины”).

Во-вторых, следует обеспечить прагматический полигон, на котором могут быть представлены и обсуждены избранные “высокие” темы. Поэтому в основу кладется задание практикума, то есть нетривиальные, классические задачи программирования. С другой стороны, так как наше требование - предъявить задания в разных стилях программирования, то это поднимает уровень их обсуждения до необходимых РО вопросов. Напомним, что в некотором роде материал для системы обучения готовится таким образом, чтобы помочь обучаемому выполнить задание практикума.

В-третьих, необходимо тесно связать теоретический материал с прагматическим материалом программирования, так как в нем максимально проявляется созидание. Этому служит взаимосвязанная проработка взаимодополняющих реализаторской, пользовательской и авторской семантик на уровне стилей. То есть специально прорабатываются стили в ключе трех собирательных ориентиров обучения - *базовый вычислитель* (реализующий), *языковой вычислитель* (проблемно-ориентированный), *теоретический вычислитель* (ориентированный на метатеорию, то есть фиксирующий границы вычислителя в аспекте предъявления знания). В этой связи на задачи по программированию накладывается требование - наличие их теоретической спецификации. Что касается понятий, то они также должны быть обеспечены, по возможности, разноуровневой семантикой.

В-четвертых, так как для *логики открытия* необходимо предъявить видение (*целеполагание*), то обеспечивается *познавательная инверсия* в отношении ключевых аспектов (идей) теоретических вычислителей для начинающего. Этому служат *Среды Саморазвития* по идеям и их интегральному предъявлению. Это интегральное видение для предъявления идей обеспечивается через общую позицию - современный аксиоматический метод. Конечно же, здесь явно фиксируется нижняя граница требований к обучаемому - посредством выбранных элементарных составляющих, на которых, во многом, разворачивается обучение.

В-пятых, необходимо всегда иметь в виду наследуемые от главных ориентиры, которые выделены в предмете на основе его прагматического развития. Они могут послужить следующим уровнем элементарных составляющих для части обучаемых, продвинутых в прагматике предмета. Поэтому же они являются хорошим ориентиром для практических целей обучения.

В-шестых, для каждого класса обучаемых свой мотив предмета: *эффективность* для профессионала, *выразимость* для универсала, к*онструктивность* для прагматика. Эти главные грани разрабатываются в соответствии с перечисленными выше пятью требованиями.

Актуальным в дальнейшем продвижении комплекса обучения является установление плодотворной связи теории информатики с практикой программирования на основании синтезирующего характера современного аксиоматического метода в отношении предъявления знания. Поэтому теоретическую часть диплома составляют две задачи:

* выявить ключевые понятия аппликативного стиля по теме ИСПОЛНЕНИЕ для трех уровней спецификаций, относящихся к классу обучаемых Прагматик;
* выявить ключевые понятия для Класса обучаемых Универсал по вопросу ВЫРАЗИМОСТИ для организации среды саморазвития категории начинающего.

Практическая часть диплома нацелена на обеспечение обучающей системы FLINT материалом для компьютерного учебника-задачника:

* реализовать задание практикума для аппликативного стиля с предъявлением спецификаций на основании теоретической части диплома (задача 1).

# 3 Аппликативный стиль. Денотационная семантика. (класс Прагматика).

В пункте 1.1 было декларировано, что связывание прагматики программирования с теорией информатики мы проводим на основании взаимодополняющих семантик в ракурсе современного аксиоматического метода. В пункте 1.2. была представлена походящая модель ПО в компьютерном учебнике-задачнике и были зафиксированы опорные моменты, которые обеспечат нужную связь. Цель этого пункта - показать, насколько близка прагматическая сторона предмета к его теоретическим основам, насколько легче её восприятие через теорию предмета. Поэтому здесь проводится анализ стиля по определенным в пункте 2 требованиям.

## 3.1 Проблемная ориентированность (программистский аспект).

Как отмечалось в пункте 2 выявлению связи теоретического материала с прагматическим служит подход по разработке учебного материала на основании общих принципов, выраженных в следующих понятиях: главные ориентиры обучения, идеи, наследуемые ориентиры обучения, среда информационного моделирования, современный аксиоматический метод, взаимодополняющие семантики, системный аналитик. Применим подход к аппликативному стилю, чтобы выделить ключевые понятия, которые необходимо будет предъявить в учебнике-задачнике.

В первую очередь исследуем общее в проблемной ориентированности конкретных языков стиля. Вспомним, что идеей среды саморазвития аппликативного стиля является функция, а наследуемыми ориентирами - смешанные вычисления, комбинаторные вычисления, языковые расширения в направлении приближения свойств стиля к выражению в нем “что делать”.

### 3.1.1 Сравнение языков программирования стиля.

Во всех языках функционального программирования способы описания функций базируются на тех или иных формализмах, созданных в математической логике: лямбда-исчислении (Лисп), комбинаторной логике (Миранда), нормальных алгоритмах Маркова (Рефал), рекурсивных уравнениях Эрбрана-Гёделя (KRC). В более современных языках, таких, как Миранда, используются и некоторые другие достижения математики, например, теоретико-множественная нотация. В чистых языках функционального программирования обычно имеется одна главная операция - применение функции к аргументу, или аппликация, из-за чего их часто называют *аппликативными*. Прозрачная математическая семантика этой операции позволила Д. Тернеру (автору Миранды) говорить о ”семантической элегантности” аппликативных языков (см. пункт 3.4). Не случайно поэтому процесс написания программ на этих языках в чем-то сродни математической деятельности. Обсуждение самих этих языков позволяет легко перейти на теорию предмета, тогда возникает связь теории и практики, необходимая для целей РО.

Для рассмотрения выберем следующие языки: Норе, Миранда, Лисп и FP. Проблемная ориентированность языков определяется функционалами или функциями высшего порядка. Концепция функций высшего порядка возникает из идеи о том, что функции должны иметь тот же статус, что и любой объект данных, так чтобы они сами могли быть входными и выходными данными других функций.

Так, например, в Норе, как и в большинстве функциональных языков, не существует ограничений на количество и сложность подобных функций. Однако в языке FP нет средств для определения пользователем функций высшего порядка. Вместо этого язык обеспечен небольшим набором *встроенных* функций высшего порядка, которые рассматриваются как программно-встроенные блоки. В языке Миранда осуществлен совершенно иной подход к функциям высшего порядка: там нет механизма для явного определения лямбда-выражения, функциональные объекты в нем генерируются частичным применением существующих функций.

**Язык Норе.**

Рассмотрим следующие функции языка Норе:

IncList - увеличивает на 1 каждый элемент списка (функция определяется на родовом (полиморфном) типе данных).

**dec** IncList : list(num)→list(num);

---IncList(nil)<=nil;

---IncList(x::l)<=(x+1)::IncList(l); имеется возможность сопоставления с образцом (::)

MakeStr - отображает каждый элемент списка в список из одного элемента.

**dec** MakeStr : list(char)→list(list(char));

---MakeStr(nil)<=nil;

---MakeStr(c::l)<=[c]::MakeStr(l);

Говорят, что “тип рекурсии” в обоих случаях одинаков. В обоих вышеприведенных примерах определенная функция применяется к каждому элементу заданного списка. В первом случае она имеет следующее определяющее уравнение:

---Inc(n)<=n+1;

во втором -

---Listify(c)<=[c];
(c соответствующими определениями типа). Можно выявить общую структуру, определив функцию высшего порядка, которая используя функции типа Listify и Inc в качестве аргумента, применяет их к каждому элементу заданного списка, выступающему в роли второго аргумента. Подобная функция высшего порядка широко используется и часто называется map. Поскольку мы не знаем наперед тип применяемой функции или тип обрабатываемых элементов списка, то map должна быть объявлена полиморфной функцией (alpha - динамическое определение типа для функций, beta - для элементов списка). Итак,

**dec** map : (alpha→beta)#list(alpha)→list(beta); функция от одной двойки

 аргументов (кортеж).

---map(f, nil)<=nil;

---IncList(F, x::l)<=f(x)::map(f, l);
Теперь, используя map и вспомогательные функции Inc и Listify можно выразить IncList и MakeStr:

IncList(L) ≅ map(Inc, L)

MakeStr(L) ≅ map(Listify, L).
Удобство такого подхода очевидно, однако явное определение подобных функций может быть довольно неудобным. Язык Норе дает нам возможность записывать выражения (лямбда-выражения), значением которого является функция. Например, Inc будет описана так:

**lambda** x=>x+1.

Тогда IncList(L) ≅ map(**lambda** x=>x+1, L).

**Язык Миранда.**

Общий подход, принятый в языке Миранда, очень похож на подход языка Норе. но значительно отличается от него подходом к рассмотрению функций. Миранда - это строго типизированный язык высокого уровня, поддерживающий типы данных пользователя и полиморфизм, но он значительно отличается от Норе подходом к рассмотрению функций. Главное отличие в том, что Миранда - ”карринговый” язык, то есть в нем объекты, значениями которых является функция, строятся путем частичного применения существующих функций, а не с помощью явных лямбда-выражений, как это делается в Норе.

Каждая функция в языке Миранда является по существу функцией высшего порядка. Когда мы на этом языке записываем определение вида
f x y z = …
мы можем обычным образом интерпретировать f как функцию от трех аргументов x, y, z или, как в Норе, в качестве функции от одной тройки аргументов (кортеж). Однако в языке Миранда в действительности f является функцией высшего порядка только от одного аргумента x. Результатом применения f к аргументу E1, который мы запишем в виде fE1, является другая функция, снова только от одного аргумента y. Применение этой функции к следующему аргументу E2 снова дает функцию от оного аргумента z. Полное применение f записывается в виде fE1E2E3. Итак, мы пришли к понятию **карринг**.

*Карринг - обработка функций от* ***n*** *аргументов как конкатенации* ***n*** *функций от одного аргумента.*

Посмотрим, как будет выглядеть функция Inc прибавления единицы к некоторому целому. На языке Миранда она может быть записана в виде частичного применения примитивной функции + к аргументу 1:
(+) 1 (скобки превращают инфиксную функцию + в префиксную).Рассмотрим функцию map. Она имеет следующее определение на языке Миранда:

map f []=[]

map f (x : l)=(f x) : (map f l)
Тогда выражение

map((+) 1) L
увеличивает на единицу каждый элемент списка L.

В языке Миранда все функции должны иметь имена. В языке Норе мы вводим вложенные функции, используя **lambda**-выражения, в Миранде это достигается с помощью определения функции внутри **where**-выражения. Например, функция
map(f 5) L **where** f x y=y+x\*x
прибавляет 25 к каждому элементу L. Заметим, что введенная функция может быть рекурсивной благодаря тому, что имеет имя.

Заметим также, что нет необходимости в том, чтобы все функции были карринговыми, поскольку Миранда предоставляет также механизм для построения кортежей и для декомпозиции этих кортежей с помощью сопоставления с образцом.

**Язык Лисп.**

Лисп является нетипизированным языком обработки списков, в котором программы и данные имеют одинаковое представление.

В языке Норе мы можем непосредственно записать обозначающее функцию выражение с помощью лямбда-выражения. В Лиспе существует идентичный механизм, только функция вводится с помощью специального атома LAMBDA, а не с помощью ключевого слова (**lambda**), как в Норе. Например, функция, увеличивающая на единицу заданное значение x:

(LAMBDA (x) (ADD x (QUOTE 1))).

Используя QUOTE и LAMBDA совместно, мы можем передавать функции (лямбда-выражения) как параметры другим функциям. Рассмотрим применение map, где в качестве функции f используется функция, увеличивающая свой аргумент на единицу. Для того, чтобы эту функцию можно было передать в качестве параметра, определяющее ее лямбда-выражение заключается в кавычки.
Опишем map:

(defun map (lambda (f L)

 (cond ((eq L nil) nil)

 (t (cons (f (car L)) (map f (cdr L)))

 )

))
Тогда увеличение каждого элемента списка (1 2 3) на единицу можно описать так:
(map (QUOTE (LAMBDA (x) (ADD x (QUOTE 1)))) (QUOTE (1 2 3)) ).

Хочется отметить преимущества бестиповости лямбда-исчисления (и Лиспа также, так как он основан на этой теории). Продемонстрируем это на примере, показывающем, что аргумент одной и той же функции может выступать в роли как функции, так и данного (в нашем случае целого числа). Здесь также используется карринговость λ-исчисления.

Рассмотрим функционал Ф2 = λf. λx. f (f x) композиции функции. Так, применение этого функционала к функции g и числу 3 дает нам Ф2 g 3 → g(g 3).
Посмотрим вычисление выражения (Ф2Ф2) g 3.

(Ф2Ф2) g 3 → (λx. Ф2(Ф2x) g 3 → Ф2(Ф2g) 3.
Далее вычисления могут идти двумя путями. Могут использоваться ленивые вычисления (вычисления параметров откладывается до их использования) или энергичные вычисления (сначала вычисляются все параметры, затем только - применение функции).

Итак, при ленивых вычислениях:

→ (Ф2 g)(Ф2 g 3) → **(λx. g (g x)) (g (g 3))** → (g (g (g (g 3)))) = g4 3;

при энергичных:

→ Ф2 (λx. g (g x)) 3 → (λx. g2 (g2 x)) 3 = g4 3.

Первоначально при описании функционала предполагалось применение его к двум аргументам: функции и объекту. Но при вычислениях в качестве x выступает также функция **g**. Бестиповость языка позволяет нам сделать это.

Отметим, что все объекты Лиспа - это S-выражения. Более того, программа на Лиспе сама является S-выражением, то есть не существует явных ключевых слов. Это приводит к одинаковому представлению программ и данных.

**Язык FP.**

Во всех языках, рассмотренных выше, определение функции дает имена объектам, передаваемым этой функции, и описывает затем, что делать с этими переданными аргументами. Поначалу этот подход кажется самым простым для понимания: определение функций принимает вид набора уравнений. Однако одной из наиболее сильных сторон функциональных языков является то, что они поддаются формальным преобразованиям. Одной из форм преобразования - “заменить равное равным”. В языке FP каждая функция выражается именно таким образом.

Отличительной особенностью FP является то, что в нем нельзя выражать функции высшего порядка, определенные пользователем. Все функции высшего порядка являются встроенными - это комбинирующие формы “применить-ко-всем”, левая вставка и правая вставка. Так, в FP существует аналог функции map, работающей со списками: α f обозначает функцию, которая применяет f к каждому элементу в заданной последовательности аргументов.

(α f) : <x1,…,xn>=<f : x1,…, f : xn>
Например, α tl применяет функция выделения хвоста последовательности к заданной последовательности последовательностей (tl - примитив языка):

(α tl) : <<1 2 3>,<4 5 6>>=<tl : <1 2 3>, tl : <4 5 6>>=<<2 3><5 6>>.
Функционалы правая и левая вставка определяются следующим образом:

(/f) : <x> = x базовый случай

(/f) : <x1,…,xn> = f : <x1, /f : <x2,…,xn>> - правая

и

(\f) : <x> = x базовый случай

(\f) : <x1,…,xn> = f : <\f : <x1,…,x-1>, xn> - левая.

Существуют также варианты вставок, имеющие связанные с ними базовые случаи. Они указываются индексами у символов \ и /.

С помощью этих встроенных функционалов мы можем записать нерекурсивный вариант такой функции как n!. Мы увидим, как FP хорошо иллюстрирует мощность функциональной нотации, и в особенности выразительную силу функций высшего порядка.

Рекурсивный вариант:

**def** fac = eq 0→1; \*•[id, fac•(-•[id, 1])]

где • - композиция функций,

 id - тождественная функция (id : x = x),

 1 - константа 1,

 [] - “конструкция”: [f1,…,fn] : x = <f1 : x,…,fn: x>.
Заметим, что в этом определении нет ссылок на объекты, и поэтому нет символов применения функции. Тело состоит целиком из функций и функционалов. При применении fac к конкретному числу выражения становятся более привычным:

fac : 3 ≡ eq 0 : 3 → 1; \* : <3, fac : (- : <3, 1>)> = \* : <3, fac : (- : <3, 1>)>.

Нерекурсивный эквивалент:

**def** fac = /1 \* • ι,

где /1 - функция вставки (со значением в базовом случае, равном 1),

 ι - примитив, при применении к n этот генератор дает последовательность целых от 1 до n.

Тогда fac : 3 = /1 \* : <1, 2, 3> = \* : <1, /1\*: <2, 3>> =

= \* : <1, \*: <2, </1\* : 3>>> = \* : <1, \*: <2, \* : <1, 3>>>> =

= \* : <1, \*: <2, 3>>> = \* : <1, 6> = 6.

### 3.1.2 Неподвижная точка как подходящее обобщение проблемной ориентированности.

Термин "неподвижная точка" встречается в различных разделах математики: в теории рекурсивных функций и алгоритмов, теории множеств, логике, теории автоматов, математическом анализе. Главный теоретический инструмент, использующий данное понятие - это теорема о неподвижной точке (или "теорема Клини").

В языках программирования высокого уровня необходимо иметь возможность записывать определение рекурсивных функций. Мы можем это сделать, так как имеем возможность дать имя любой функции **f**, а затем ссылаться на это имя где угодно в программе (при определённых ограничениях, обусловленных языком), даже в теле функции, названной этим именем, например, **f(f(x)).** Однако в исчислении функции не имеют имён. Они представляются выражениями (впрочем, это единственный объект теории). Поэтому для представления рекурсии необходимо придумать какой-то метод, который позволит функциям вызывать себя не по имени, а каким-то другим образом. Этим способом может быть механизм копирования аппликаций: то есть вид выражения остаётся неизменным, а рекурсия будет осуществляться уже при выполнении, посредством копирования тела функции. Для этого и используется неподвижная точка.

* **Неподвижная точка в лямбда-теории.**

Перейдём непосредственно к решению проблемы рекурсии в исчислении. Для этого представим рекурсивную функцию как функцию, имеющую себя в качестве аргумента. Таким образом, она будет содержать в своём теле ссылки на себя. Рассмотрим данный метод на примере функции факториал:

Обобщим выражение E(F) посредством абстракции

 **f. x. (x=0 | 1 | \* x (f (- x 1)))**.

Обозначим эту абстракцию через G. Возникает уравнение для оператора G: z=Gy.

Чтобы смоделировать рекурсию, необходимо иметь оператор **Y** со следующим свойством

**YG = G(YG)** = **x. (x=0 | 1 | \* x (YG (x-1)))**

Тогда оператор Y для операторного уравнения z=Gy отыскивает неподвижную точку

**YG = G(YG)**.

Тогда рекурсивная функция F с телом Е(F) записывается в исчислении в виде:

**Y (F. E(F)).**

Осталось определить оператор неподвижной точки **Y**. Нам необходимо записать **Y** в виде выражения, так как чистое исчисление не имеет встроенных функций. Вид **Y** встречается при доказательстве теоремы о неподвижной точке в исчислении:

Т**еорема**.  ** f x (f x = x).**

*Доказательство:*

Возьмём **w = x. f(xx).** Тогда искомым **x** является **ww.** Проверим утверждение теоремы: **x = ww= x.f(xx)) w=f(ww)=f x.**

Как видно из доказательства оператором Y является:

**Y =  h. ( x. h(xx))( x. h(xx)).**

Убедимся, что именно это выражение и есть оператор неподвижной точки **Y.** Применим его к произвольной функции **f**:

 **Y f = ( h. ( x. h(xx))( x. h(xx))) f** 

** x.f(x x))  x.f(x x))**  **f ( x.f(x x)) x.f(x x))) = f (Yf)**

Однако данный способ работает только в случае использования так называемого ленивого вычислителя, который “не делает ничего, пока это не потребуется“. В терминах традиционного программирования ленивое вычисление можно соотнести с вызовом по необходимости, в котором все аргументы передаются функции в невычисленном виде и вычисляются только тогда, когда в них возникает необходимость внутри тела функции. Если же использовать аппликативный порядок вычислений или энергичный вычислитель, который соответствует вызову функции по значению (при применении функции к аргументу последний сначала вычисляется, а затем уже передается функции) то самоприменение приводит к зацикливанию:

**Y f → f(Y f) → f(f(Y f)) → …**

 Однако можно модифицировать определение **Y** так, чтобы обойти эту проблему, определив альтернативный **Y**-комбинатор как:

 **Y1 = h. (x. h(y. xxy))(x. h(y. xxy))**

**Y** и **Y1** эквивалентны, поскольку один можно получить из другого с помощью преобразования (λy. xxy **xx[[1]](#footnote-1))**. Если теперь вместо **Y** использовать **Y1**, самоприменение вычисляется, только когда оно применяется к аргументу, соответствующему переменной **y**, например, к целому аргументу функции факториал **F**.

Пример энергичного(по значению) вычисления **F=f.x. (x=0|1|\*x(f(- x 1))).**

**Y1F 3 → (λx. F (λy. xxy))(λx. F (λy. xxy)) 3 →**

**→F (λz. (λx. F (λy. xxy))(λx. F (λy. xxy)) z ) 3 →**

**→ (\* 3 ((λz. (λx. F (λy. xxy))(λx. F (λy. xxy)) z ) 2)) →**

**→ (\* 3 ((λx. F (λy. xxy))(λx. F (λy. xxy)) 2)) → /\* (\* 3 (Y1 F 2)) \*/**

**→ … → (\* 3 (\* 2 (\* 1 (λx. F (λy. xxy))(λx. F (λy. xxy)) 0)))→**

**→(\* 3 (\* 2 (\* 1 1)))→6.**

Так как **w1w10 → F (λy. w1w1y) 0 → 1**, где **w1=(λx. F (λy. xxy)).**

Таким образом, с помощью комбинаторов механизм рекурсии заменяется механизмом подстановки и эквивалентных преобразований.

Реализовать комбинатор Y можно двумя способами: в виде лямбда-выражения или, если все лямбда-выражения представлены в графовой форме, Y просто реализуется установкой циклического указателя (см. п. редукционные реализации).

* **Теорема о неподвижной точке**

Частично упорядоченное множество называется *полной структурой*, если всякое его непустое подмножество имеет как точную верхнюю, так и точную нижнюю грань. Обозначим посредством ⊥ наименьший элемент.

*Определение:* Точка **f** называется *неподвижной (фиксированной)* точкой функции (функционала) ****, если **(f)=f.** Такая точка называется наименьшей, если **е: (e)=e => f≤e** (где **≤** - отношение порядка (возможно частичного) на множестве определения ****).****

Рассмотрим класс функций, имеющих фиксированные точки, и укажем способ нахождения наименьшей.

Возьмём **PP**монотонное отображение на полной структуре **P**. Применим ****к произвольному элементу **α** из **P**. К результату снова применим ****и так далее. Вследствие полноты **P** будет существовать точная верхняя грань последовательности. Если****полученная последовательность значений будет монотонной, то её и стоит подозревать на неподвижность, т.е. **supk{a**?

Следовательно,

**Теорема о неподвижной точке.** Если ****монотонное отображение полной структуры **P** в себя, то **(а) = а** для некоторого **аP**.

*Доказательство:*

Рассмотрим множество **Х = {х | x) х, xP}.** Минимальный элемент ****(основание) существует в силу полноты **Р** и **Х**, таким образом множество **Х** не пусто. Следовательно, существует **а = sup Х**. При этом для любого **x Х,**  так как **а  х** и **** - монотонно, то **а)(х)  х,** следовательно **(а) а**. Аналогично, т.к. ****монотонно, то **а))(а),** что влечет **а)Х** и, значит, **а** ** (а),**так как **а = sup Х.**

Таким образом, **а** ** (а**) ** a** => **а** = **(а).**

Рекурсию можно воспринимать двояко: индуктивно и через решение функционального уравнения. Эти толкования рекурсии совпадают при определенных условиях на функционал.

Монотонный функционал ϕ называется (*цепочно*) *непрерывным*, если для монотонно возрастающей последовательности x0=⊥ и xi+1=ϕ(xi)=ϕi(⊥) справедливо:

suр{ϕ[xi]: i∈N}=ϕ[suр{xi: i∈N}].

**Теорема о совпадении наименьшей неподвижной точки с супремумом цепочки.**

Если ϕ непрерывен, то Yϕ совпадает с супремумом цепочки {xi: i∈N} (x0=⊥, xi+1=ϕ(xi)).

*Доказательство:* Из непрерывности ϕ следует, что супремум цепочки является неподвижной точкой для ϕ:

ϕ[suр{xi: i∈N}]=suр{ϕ[xi]: i∈N}=suр{xi+1: i∈N}=suр{xi: i∈N}.

Осталось доказать, что точка b=sup{xi: i∈N} будет наименьшей неподвижной точкой (по индукции). Предположим, что a - также фиксированная точка ****Докажем, что **b  х**. В базовом случае ** a,** так как ****- основание (минимальный элемент). Предположим, что **** **a,** тогда **** ** a= a,** так как **** - монотонно, а **a** - фиксированная точка по предположению. Так как a **≥ **для любого kто **х** является верхней границей монотонной последовательности. По определению наименьшей верхней границы и **b  a.**

* **Применение теоремы в программировании.**

Надо позаботиться о простоте восприятия монотонности и непрерывности оператора ϕ при толковании рекурсивных объектов посредством теоретико-множественного решения подходящего уравнения. Во многих случаях подходит плоский (дискретный) порядок. Введем порядок на произвольных элементах: x≤y, если (x=⊥)∨(x=y). Например, для N мы, тем самым, определили дискретную решетку:



Она является полной.

*Рекурсивные функции* могут трактоваться как решения функциональных уравнений. Рассмотрим для простоты f: N→N. Индуцируем порядок на функциях на основании плоского порядка на N.

**f ≤ g ⇔Df ∀x [f(x) ≤ g(x)].**

Таким образом, f≤g, если g является продолжением (расширением) функции f (то есть на области определения f эти функции совпадают, и g может быть “шире”). В каком-то смысле g уточняет функцию f. В таблице изображены примеры функций, находящихся в таком порядке. Эти функции монотонны по определению порядка.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **F0** | **F1** | **F2** | **F3** | **…** | **F∞** |
| T | ⊥ | Т | T | T | **…** | T |
| **0** | ⊥ | 0 | 0 | 0 | **…** | **0** |
| **1** | ⊥ | ⊥ | 1 | 1 | **…** | **1** |
| **2** | ⊥ | ⊥ | 4 | 4 | **…** | **4** |
| **…** |  |  |  |  |  |  |
| **N** | ⊥ | ⊥ | ⊥ | ⊥ | **…** | **N2** |
| **⊥** | ⊥ | ⊥ | ⊥ | ⊥ | **…** | ⊥ |

**Пример** (индуктивное толкование рекурсивного соглашения о функции факториала). Рассматривается рекурсивное объявление

**fсt** fас = (**nаt** х)**nаt**:

 **if** х0 **then** 1 **else** х\*fас(х-1) **fi**,

при этом получают следующий функционал ϕ:

В соответствии с нашим определением справедливо для х∈N⊥, i=0,1,…:

fас 0(х)=⊥, т.е. fас 0=Ω.



Тем самым для функции fас i получается не рекурсивное равенство:

Это можно доказать индукцией по i∈N, i=0,1,…:

(1) i=0, тогда утверждение тривиально.

(2) Пусть утверждение правильно для fас i

fас i+1(х)=τ[fасi](х)=



**Пример** (наименьшая неподвижная точка для функционала функции факториала). Функция

fас: N⊥→N⊥

с fас(n)=n! есть наименьшая неподвижная точка ϕ, т.е. наименьшее решение функционального уравнения ϕ[f]=f с

Индуктивное толкование suр{ƒi: i=0,1,…} и толкование неподвижной точки **fiх** ϕ полностью идентичны, если функционал ϕ обладает так называемым свойством непрерывности.

Проверим ϕ на непрерывность

ϕ[sup{fi: i=0,1,...}](x) = ϕ[sup{fi(x-1): i=0,1,...}] =

sup{ϕ[fi]: i=0,1,...}](x) = sup{ϕ[fi(x)]: i=0,1,...}] = sup{faci+1(x): i=0,1,...}=x!

Для сравнения с хорошими функционалами рекурсивных определений приведем пример немонотонного функционала. Пусть функционал ϕ, действуя на функцию f: N→N, выглядит следующим образом:

Пусть f≤g, g≠f, x0 - наибольшая точка (обычный порядок на N) области определения f. Для x≠x0 области определения f τ[f]=τ[g]=⊥.

Для x=x0 τ[f](x0) = 1, а τ[g](x0) = ⊥.

Таким образом, мы получили τ[f](x0) ≥ τ[g](x0). Значит, построенный нами функционал не является монотонным.

*Рекурсивные типы* могут трактоваться как решения уравнений для множеств.

*Пример* (индуктивное толкование рекурсивного объявления типа). Пусть дано следующее объявление типа двоичных деревьев

**sоrt m** = еmрtу(**еmрtу**)⎪соns(**m** lеft, **nаt** rооt, **m** right).

Если записывать ε для единственного элемента типа еmрtу, то для любого множества данных М получим:

Δ(М)={ε}∪{(l,х,r): l∈М\{⊥} ∧ х∈N ∧ r∈М\{⊥}}∪{⊥}.

Это приводит к следующим множествам данных:

М0={⊥},

Мi+1={ε}∪{(l,х,r): l∈Мi\{⊥} ∧ х∈N ∧ r∈Мi\{⊥}}∪{⊥},

т.е.

М1={⊥, ε},

М2={⊥, ε, (ε,0,ε), (ε,1,ε), ...},

М3={⊥, ε, (ε,0,ε), ((ε,0,ε), 0, ε), ...}.

Итак, получаем множество, эквивалентное множеству двоичных деревьев.

**Пример** (трактовка неподвижной точки для двоичных деревьев в рекурсивном объявлении типов). Пусть имеют место определения предыдущего примера. Для множества М с

М=∪Мi

имеет место

М={ε}∪{(l,х,r): l∈М\{⊥} ∧ х∈N ∧ r∈М\{⊥}}∪{⊥}, (\*)

т.е. М есть решение уравнения М=Δ(М). М является также наименьшей неподвижной точкой оператора Δ.

Действительно, оператор Δ является монотонным и непрерывным по теоретико-множественному включению.

Монотонность очевидна. Если M1⊆M2, то Δ(M1)⊆Δ(M2).

Проверим непрерывность: M0=⊥, Mi+1=Δ(Mi), тогда Mi⊇Mi+1.

**U**Δ(Mi)=UMi+1=Δ(UMi).

*Формальные языки* могут трактоваться как решения уравнений для множеств. Рассмотрим язык, определяемый БНФ (см. п. 3.4).

<x>::=1|2<x>

Зафиксируем решетку множеств по включению. Язык можем воспринимать как решение уравнения

Доказательство монотонности и непрерывности оператора τ аналогично случаю для оператора Δ.

Решением уравнения является регулярное множество 2\*⋅1.

Подведем итог использованию неподвижной точки. В рассмотренных примерах практическая роль уравнений незначительна, так как примеры достаточно простые. Это похоже на использование техники пределов для поиска предельного значения простой непрерывной функции. Но, уже приятно то, что с единых позиций рассматриваются рекурсивные функции, типы, формальные языки.

Конечно же, настоящая роль теории неподвижной точки скажется в исследовании классов задач. Например, если регулярные языки будут определяться взаимной рекурсией, то известно, что их решение (наименьшую неподвижную функцию) можно искать аналогом метода Гаусса. Доказательство этого факта легче получить, если обобщить теорию неподвижной точки на декартово произведение. Тогда, выяснив совпадение индуктивной схемы и теоретико-множественной, от преобразований надо требовать свойства равносильности, то есть исследовать на теоретико-множественном уровне.

## 3.2 Редукционные реализации (математический аспект).

Кажется весьма выгодной в отношении обучения тесная связь, существующая между теорией неподвижной точки, проблемой свободных переменных и комбинаторной редукцией. Это позволяет в обучении естественно переходить к математическому аспекту прагматика, а от него - к классу профессионала.

Поэтому в этом пункте мы собираемся рассмотреть различные методы, используемые для реализации функциональных языков. При этом будет использоваться лямбда-нотация для функциональных выражений, которая является достаточно мощной, чтобы выразить все черты функционального языка, и в то же время достаточно простой, чтобы легко описать различные методы реализации без рассмотрения специфики языков. Кроме того, лямбда-нотация происходит из области математической логики, называемой лямбда-исчислением, на котором основаны многие теоретические результаты в области функциональных языков. Основные понятия бестипового лямбда-исчисления приводятся в приложении.

Все функциональные языки основаны на применении функций. В лямбда-исчислении применение функций выражается β-редукцией. Неформально роль β-редукции состоит в том, чтобы заменить каждое вхождение формального параметра функции соответствующим параметром. В основе любой реализации лежит порядок выполнения последовательных редукций и условий, при которых последовательность редукций завершается. Существует два пути выполнения β-редукций: мы можем либо оставить ссылку на формальный параметр баз изменений, но запомнить обозначаемую ею величину, либо выполнить подстановку, которая заменяет ссылку на параметр его фактическим значением. Эти два подхода ведут к реализациям, основанным соответственно на *контексте* и *копировании*.

Примером первого подхода является реализация, основанная на SECD-машине (см. Приложения). Кроме того, что фактические параметры запоминаются в отдельном контексте, эта реализация рассматривает лямбда-выражения “текстуально”, то есть как строки символов. Альтернативой этому является использование двумерного графового представления выражений. Общая модель, использующая такое представление, называется *редукцией графов*. Эта модель основана на копировании, поскольку применение функций к аргументу ведет к копированию графа тела функции с выполнением соответствующей подстановки аргумента.

***3.2.1 Несколько слов о редукции графов****.*

Редукция графов возникла как мощная модель реализации “ленивых” функциональных языков, особенно на параллельных компьютерах. Суперкомбинаторы и вполне задержанные вычисления являются двумя из важнейших методов, пригодных для эффективной реализации редукции графов.

Дадим краткое описание редукции графов. Рассмотрим функциональную программу (λx. AND x x) (NOT TRUE). Представим эту программу деревом ее грамматического разбора

где знак @ стоит на месте функциональной аппликации. Отметим, что карринг отражен в поддереве для (AND x x).

Это дерево содержит два так называемых *редуцируемых выражения* (или *редекса*), которые состоят из применения (аппликации) функции к своему аргументу (аргументам): применение NOT к TRUE и применение λx-абстракции к (NOT TRUE). Чтобы применить λx-абстракцию к ее аргументу (NOT TRUE) мы заменяем верхний узел @ некоторой конкретизацией тела λx-абстракции, в которой вместо каждого вхождения x подставляется указатель на аргумент:

Заметим, что дерево превратилось в граф. Теперь мы можем применить NOT к его аргументу TRUE, получив

Наконец, выполнив операцию AND, получим FALSE.

Каждый из выполненных шагов вычисления называется *редукцией*. Этот простой пример показывает, что

1. Исполнение функциональной программы заключается в *вычислении* некоторого выражения.
2. Функциональная программа допускает естественное представление в виде *дерева* (или, в более общем случае, *графа*).
3. Вычисление сводится к последовательности простых шагов, называемых *редукциями*, которые заключается в применении функции к некоторым аргументам (отсюда термин *редукция графов*). Каждая редукция является локальным преобразованием графа.
4. Лямда-абстракция применяется к своему аргументу путем создания нового наполнения (конкретизации) своего тела, в котором вместо каждого свободного вхождения формального параметра подставляется указатель на этот параметр.
5. Вычисление завершается, когда больше нет редуцируемых выражений. При этом говорят, что выражение имеет *нормальную форму*.
6. Порядок редуцирования редексов не влияет на ответ. Теорема Черча-Россера гарантирует, что стратегия проведения - редукций - безразлична. **Теорема Черча-Россера для β-редукции:** Если P|>βМ и P|>βN, то существует такой терм T, что M|>βT, N|>βT.
7. Редукции можно без опасений применять одновременно, так как они не мешают друг другу. Например, первые две редукции в нашем примере можно выполнить одновременно.

Простейший способ реализации редукции графов заключается в том, чтобы выполнить обход тела лямбда-абстракции и построить его новую копию, подставляя указатель на аргумент вместо каждого свободного вхождения формального параметра. Тело абстракции рассматривается как шаблон и не изменяется в результате этой операции. Однако, этот способ неэффективен по причине большого количества проверок в каждом узле шаблона. Этот способ моделирует редукцию графов и является интерпретирующим.

Более эффективным способом является *компиляция*, сопоставляющая каждому лямбда-телу фиксированную последовательность команд, которую компилятор может построить заранее. Скомпилированный код будет выполняться гораздо быстрее, чем алгоритм обхода дерева (по тем же причинам, по которым скомпилированный код выполняется быстрее интерпретированного кода в обычных языках).

Но, к сожалению, не все лямбда-абстракции поддаются такой компиляции. Возникает проблема свободных переменных.

***3.2.2 Проблема свободных переменных****.*

Рассмотрим лямбда-абстракцию λx. (λy. - x y). При применении ее к некоторому аргументу, например 3, мы наполняем ее тело, создавая тем самым новую лямбда-абстракцию (λx. - y 3). Каждое применение λx-абстракции к новым аргументам создает новые и различные λy-абстракции, тем самым лишая нас надежды скомпилировать единственный фиксированный код для каждой лямбда-абстракции.

Проблема заключается в свободном вхождении x в тело λy-абстракции. Каждый раз мы вынуждены строить новое наполнение λy-абстракции.

Имеются ли какие-либо решения этой проблемы? К счастью, ответом является “да”. Есть два подхода, которые можно выбрать. Один из путей решения данной проблемы - обеспечить прямой доступ к значениям свободных переменных, то есть такой механизм, который поддерживает явный контекст. Этот подход приводит нас к SECD-машине, основанной на *среде*, содержащей значения всех свободных переменных.

Другой подход заключается в том, чтобы преобразовать явные свободные переменные при компиляции. Один из способов добиться этого заключается в трансляции λ-выражений в ***комбинаторную логику*** и в определении машины редукции графов для комбинаторных графов (комбинаторы, как и λ-выражения, могут быть представлены в виде графов)*.* Второй метод состоит в трактовке всех свободных переменных как аргументов дополнительных λ-абстракций; этот метод называется λ-поднятием (*lifting*) [9] [25].

Существуют также более сложные методы, например, использование *суперкомбинаторов* [9].

***3.2.3 Комбинаторная редукция***

Рассмотрим формализацию вычислительных предписаний, которые представляются КОМБИНАТОРАМИ. В аппликативном программировании их роль значительна и в прагматическом плане (т.к. они могут послужить реализацией аппликативного интерпретатора), и в теоретическом плане (т.к. они являются теми функциями, которые порождают функции).

Важно заметить, что при вычислении комбинаторы одно­образно применяются к переменным и к комбинаторам. К тому же они позволяют избежать проблемы свободных и связанных переменных. Это те факты, которые учитывались при создании аппликативного программирования.

При комбинаторной реализации (реализации на основе комбинаторов) каждое λ-выражение преобразуется в эквивалентное выражение, построенное только из применений примитивных функций и комбинаторов, каждый из которых является замкнутым выражением, то есть выражением без свободных переменных. В основе такого преобразования лежит функция, которая абстрагирует свободные переменные, оставляя после этого цепочку применений комбинаторов.

**Комбинаторная логика.**

Рассмотрим теорию, которая лежит в основе этого типа реализации, - комбинаторную логику. Теоретически только два комбинатора, называемые K и S, требуются для представления любого λ-выражения. Но надо сказать, что результирующая комбинация является неэффективной из-за того, что требуется очень большое число таких комбинаторов (и связанных с ним применений).

*Комбинатором* называется λ-выражение, не содержащее свободных переменных. Например, функция *тождества* λx. x является комбинатором и обозначается обычно буквой *I.* Другим примером является комбинатор *фиксированной точки* Y = l h. (l x. h(xx))(l x. h(xx)). Двумя следующими примерами являются *вычеркиватель* K, задаваемый в виде K=λx.λy.x, и *распределитель* S, задаваемый S=λf.λg.λx. fx (gx).

Любое λ-выражение может быть преобразовано в **аппликативное выражение**, то есть в выражение, построенное целиком на применениях функций, где, таким образом, отсутствуют λ-абстракции. Чтобы добиться этого нам достаточно включить в синтаксис выражения два комбинатора S и K в качестве дополнительных констант.

***Комбинаторная логика (КЛ).*** Созидается теория, в которой фиксируется, что значит равенство термов. Равенство фиксируется посредством доказательства.

*Подразумеваемая структура:* A = <A, \*, c1,…,cm> - алгебраическая структура с непустым носителем А; выделенными в нем элементами c1, c2,…, cm, m≥0; двуместной операцией \*. {А - множество имен}

*Язык:* термы, построенные из переменных, K, S посредством операции \*.

*Формулы:* равенства термов.

*Аксиомы:* t = t для атомарных термов.

St1t2t3 = t1t3(t2t3)

Kt1t2 = t1 для произвольных термов.

*Правила вывода:*

**

*Доказуемость:* последовательность равенств α1, α2,…, αn называется доказательством αn, если αi - аксиома, либо имеются j, k < i такие, что αi получается из αj и αk по правилу вывода.

Фактически λ-исчисление и КЛ, определенная выше, являются эквивалентными в следующем смысле. Предлагаемый набор аксиом устанавливает эквивалентность двух комбинаторных выражений MКЛ и NКЛ, обозначаемую MКЛ =КЛ NКЛ. Подобным образом и в λ-исчислении эквивалентность выражений =λ, задается как рефлексивное, симметричное, транзитивное замыкание правил преобразования.

Тогда, если Eλ и EКЛ представляют одно и то же выражение в этих двух логиках, то можно показать, что

MКЛ =КЛ NКЛ ⇔ Mλ =λ Nλ

(если выполняется правило экстенсиональности Mx=Nx→M=N или правило η-конверсии λx. Mx=M).

В этом смысле комбинаторная логика может рассматриваться как модель λ-исчисления.

В нашем представлении мы будем также использовать тождественный комбинатор I (его можно выразить через K и S). Заметим, что единственным механизмом сочетания выражений является применение функций - отсюда термин “аппликативное выражение”.

В этом синтаксисе каждое выражение находится в чисто аппликативной форме и поэтому может быть представлено графом, в котором единственным типом внутренних вершин являются @-вершины. В этом случае не существует λ-вершин.

Сначала заметим, что λ-выражения можно транслировать в комбинаторную форму путем последовательного абстрагирования (выделения) переменных из подвыражения. Функция *comb*, осуществляющая отображение λ-выражений в комбинаторную форму, подробно описана в книге “Функциональное программирование” [24, стр. 294-295]. Так, например, применяя функцию трансляции *comb* к выражению λx.(λy. + x y), мы получим ее комбинаторную форму S (S(KS)(S (S(KS)(S(KK)(K+)) )(S(KK) I) )) (KI). В данном случае мы приходим к “программированию без переменных”. Так как переменные отсутствуют, то нет необходимости рассматривать вопрос об их именах и контексте. Таким образом, проблема свободных переменных оказывается решенной.

Основная привлекательность комбинаторного подхода заключается в его математической элегантности и простоте вычислений, вытекающей из наличия только трех правил преобразования графов (исключая те, что связаны с примитивными функциями). Итак, все, что нам нужно сделать - это определить правила преобразования графов, соответствующее применениям S, K и I, и объединить их с правилами применения примитивных функций. Но, к сожалению, даже очень простое λ-выражение имеет очень сложный эквивалент в КЛ. Поэтому проведем оптимизацию путем расширения набора примитивных комбинаторов S, K и I.

***3.2.4 Комбинаторная редукция с помощью графов.***

Введем дополнительные комбинаторы **B, C** (композитор, перестановщик)

**B f x y → f (x y)**

**C f x y → f y x.**

Тогда в КЛ, имеющей примитивные комбинаторы K, S, I, C, B, выполняются следующие равенства:

S (K t1)(K t2) =КЛ K (t1 t2)

S (K t1) I =КЛ t1

S (K t1) t2 =КЛ B t1 t2

S t1(K t2) =КЛ C t1 t2.

Каждое из этих равенств дает правило для упрощения применения S.

Тогда применение функции **comb** приводит к оптимизированной форме**:**

**comb (λx.(+ 2 (- x 3))) = B (+ 2) (C - 3).**

Теперь рассмотрим представление комбинаторных функций графами. Существует 2 правила.

1) Константу мы представляем в виде дерева с одной вершиной соответствующей самой константе.

2) Применения функций вида **A B** в виде:



Пример: **+ 1 3 = (+ 1) 3**



Пример: **B (+ 2) (C - 3) 4**



Для каждого комбинатора существует правило редукции, которое можно отобразить с помощью графов.

1) Редукция тождественного комбинатора

2) Редукция вычеркивателя



3) Редукция распределителя



4) Редукция перестановщика



5) Редукция композитора



Ïðèìåð êîìáèíàòîðíîé ðåäóêöèè:

**B (+ 2) (C - 3) 4**



B-ðåäóêöèÿ:



2.Ïîñòðîåíèå ãðàôîâ äëÿ âçàèìíî-ðåêóðñèâíûõ ôóíêöèé:

Ïóñòü èìååòñÿ ñèñòåìà ôóíêöèé

**f1 = E1**

**f2 = E2**

 **...**

**fn = En**

Ïðè÷åì **Å1...Ån** ìîãóò çàâèñÿòü îò **f1...fn**, òîãäà íàäî íàðèñîâàòü ãðàô äëÿ êàæäîé ôóíêöèè è ñîåäèíèòü äóãàìè îáðàùåíèÿ ê äðóãèì ôóíêöèÿì ñ âõîäàìè â íèõ.

Ïðèìåð: **f = comb( λx.cons x ( g x)) = S cons g**

**g = comb( λx.cons( \* x x) ( f x)) = S ( B cons square) f**

Îáðàùåíèå **f 2** âûäàñò ïîñëåäîâàòåëüíîñòü 2 4 2 4 2 4...

Ãðàô âûãëÿäèò ñëåäóþùèì îáðàçîì:



Ïîñëåäíåå, ÷òî ìû ðàññìîòðèì ýòî ïðåäñòàâëåíèå ðåêóðñèâíûõ ôóíêöèé ñ ïîìîùüþ ãðàôîâ íà ïðèìåðå ôóíêöèè **factorial**.

Ôóíêöèÿ **factorial** â êîìáèíàòîðíîì ïðåäñòàâëåíèå èìååò âèä

**factorial = S ( C ( B cond ( = 2)) 2) ( S \* ( B factorial ( C - 1))).**

Ãðàô âûãëÿäèò ñëåäóþùèì îáðàçîì.



Определим преобразование графа для представления применения Y-комбинатора. Это нетрудно сделать. Такое определение вытекает из определения фиксированной точки, то есть для любого λ-выражения X имеем YX = X (YX) = X (X (X...X (YX)...)). Таким образом, граф обеспечивает корректное представление:



## 3.3 Модели теорий и типовые языки (системологический аспект).

В этом пункте следовало бы “во всей красе” показать тесную связь техники неподвижной точки с денотационной семантикой лямбда-исчисления по Д. Скотт. Как уже отмечалось, на плоских доменах D’ и D’’ можно определить домен всех монотонных и непрерывных функций f :  [D’ → D’’]. В трактовке функций как абстрактных объектов нет ничего нового, также через комбинаторы (Карри, Чёрча) мы уже привыкли, что выражение f(x) определяется как вычислимая функция и от f, и от x. Но Д. Скотт решил задачу интерпретации: желательно было иметь множество A, в котором вкрадывалось бы его функциональное пространство A→A; по мощностным соображениям Кантора этого, однако, невозможно достичь; в 69 г. Д. Скотт построил модели лямбда-исчислений, урезав A→A до множества непрерывных (в надлежащей топологии; или, как мы показывали, через последовательности) функций на A.

Конечно же, следовало бы рассмотреть пространство D∞, но мы не готовы к этому. Поэтому в пункте 3.3.1 предлагается подход к рассмотрению графиковых моделей лямбда-исчисления по книге Э. Энгелера “Метаматематика элементарной математики”.

Пункт 3.3.2 затрагивает общее значение теоремы о неподвижной точке, которое также следует разработать через аппликативный стиль. Это позволит в обучении непосредственно перейти от системологического аспекта прагматика к классу универсала. Завораживают слова Х. Барендрегта в книге “Ламбда исчисление” (стр. 155): “Как геделевская конструкция рефлексивного предложения, так и теорема рекурсии могут быть истолкованы как теоремы о неподвижной точке для подходящих предполных нумерованных множеств в смысле Ершова”.

### 3.3.1 Î ìîäåëÿõ λ-àëãåáðû èëè λ-òåîðèè.

(по Э. Энглер “Метаматематика элементарной математики”, Москва “МИР”, 1987г.)

Возможности и границы эффективных методов, т.е. алгоритмов, в математике образуют столь же фундаментальный комплекс вопросов, как и возможности и границы аксиоматического описания основных математических структур.

Мы снова применим метатеоретический подход. Мы попытаемся говорить об алгоритмах как об объектах в том же смысле, как раньше говорили о вещественных числах или таких геометрических объектах, как точки или прямые. Основные связи между такими объектами будут введены в качестве элементов языка.

Язык позволяет использовать выразительные средства, во-первых, при аксиоматическом исследовании; во-вторых, при обосновании вычислимости как теории формального оперирования с вычислительными операциями.

**1 Что такое вычислительное предписание?**

*1.1* Алгоритмика начинается с понятия функции, точнее, понятия вычислимой функции. Разумеется, всем знакомо сведение функции к понятию множества (по Дирихле): функция F: A→B - это подмножество F ⊆ A x B, обладающее свойством

∀x∈A ∃!y∈B (<x,y>∈F)

В соответствии с этим реляционным определением функция расположена перед нами как единое целое во всем своем протяжении или *экстенсионале*. Однако вычислительное предписание не соответствует такому идеализированному теоретико-множественному пониманию; оно начинает с понятия функции, которая в применении к аргументу выдает вполне определенное значение. В школе вычислительные предписания принимают главным образом вид выражений. Например,

f1=(x+2)(x-2)

или

f2=x2-4.

Таким образом, f1 ≠ f2, хотя f1(x) = f2(x). То есть в случае вычислительных предписаний наш интерес направлен не на экстенсионал функции, совокупность ее значений, а скорее на ее *интенсионал.* Например, смысл функционального выражения в алгоритмике состоит в его сущности как предписания для исполнения одной или нескольких вычислительных операций.

Наша цель - обосновать алгоритмику как теорию оперирования с вычислительными предписаниями и над ними. Сравним с теорией групп - она работает с понятием умножения элементов группы. В аксиоматической алгоритмике основной операцией является *применение функции к аргументу* или *аппликация* *ƒ\*a*.

Итак, исходный пункт - понятие функции, понимаемой как вычислительное предписание. Теперь мы делаем большой скачок в неизвестное: мы мыслим вычислительные предписания собранными в одну-единственную неструктурированную совокупность. Предусмотренное нами слияние типов родственно точке зрения, принятой в теории множеств, где между множествами также нет никаких различий в уровне. (В теории множеств удается образовать на основе одного-единственного понятия “x есть элемент совокупности y” чрезвычайно многообразно структурированную совокупность математических понятий. Вспомним пару {{x}, {x,y}}. Совсем не такая точка зрения принята в анализе, где функции из R→R отличаются от функционалов (R→R)→R и т.д.)

Бестиповое понимание совокупности вычислительных предписаний делает возможным столь же экономный выбор исходных понятий при аксиоматизации, как и в аксиоматической теории множеств. Например, нам не нужно различать одноместные и многоместные вычислительные предписания (карризация; F=λx.λy.f x y, тогда F\*x=λy.f x y, тогда (F\*x)\*y=f x y. Эта идея восходит от Фреге и была впервые систематически применена Шейнфинкелем.)

***Примеры вычислительных предписаний.***

Мы можем мыслить объект B, удовлетворяющий равенству

Bfgx = f(gx) {звездочки, по соглашению, опускаются}

или

fxyzuv = z (x (zu (yv))) (y (x (zuv)))

Теперь мы выдвинем неограниченную выполнимость этой функциональной абстракции в качестве аксиоматического требования к предметной области.

***Комбинаторные алгебры.***

Пусть A = <A, \*, c1,…,cm> - алгебраическая структура с непустым носителем А; выделенными в нем элементами c1, c2,…, cm, m≥0; двуместной операцией \*. {Итак, А - множество имен}

Пусть t(x1,…,xn) - терм в А, (т.е. построен из констант, обозначающих выделенные элементы, и переменных x1,…,xn с помощью скобок и знака операции \*.

Говорят, что элемент ƒ множества А *представляет* терм t в А, если

∀a1,…,an (ƒ a1…an = t(a1,…,an)).

Алгебра А *комбинаторно полна,* если любой терм t представим в А. В этом случае А называется *комбинаторной алгеброй.*

Она называется *нетривиальной*, если А содержит более одного элемента.

***Достаточно комбинаторов S и K.***

Пусть A = <A, \*, K, S>

 Sxyz = xz (yz)

 Kxy = x.

Тогда А - комбинаторная алгебра.

Таким образом, комбинация, заданная термом t, всегда представляется в комбинаторной алгебре (хотя бы одним) объектом. Ввиду их происхождения мы называем такие объекты *комбинаторами*.

Доказательство обеспечим леммой. Из доказательства следует естественность выбора комбинаторов S, K.

**Лемма**. *Для любого n и терма tn(x1,…,xn), построенного из S, K и x1,…,xn, имеется терм tn-1(x1,…,xn-1), построенный из S, K и x1,…,xn-1, такой, что*

 *tn-1(x1,…,xn-1)\* xn = tn(x1,…,xn).*

*Доказательство:* индукцией по построению

База:

1. Пусть tn≡ K | S | xi, i≠n, тогда имеем tn-1=(Ktn). Действительно, (Ktn)\*xn=tn.
2. Пусть tn≡ xn, тогда имеем tn-1=I=SKK. Действительно, SKK\*xn = Kxn (Kxn) = xn.

 Составные термы:

, тогда имеем tn-1 = St’n-1t’’n-1.

Действительно, St’n-1t’’n-1\*xn = t’n-1xn (t’’n-1xn) = t’nt’’n.

Теперь комбинаторная полнота получается многократным применением леммы: tn(x1,…,xn) = tn-1(x1,…,xn-1)xn =tn-2(x1,…,xn-2)xn-1xn =…= tox1,x2…,xn.

Применим доказательство для построения комбинаторов для выражений.

**1) Bxyz = x (yz)**

Итак,

t2(x,y) = S(Kx)y = t1(x)y,

Осталось t1(x) = S(Kx) = tox

t’o = S = Ksx

t’’o = Kx,

Значит tox = **S(KS)Kx.**

Тем самым B = S (KS) K. Действительно,

Bxyz = S(KS)Kxyz = (KSx)(Kx)yz = S(Kx)yz = (Kxz)(yz) = x(yz).

**2) Cxyz = (xz)y**

Итак,



Итак,

,

Итак, C=S (1S (KS) (2S (KK) S)2)1 (KK).

Действительно, Cxyz = S()1(KK)xyz = (()1x)Kyz = (S(()2x))Kyz = S(K(Sx))Kyz = (K(Sx)y)(Ky)z = Sx(Ky)z = (xz)y.

**3)**

, т.к. K=KKx.

**2 Существование комбинаторных алгебр.**

Карри и Чёрч построили системы математики (добавив логические законы и части математики), которые оказались противоречивыми. Вспомним Фреге. Исторически первым был *теоретико-доказательный подход.* (То есть дань времени - программы Гильберта. Вопрос о существовании таких алгебр заменяется вопросом о непротиворечивости формальной системы; комбинаторная алгебра превращается в комбинаторную логику.

***Комбинаторная логика.*** Созидается теория, в которой фиксируется, что значит равенство термов. Фиксируется посредством доказательства.

*Подразумеваемая структура:* A = <A, \*, c1,…,cm>.

*Язык:* термы, построенные из переменных, K, S посредством операции \*.

*Формулы:* равенства термов.

*Аксиомы:* t = t для атомарных термов.

St1t2t3 = t1t3(t2t3)

Kt1t2 = t1 для произвольных термов.

*Правила вывода:*

**

*Доказуемость:* последовательность равенств α1, α2,…, αn называется доказательством αn, если αi - аксиома, либо имеются j, k < i такие, что αi получается из αj и αk по правилу вывода.

{278 Гоген} Эквациональная логика, которая по существу является логикой подстановки “равных вместо равных”, составляет фундаментальное ядро (???) функционального программирования.

Вопрос о непротиворечивости сводится к доказательству того факта, что невозможно вывести все формулы соответствующего языка. Например, формулу K = S. Действительно, если K = S, то

То есть t1 = t2 (дал произвольные).

***Исчисление редукций.*** Создается теория, в которой фиксируется, что значит редуцируется терм к другому. Фиксируется посредством доказательства.

*Язык:* термы как в комбинаторной логике.

*Формулы:* высказывания о редуцируемости t1 → t1

*Аксиомы:* t → t для атомарных термов.

St1t2t3 → t1t3(t2t3)

Kt1t2 → t1 для произвольных термов.

*Правила вывода:*

*Редуцируемость:* как доказуемость в комбинаторной логике.

***Исчисление конверсий.*** Еще более общее понятие (эквивалентное одновременной конверсии).

*Язык:* термы комбинаторной логики.

*Формулы:* t1 ||→ t2 (||→ семантика этого знака - редуцируется равенством).

*Аксиомы:* t ||→ t для атомарных термов.

St1t2t3 ||→ t1t3(t2t3)

Kt1t2 ||→ t1 для произвольных термов.

*Правила вывода:*

*Конвертируемость:* как доказуемость в комбинаторной логике.

Индукцией по длине доказательства можем доказать, что

*Лемма 1.* *t1→t2 ⇔ t1||→t11[[2]](#footnote-2)\*, t11||→t12, …, t1n||→t2. {Справа - это не тоже, что t1||→t2}.*

Можем получить теорему Чёрча-Россера: если m=n в комбинаторной логике, тогда ∃z (m→z ∧ n→z), причем строится z эффективно по доказательству m=n. (При доказательстве теоремы используются похожие свойства для редукции и конверсии. Для редукции: если (a→m ∧ a→n) ⇒ ∃z (m→z ∧ n→z). Для конверсии аналогично.)

**Теорема.** *S=K недоказуемо в комбинаторной логике.*

*Доказательство:* от противного. Пусть S=K доказуемо. Тогда ∃z (S→z ∧ K→z) по теореме Чёрча-Россера. Тогда существует последовательность конверсий

S||→c11, …, c1n||→z, что невозможно, т.к. S||→S и только.

**3 Модели из термов (эрбрановская модель).**

Подобные модели конкретны в той же степени, что и объекты классической математики вообще. Показав, что комбинаторная логика непротиворечива, мы получили разбиение термов комбинаторной логики на классы эквивалентности посредством доказуемости.

Итак, t~t′ ⇔ t=t′ - можно доказать. Эта эквивалентность, по определению комбинаторной логики (t1=t2 ⇒ zt1=zt2 и t1z=t2z), является даже конгруэнтностью относительно операции \*. То есть если t1=t1′ и t2=t2′ ⇒ t1\*t2=t1′\*t2′. Действительно, t1=t1′ ⇒ t1\*t2=t1′\*t2, t2=t2′ ⇒ t1′\*t2=t1′\*t2′.Тогда t1\*t2=t1′\*t2′.

Итак, определения.

***Модель из термов.***

T = термы комбинаторной логики,

t~t′ ⇔ t=t′ доказуемо. Пусть ‾t = {t′ | t~t′}.

Заметим, что отношение эквивалентности является конгруэнцией для \*, т.е. t1~t1′, t2~t2′ влечет t1\*t2 ~ t1′\*t2′

Система <‾A, \*, ‾S, ‾K> является комбинаторной алгеброй, где

‾A = {‾t | t∈T}:

.‾

Она называется *моделью из термов.*

Сравним конструкцию модели из термов с моделью поля рациональных чисел. Сделаем все последовательно. Рассмотрим натуральные числа N={1,2,…} как абелеву полугруппу с дополнением, т.е.

<N, +>; x+y=y+x; x+(y+z)=(x+y)+z; x+y≠x;

x≠y ⇒ ∃z (x+z=y ∨ x=y+z).

Перейдем к целым числам.

Если n ≤ m ⇒ вычитание не может быть натуральным числом; обозначим его временно символом n Θ m: n Θ m = x ⇔ n = x+m.

Если мы хотим сохранить аксиомы сложения, тогда

m+k = x+(m+k), k∈N

m-k = x+m-k, 1 ≤ k < min(m,n), т.е.

(\*)

Например, (1Θ3) = (2Θ4) = (3Θ5) = (4Θ6) = (5Θ7) = …

Класс эквивалентности (\*) можем задать в виде

(\*\*) (n1Θm1) + (n2Θm2) ⇔ n1+m2 + n2+m1

Рассмотрим теперь всевозможные выражения nΘm, n,m∈N с законом отождествления (\*\*). Посмотрим на сложение

,

т.е. складывать можем покомпонентно.

Посмотрим на вычитание

(nΘm) Θ (n′Θm′) ⇔ yΘz. Как обычно, выразим через сложение (nΘm) = (yΘz) + (n′Θm′) = (y+n′) Θ (z+m′), т.е.

.

Это нужно, так как мы хотим представить (yΘz) = (y+n′+m′)Θ(z+n′+m′) = (n+n′+k)Θ(m+n′+k) = (n+m′)Θ(m+n′). Итак, (nΘm)Θ(n′Θm′) = (n+m′)Θ(m+n′).

Рассмотрим класс эквивалентности, выберем для них формы представления и нормальные формы. Введем символ 0.

Итак, целые положительные представляются (n,0),

целые отрицательные представляются (0,n),

ноль представляется (0,0).

Классы эквивалентности образуют группу по сложению, она изоморфна множеству отрицательных чисел Ζ.

Установление равенства представителей целых чисел устанавливается или приводимостью к нормальной форме или по свойству

(m1,n1) = (m2,n2) ⇔ m1+n2 = m2+n1.

Сравним с установлением равенства двух термов комбинаторной алгебры. Значительно сложнее. Дело в том, что вопрос о доказуемости равенства t = t′ алгоритмически неразрешим.

**4 Графиковые модели - существование комбинаторных алгебр.**

Значение таких моделей, как подчеркивает Д. Скотт, в том, что они действительно делают приведенные теории исчислениями. Через интерпретацию фиксируется “истинность” равенства двух термов.

Попытаемся реализовать элементы комбина′торной алгебры в виде множеств. Будем рассматривать (по Дирихле) график функции ƒ

*F =* {<x, ƒ(x)> | x ∈ Dƒ}.

Применение *F*{a} *=* {b | ∃x∈{a}. <x, b>∈*F*}.

В общем случае M∇N = {y | ∃x∈N. <x,y>∈M}.

Вспомним сложность, возникающую из-за несоответствия мощностей. Поэтому опираемся на конечные множества. Здесь, в конечности множеств, конечно же “твердый орешек” и достижение Скотта.

Итак, M\*N =Df{y | ∃x⊆N. <x,y>∈M ∧ x - конечное}.

Чтобы легче воспринимать функциональные связи пишем (x→y) вместо <x,y>. Попробуем смастерить комбинатор K. (на множествах, вопрос об Универсуме пока открыт)

KMN=M={m | m∈M}

Хотелось бы иметь FM=N, это получим, если F = {(∅→y) | y∈M}.

KM должен давать F, тогда K = {{y}→(∅→y) | y∈Универсума}.

Действительно,

Пусть Универсум формируется на основе множества A={a,b,c}. Кроме A он должен также содержать функции. Рассмотрим K{a}{b} = {{∅→a}}\*{b} = {a}.

Попробуем смастерить комбинатор S: SMNL = ML (NL). Рассмотрим выражение ML (NL), которое дает некоторое {s | “условие”}. Затем по “условию” попробуем восстановить S. {где ML применяют к NL}.

M\*L\*(N\*L) = {непустому значению} = {s | ∃ρ∈N\*L . (ρ→s)∈M\*L} =

= {s | ∃ρ⊆N\*L, ∃τ⊆L . (τ→(ρ→s))∈M} =

= {s | ∃n≥0 ∃r1,…,rn ∈ Универсума ∃σ1,…,σn∈L . (σ1→r1)∈N ∧…∧(σn→rn)∈N ∧ (τ→({r1,…,rn}→s))∈M}.

Тогда за S можем взять множество

S={{τ→({r1,…,rn}→s)} → ({σ1→r1,…,σn→rn} → (τ∪σ1∪…∪σn→s)) | n≥0, r1,…,rn∈Универсума}.

Проверим правильность выбора S на простом примере

.

Чуть изменим SMN{a} = {{a,c}→c}\*{a} = ∅,

.

Легко убедиться, что S≠K. Действительно,

.

***Графиковые модели.***

Пусть A≠∅. Тогда

Go(A)=A,

Gn+1(A)=Gn(A)∪{(α→y) | y∈Gn(A), α⊆Gn(A), α - конечно}.

Пусть

,

и пусть B = 2G(A) - множество-степень множества G(A). {M⊆G(A) ⇔ M∈B}

К = {{y}→(∅→y) | y∈G(A)}. /\*y∈G(A), а не B, но K∈B\*/

S={{τ→({r1,…,rn}→s)} → ({σ1→r1,…,σn→rn} → (τ∪σ1∪…∪σn→s)) | n≥0, r1,…,rn∈G(A); τ, σ1,…,σn∈B и конечны}.

Итак, B = <B, \*, S, K> - комбинаторная алгебра.

Универсум G(a) определяется довольно сложно. Приведем его в общем определении графиковой модели.

Изоморфное вложение ƒ: A→B составляется хитро:

Рассмотрим последовательно

ƒo(a)={a}

ƒ1(a) = {a}∪{{x}→a⋅x | x∈A}

ƒ2(a) = {a} ∪ {{x}→a⋅x | x∈A} ∪ {{x}→({y}→axy) | x,y∈A}

ƒ3(a) =ƒ2(a) ∪ {{x}→({y}→({z}→axyz))) | x,y,z∈A}.

Тогда можем представить, чем является . Легко доказывается, что ƒ(а)\*ƒ(b)=ƒ(a⋅b):

Так как ƒ(b) = {b} ∪ {γ→δ | …}, а α={x}, где x∈A, то

По определению ƒi+1(a) {y | ({b}→y)∈ƒi+1(a)} = {y | y∈ƒi(a⋅b), y∈ƒi-1(a⋅b),…}.

Поэтому

ƒ: A→B является инъективным отображением. Действительно, если a≠b ⇒ ƒ(a)≠ƒ(b), т.к. ƒ(a)∩A = {a}, а ƒ(b)∩A = {b}.

### 3.3.2 Îáùèå àñïåêòû, ñâÿçàííûå ñ íåïîäâèæíîé òî÷êîé.

В изложении многих теорий теорема о неподвижной точке тесно переплетается с понятием вычислимости и самовыразимости, поэтому для начала рассмотрим ее в свете теории вычислимых функций.

Важнейшим вопросом этой теории является вопрос разрешимости (или неразрешимости) проблемы, который в свою очередь неминуемо приводит к вопросу о формализации понятия алгоритма (т.е. процедуры) и вычислимой "с помощью алгоритма" функции (т.е. отображения, задаваемого процедурой). Один из методов такой формализации лежит через определение рекурсивной функции.

Трудности, возникающие в этой связи состоят в выборе рекурсивного определения для описания класса функций некоторого рода. Грубо говоря, рекурсивное определение - это определение, в котором значения функции для данных аргументов непосредственно определяются значениями той же функции для "более простых" аргументов или значениями более простых функций (например, констант). Формализация понятия "более простых" функций как констант удобна, так как тогда рекурсивные определения можно рассматривать как алгоритмы (например, ряд Фибоначчи).

Важно ввести рекурсивное определение так, чтобы описываемый им класс был формальным аналогом неформального понятия всюду определенной функции, вычислимой алгоритмом. (Частным случаем может быть определение примитивнорекурсивной функции [1,стр 22]). Попытки такой формализации наталкиваются на проблему, возникающую в связи с так называемым методом диагонализации [1,стр 25], который дает алгоритмическую функцию, не попадающую в данный формализованный класс. Чтобы избежать этих трудностей можно рассмотреть наравне с всюду определенными функциями и функции не всюду определенные (частичные). Такой подход дает хорошо известные формализации Тьюринга, Клини, Черча, Поста, Маркова, которые являются эквивалентными в смысле получаемого класса функций, дающего наряду с частичными (частичнорекурсивными) функциями и всюду определенные функции (общерекурсивные).

Для каждой функции из полученного класса определяется схема вычисления (набор инструкций). Можно найти алгоритм перечисления всех таких наборов, получаемых данной формализацией (в формализации Тьюринга - наборы четверок, удовлетворяющих условию совместимости), который каждому числу Х сопоставляет набор инструкций стоящий на (Х+1)-ом месте в перечислении. Тем самым мы можем перейти от класса функций к множеству натуральных чисел и проводить исследования на этом множестве.

В этих исследованиях мощным аппаратом и является понятие о "неподвижной точке". Теорема рекурсии (или теорема о неподвижной точке в теории рекурсивных функций) позволяет достаточно просто установить некоторые свойства множеств натуральных чисел. Например, полнота творческих (креативных) множеств и, что всякое продуктивное множество - вполне продуктивно [см.1].

Данные же свойства в свою очередь тесно перекликаются с таким свойством как рекурсивность (обладание рекурсивной характеристической функцией) наиболее важный аспект связанный с разрешимостью и неразрешимостью.

Теорема рекурсии также позволяет перенести понятия рекурсивности и рекурсивной перечислимости из области натуральных чисел в область более сложных объектов, а именно - N-ок чисел. Аппарат неподвижной точки дает простой метод порождения различных патологических структур в теории рекурсивных функций, дает возможность установить соотношения между некоторыми понятиями (например, существование для каждой частичнорекурсивной функции соответствующей примитивнорекурсивной функции особого рода - см. формулировку теоремы Клини в [2]). Теорема о неподвижной точке может быть использована (так же как и в анализе) для доказательства существования многих неявно заданных функций.

Приведем формулировку теоремы о неподвижной точке в теории рекурсивных функций и ее следствие.

Т е о р е м а. Пусть f - общерекурсивная функция, тогда существует число n, такое, что H(n)=H(f[n]), где H(n) - частичнорекурсивная функция, соответствующая n-ому геделеву номеру, а n - неподвижная точка.

С л е д с т в и е. f - произвольная обобщеннорекурсивная функция => существует n, такое, что W[n]=W[f(n)], где W[n] - перечислимое множество, определяющееся как W[n]= Arg ϕ(n)/

Рассмотрим теперь использование теории рекурсивных функций в логике. Это становится возможным путем ввода понятия правильно построенной формулы (конечной цепочки символов конечного алфавита), совпадающим с понятием высказывания в логике предикатов, а также ввода процедуры, позволяющей определить какие цепочки являются правильно построенными формулами (ППФ), а какие нет. Для этого предпринимается кодирование ППФ натуральными числами (аналогично кодированию наборов инструкций в теории рекурсивных функций).

При изучении конкретных логических систем некоторые множества ППФ выделяются как "представляющие особый интерес" (например, "доказуемые", "истинные"). Эти множества называются теориями. ППФ теории оказываются часто эффективно перечисляемыми (например, "доказуемые"). Теория называется аксиоматизируемой, если соответствующее множество ППФ может быть эффективно пересчитано. Аналогично: теория разрешима, если она рекурсивна. Существование рекурсивно перечислимых, но не рекурсивных теорий показывает, что некоторые хорошо известные аксиоматизируемые теории могут быть неразрешимыми ("доказуемые" ППФ логики предикатов образуют неразрешимую теорию).

Иллюстрацией применения теоремы о неподвижной точке в логике могут быть аксиомы утверждающие свою собственную непротиворечивость.

Пусть логическая система с кодированными ППФ на натуральных числах, пусть также определено понятие "непротиворечивости" (множеств ППФ). Существует общерекурсивная функция С, такая, что для любого Х ППФ (с кодовым номером) С(Х) понимается как утверждение о непротиворечивости множества W[x] ППФ.

Пусть дано рекурсивно перечислимое множество W[p] ППФ. Возникает вопрос: можем ли мы добавить к W[p] ППФ с кодовым номером q, которая будет утверждать непротиворечивость W[p] U {q}? Ответ заключен в следующей

Т е о р е м е: каковы бы ни были натуральное число p и общерекурсивная функция С, существует натуральное число - n, такое, что W[n] = W[p] U {C(n)}.

(Доказательство вытекает из следствия [см.1]).

Мы применили теорему рекурсии в логике. Заметим также, что многие синтаксические и семантические понятия можно переформулировать в терминах теории рекурсивных функций.

Понятие неподвижной точки фигурирует также в логическом программировании при описании сути процедурной семантики с помощью семантики неподвижной точки.

Каждое правило А(0) <- A(1),...,A(m) логической программы Р можно рассматривать как m-арную частичную операцию на множестве всех основных атомов:

если правило QA(0) <- QA(1),...,QA(m) принадлежит программе P\*, то эта операция применима к кортежу QA(1),...,QA(m) и QA(0) - ее результат. Тем самым на множестве основных атомов задается алгебра (с не всюду определенными и, вообще говоря, многозначными операциями). Операции этой алгебры задают оператор Фр, переводящий каждую интерпретацию Int в интерпретацию

Фр(Int) = {QA(0)| в P\* существует правило QA(0) <- QA(1),...,QA(m) и {QA(1),...,QA(m)} ∈Int}.

Можно показать, что множество неподвижных точек этого оператора (т.е. множество таких интерпретаций, что Фр(Int)=Int) совпадает с множеством моделей программы Р для "если и только если” - семантики [см.3,стр.305]. Главная модель Int(р) программы Р - это наименьшая неподвижная точка оператора Фр, т.е. наименьшая подалгебра описанной выше алгебры. Для этой модели имеет место

Int(p) = Фkn(∅) (2)

Эта алгебраическая структура проясняет "устройство" моделей программы. Соотношение (2) говорит о том, что каждый атом главной модели "получается" за конечное число шагов с помощью операций данной алгебры и тем самым быть установлена за конечное число шагов. На этом и основана процедурная семантика программы. Семантика неподвижной точки для логических программ аналогична денотационной семантике для "обычных" программ.

## 3.4 Базовое обучение на основе аппликативной среды информационного моделирования в учебнике М. Броя “Информатика. Основополагающее введение.” Часть I.

### 3.4.1 Пример базового обучения на основе аппликативного стиля.

Проблема базового обучения, включающего предъявление взаимодополняющих семантик, нашла свое отражение и в уже упоминавшемся во введении учебнике М. Броя “Информатика. Основополагающее введение.” Совпадение наших задач диктует необходимость более пристального взгляда на позицию автора. При ее рассмотрении уместно воспользоваться следующей точкой зрения - стиль программирования во многом определяется логикой, лежащей в основе [Гоген Дж.А., Мезегер Ж. “Модели и равенство в логическом программировании”, 1991]. В этом ключе в учебнике:

1) в начале исследуются возможности *эквациональной логики*, используется СПТ - система подстановки термов (что естественно для вхождения в аппликативную среду информационного моделирования);

**“Пример** (алгоритмы Маркова). Пусть задана система подстановок R на множестве символов { ∨, ¬, true, false, (, ) } для редукции булевских термов, которые построены только из символов данного множества, к нормальной форме. Эта система состоит из следующих правил:

(1) ¬¬ → ε

(2) ¬true → false

(3) ¬false → true

(4) (true) → true

(5) (false) → false

(6) false∨ → ε

(7) ∨false → ε

(8) true∨truе → true

Алгоритм, определенный данной системой, работает по марковской стратегии корректно для замкнутых булевских термов (т.е. замкнутые булевские термы переводятся в семантически эквивалентные однозначные нормальные формы, состоящие из true и false). Это справедливо даже для замкнутых термов с неполностью расставленными скобками. Однако в этом случае возможны такие применения, которые не являются эквивалентными преобразованиями. Для терма

¬true ∨ true

по марковской стратегии получаются вычисления:

¬true ∨ true → (правило (2) )

false ∨ true → (правило (6) )

true

В общей недетерминистической стратегии дополнительно получают (в смысле постановки задачи корректное) вычисление:

¬ true ∨ true → (правило (8) )

¬ true → (правило (2) )

false “

1. затем происходит расширение эквациональной логики до *многосортной логики* через алгебраическую систему (у М. Броя - “вычислительную систему”). Здесь же вводится денотационная семантика, так как уже появляются функции.
“**Пример** вычислительной структуры ВООL.

***Вычислительная структура ВООL булевских значений***

Множество S типов вычислительной структуры ВООL задано так:

S = {**bооl**}.

Множество F символов функций структуры BOOL задано так:

F = { true , false , ¬ , ∨, ∧ }.

Множество данных В⊥ сопоставлено типу **bооl**, т.е. имеет место

**bооl**BOOL = В⊥ = {L, О, ⊥}

Для лучшей читаемости символы двуместных функций ƒ часто записываются в инфиксной форме. Вместо ƒ(x, y) мы пишем выражение x ƒ y. Употребление инфиксной записи указывается через комментарии при задании функциональной зависимости. Аналогично будем записывать и символ ƒ одноместной функции - иногда предпочтем запись ƒ x вместо ƒ(x). Символы функций обозначают следующие типы функции:

trueBOOL : →В⊥,

falseBOOL : →В⊥,

¬BOOL : В⊥→В⊥, бесскобочный префикс

∧BOOL : В⊥×В⊥→В⊥, инфикс

∨BOOL : В⊥×В⊥→В⊥. инфикс

Причем для а, b ∈ В имеет место (это уже описания функций):

trueBOOL = L, falseBOOL = О,

¬BOOL b=nоt(b), а ∨BOOL b = оr(а,b), а ∧BOOL b = аnd(а,b). “

1. требования реального программирования (динамические структуры) вынуждают перейти к *инициальным моделям* и, таким образом, к *логико-алгебраическим спецификациям*.

**“Пример** (двоичные деревья).

Пусть М - произвольное множество. Множество TREE(M) конечных, слоистых двоичных деревьев над М определяется индуктивно следующим образом:

1. пустое дерево ε есть элемент TREE(M),
2. если t1, t2 ∈ TREE(M), то есть t1 и t2 - конечные деревья, то тройка (t1, x, t2) для каждого x∈M есть конечное дерево. Тогда x называется *корнем* дерева (t1, x, t2), t1 называется *левым поддеревом*, t2 - *правым поддеревом*.

Множество двоичных деревьев над натуральными числами N представляется следующим образом:

TREE(N) = { ε,

 (ε, 0, ε), (ε, 1, ε), ...

 ((ε, 0, ε), 0, ε), ((ε, 0, ε), 1, ε), ...

 (ε, 0, (ε, 0, ε)), (ε, 1, (ε, 0, ε)), ... }”

Однако, нельзя не отметить главные недостатки (по крайней мере с точки зрения рассматриваемых в дипломе позиций). Во-первых, наблюдается диспропорция материала -богатство прагматического уровня и бедность теоретического, служащего обоснованием денотационной семантики рекурсии динамических структур. Во-вторых, теоретическое значение логико-алгебраических спецификаций здесь как бы “зависает”, так как дается инициальная модель и явно фиксируются равенства, заведомо ей удовлетворяющие. При этом не обсуждаются возникающие вопросы логико-алгебраической спецификации:

1. проблема полноты,
2. проблема непротиворечивости.

Итак, учебник М. Броя является примером базового обучения на основе аппликативной среды информационного моделирования с элементами логико-алгебраической спецификации при определении типов данных..

### 3.4.2 Некоторые замечания.

* *Синтаксис: описание формального языка с помощью БНФ*. В одноименном пункте М. Брой обсуждает синтаксис, формальный язык БНФ через индуктивное определение регулярных выражений (через неподвижную точку теретико-множественного решения уравнения). Можно говорить о регулярных языках двумя способами - через грамматики (порождение) и через множество слов (языки). М. Брой выбирает второе и, говоря о БНФ-нотации, вводит понятие регулярных выражений. Посмотрим, как естественно получилось увязать понятия языка и теоретико-множественного восприятия БНФ.

“При определении формальных языков для установления контекстносвободного синтаксиса ЯП в дальнейшем вводится так называемая БНФнотация (форма БэкусаНаура). БНФнотация позволяет, в частности, писать выражения, которые определяют множество последовательностей символов, т.е. формальных языков.

Эти выражения, в силу простоты их построения, называют также *регулярными выражениями*.

**Пример** (десятичная запись чисел как регулярное выражение). Множество последовательностей символов, которые представляют целые числа в десятичной записи без ведущих нулей, может быть описано следующим образом:

0⎪[{}[1⎪2⎪3⎪4⎪5⎪6⎪7⎪8⎪9]{0⎪1⎪2⎪3⎪4⎪5⎪6⎪7⎪8⎪9}\*]

Пусть С - множество символов, называемых *терминальными символами.* Регулярные выражения над множеством символов С определяют формальный язык, т.е. подмножество из С\*. B дальнейшем будут заданы отдельные элементы регулярных выражений и их значение.

(0) *Терминальные символы:* Пусть с есть символ из множества С терминальных символов; тогда с есть регулярное выражение с языком {c}.

(1) *Объединение*: пусть R и Q -регулярные выражения с языками Х и Y; тогда

R⎪Q

обозначает регулярное выражение с языком Х∪Y.

(2) *Конкатенация*: пусть R и Q - регулярные выражения с языками Х и Y; тогда

RQ

обозначает регулярное выражение с языком {xy: x∈X, y∈Y }.

(3) *Опция, итерация, расстановка скобок, пустой язык:* пусть R - регулярное выражение с языком Х; тогда обозначает:

 (a) {R}- регулярное выражение с языком X∪{ε},

 (b) {R}\* - регулярное выражение с языком

 {x1...xn: n∈N ∧ x1,...,xn∈X},

 (c) {R}+ - регулярное выражение с языком

 {x1...xn: n∈N ∧ n>0 ∧ x1,...,xn∈X},

 (d) [R]- регулярное выражение с языком Х,

 (е) {}- регулярное выражение с языком ∅.”

Следует отметить важность такого подхода, так как включается теоретическое обобщение, которое работает во многих случаях - при определении типов данных, рекурсивных функций, формальных языков.

БНФ-нотация как средство описания синтаксиса формального языка активно используется автором во всех следующих главах для определения языка аппликативной среды программирования.

* *Функции.*

Что касается описания функции, следует отметить проводимую параллель с математическим описанием. При этом ставится проблема свободных идентификаторов, которая решается с помощью понятия абстракции функции. Сразу же возникают понятия связывания, λ-абстракции, α-конверсии. (стр. 117, 121, 127).

Объявления обозначений элементов данных и функций рассматриваются как расширение аппликативных языков. При объявлении функции автор подчеркивает единство объявления идентификатора функции, типа и спецификации вычислений.

“...идентификаторы для функций объявляются ("принимаются соглашения") с помощью записи вида

**fсt** ƒ=(**s**1 х1,..., **s**n хn)**s**: Е

и говорят об *объявлении функции* или *соглашении о функции*. В объявлении функции одновременно и вводится идентификатор для функции, функциональность которой задается, и специфицируется ход (процесс) вычисления отображения.

**Пример** (объявления функций)

(1) Идентификатор аbs связывается с функцией ƒ(х)=⎪х⎪ с помощью следующего объявления:

**fсt** аbс=(**int** х)**int**:

 **if** 0≤х **then** х **else** -х **fi”**

При объявлении функций рассматривается функциональная семантика как не рекурсивного, так и рекурсивного объявлений. Подробней о семантике рекурсивных объявлений - ниже.

Рассмотрим, как в учебнике представлено исполнение функции, а именно вызов функции (механизм передачи параметров) и аппликация (применение функций).

“Обозначим через F (функциональное выражение для) функцию с функциональностью

(**s**1,...,**s**n)**s**

и пусть Е1,...,Еn - любые выражения типов **s**1,...,**s**n; тогда

F (Е1,...,Еn)

есть выражение типа **s**, называемое *применением функции* или *вызовом* (*указателем*) *функции*. Значение указателя функции есть ƒ(а1,...,аn), причем пусть ƒ - функция, обозначенная через F, а а1,...,аn - значения выражений Е1,...,Еn. Е1,...,Еn - *фактические параметрывыражения* (или *аргументы*), а а1,...,аn - *фактические параметрызначения*.

В БНФнотации синтаксис указателя функции выглядит следующим образом:

<указатель\_функции>::=<функция>{(<выражение>{,<выражение>}\*)}

При этом предполагаются следующие дополнительные условия: функция, обозначенная через F, должна быть nместной и типы фактических параметроввыражений Е1,...,Еn должны соответствовать типу функции”.

Автор упоминает в этом контексте различные способы вычислений параметров функций (“по значению”, “по ссылке”), а также другие стратегии вычисления указателя функции (“по имени”).

* *Семантика рекурсий*.

Существуют два основных подхода для толкования рекурсивных соглашений о функциях:

-индуктивное толкование,

-толкование через равенство фиксированной (неподвижной) точки.

В обоих толкованиях однозначно устанавливается, с каким отображением сопоставляется идентификатор с помощью рекурсивного объявления. Оба способа рассмотрены на простых примерах. Как уже отмечалось, мы считаем полезным продемонстрировать семантику неподвижной точки на более сложных примерах (грамматики, рекурсивные типы данных и т.п.).

Необходимо отметить также, что, развивая перед читателем идею рекурсивных объявлений с помощью толкования через равенство неподвижной точки, автор использует при теоретическом обосновании данного подхода более общий случай: он использует элементы теории решеток, но рассматривает множество функций как полно упорядоченное, а не как полное. Действительно,

“Наряду с индуктивным толкованием, рекурсивное объявление функции

**fсt** ƒ = (**m** х)**n**: Е

можно истолковывать через решение функционального уравнения

(\*) ƒ=τ(ƒ),

где τ было определено выше, т.е. ищут отображение

ƒ: М→N

с тем свойством, что для всех х∈М справедливо

ƒ(х)=τ[ƒ](х).

Тогда ƒ называют *неподвижной точкой* τ, а (\*)-*функциональным уравнением*, относящимся к рекурсивному объявлению ƒ. То, что функциональное уравнение всегда обладает решением, будет показано ниже. Для этого понадобятся некоторые теоретические понятия упорядоченности.

Функция

ƒ: М→N

называется *монотонной*, если для всех х1,х2∈М имеет место:

х1⊆х2 ⇒ ƒ(х1)⊆ƒ(х2).

Множество монотонных отображений обозначают через

[М→N].

Для произвольного функционала τ не обеспечивается ни существования, ни однозначности неподвижной точки, т.е. решения функционального уравнения. Однако на основе структуры имеющегося ЯП справедливо, что функционал τ является монотонным, т.е. для всех функций ƒ и g имеет место:

ƒ⊆g ⇒ τ[ƒ]⊆τ[g].

Для монотонного функционала τ на вполне упорядоченном множестве с наименьшим элементом гарантируется существование неподвижной точки, т.е. решения приведенного выше уравнения (\*). Отсюда всегда имеется однозначно определенная *наименьшая неподвижная точка* τ.

**Теорема** (KnasterTarski). Если τ-монотонное отображение на вполне упорядоченном множестве А с наименьшим элементом, то уравнение

х=τ(х)

разрешимо и существует однозначно определенная *наименьшая* неподвижная точка для τ.”

Функционал τ вводится следующим образом:

“Для заданной конкретизации β с правой частью объявления, т.е. с абстракцией

(**m** х)**n**: Е

связывается функционал τ, т.е. отображение между множествами отображения:

τ : (М→N)→(М→N).

Функционал τ для функций g: М→N через τ[g] снова дает функцию. Для аргумента а∈М функция τ[g] определяется следующим образом:



причем пусть имеет место β1=β[g/ƒ,а/х]. Различение случаев снова соответствует принципу строгости. Способ записи τ[g](х) означает, что τ для каждой функции g через τ[g] порождает функцию, которая для каждого значения х типа **m** доставляет значение τ[g](х) типа **n**. “

Для частично упорядоченного множества Z непустое подмножество Z0⊆Z называется *направленным*, если для каждой пары х,у∈Z0 существует общая граница z∈Z0, т.е. существует z∈Z0 с х⊆z ∧ у⊆z.

Частично упорядоченное множество называется *полно упорядоченным*, если для каждого направленного множества имеется наименьшая верхняя граница (suрrеmum).

Частично упорядоченное множество называется *полной структурой*, если всякое его непустое подмножество имеет как точную верхнюю, так и точную нижнюю грань.

М. Брой рассматривает более общий случай - полно упорядоченный, а не полный, что, в общем, представляется нам лишним (например трудно доказывается теорема Knaster’s).

* *Типы на основе аппликативной среды информационного моделирования, предназначенной для базового обучения*

Рассмотрим метод разработки учебного материала по теме "Типизация" с учетом выбора за основу теоретической программы учебника M. Broy. Основой для демонстрации понятия типов данных в учебнике служат алгебраические спецификации. А именно, дается определение понятия вычислительной структуры и сигнатуры.

Надо отметить важность момента определения множества основных термов сигнатуры, фактически задающее инициальную модель абстрактной алгебры.

**“Îïðåäåëåíèå**(âû÷èñëèòåëüíàÿ ñòðóêòóðà). Ïóñòü S è F - ìíîæåñòâà îáîçíà÷åíèé; âû÷èñëèòåëüíàÿ ñòðóêòóðà À ñîñòîèò èç ñåìåéñòâà {sÀ: s∈S} òèïîâ äàííûõ sÀ è ñåìåéñòâà {ƒÀ: ƒ∈F} îòîáðàæåíèé ƒÀ ìåæäó ýòèìè òèïàìè äàííûõ. Ìû ïèøåì:

À=( { sÀ : s∈S}, {ƒÀ : ƒ∈F} ).

Ýëåìåíòû s∈S åñòü îáîçíà÷åíèÿ äëÿ òèïà äàííûõ è íàçûâàþòñÿ *òèïû*. Ýëåìåíòû ƒ∈F åñòü îáîçíà÷åíèÿ äëÿ îòîáðàæåíèé è íàçûâàþòñÿ *ñèìâîëàìè ôóíêöèé* èëè *çíàêàìè îïåðàöèé*. Äëÿ êàæäîãî ƒ∈F ñóùåñòâóåò îäíî n∈N òàêîå, ÷òî èìååò ìåñòî: ƒÀ åñòü nìåñòíàÿ ôóíêöèÿ, è ñóùåñòâóþò òèïû s1,s2,..,sn+1∈S òàêèå, ÷òî



Ìîæåò òàêæå áûòü è n=0, ò.å. äîïóñêàþòñÿ è "íóëüìåñòíûå" îòîáðàæåíèÿ. Ýòî òàêèå îòîáðàæåíèÿ, êîòîðûå (ñ ïóñòûì ñïèñêîì àðãóìåíòîâ) ïîëó÷àþò â òî÷íîñòè îäèí ýëåìåíò èç îáëàñòè çíà÷åíèé (ñìîòðè true , false , zårî â ñëåäóþùåì ïðèìåðå), êîòîðûé îïðåäåëåí â òèïå s1 .

**Îïðåäåëåíèå** (ñèãíàòóðà). *Ñèãíàòóðà* Σ - ýòî ïàðà (S,F) ìíîæåñòâ S è F, ïðè÷åì

- S îáîçíà÷àåò ìíîæåñòâî *òèïîâ*, ò.å. èìåí äëÿ òèïîâ äàííûõ,

- F îáîçíà÷àåò ìíîæåñòâî ñèìâîëîâ ôóíêöèé;

ïóñòü äëÿ êàæäîãî ñèìâîëà ôóíêöèè ƒ∈F çàäàíà åå ôóíêöèîíàëüíîñòü **fñt** ƒ∈S+.

Âû÷èñëèòåëüíûå ñòðóêòóðû óêàçûâàþò íà ñòðóêòóðíûå ñâîéñòâà èíôîðìàöèîííûõ ñèñòåì ... Èìååò ìåñòî:

( {sÀ: s∈S}, S, Ò )

îáðàçóåò ñíîâà èíôîðìàöèîííóþ ñèñòåìó ñ

Ò: S→{sÀ: s∈S}, ïðè÷åì Ò[s]=sÀ.

Òàêæå ( {ƒÀ: ƒ∈F}, F, I ) ñ I: F→{ƒÀ: ƒ∈F}, ïðè÷åì I[ƒ]=ƒÀ, ÿâëÿåòñÿ èíôîðìàöèîííîé ñèñòåìîé.”

Çàòåì îïðåäåëÿåòñÿ ìíîæåñòâî îñíîâíûõ òåðìîâ ñèãíàòóðû, ïðèáëèæàþùåå îáó÷àåìîãî ê ïîíÿòèþ èíèöèàëüíîé ìîäåëè àáñòðàêòíîé àëãåáðû.

**“Îïðåäåëåíèå** (ìíîæåñòâî îñíîâíûõ òåðìîâ). Ïóñòü Σ=(S,F) åñòü ñèãíàòóðà. *Ìíîæåñòâî îñíîâíûõ òåðìîâ òèïà* s ñ s∈S îïðåäåëÿåòñÿ ñëåäóþùèì îáðàçîì:

(i) êàæäûé íóëüìåñòíûé ñèìâîë ôóíêöèè ƒ∈F ñ **fñt** ƒ=**s** îáðàçóåò îñíîâíîé òåðì òèïà **s**,

(ii) êàæäàÿ ïîñëåäîâàòåëüíîñòü ñèìâîëîâ ƒ(t1,...,tn) ñ ƒ∈F è **fñt**ƒ=(**s**1,...,**s**n)**s** åñòü îñíîâíîé òåðì òèïà **s**, åñëè äëÿ âñåõ i, 1≤i≤n, ti åñòü îñíîâíîé òåðì òèïà **s**i .

Ìíîæåñòâî âñåõ îñíîâíûõ òåðìîâ ñèãíàòóðû Σ îáîçíà÷èì ÷åðåç WΣ, à ìíîæåñòâî îñíîâíûõ òåðìîâ òèïà **s** ÷åðåç WΣs. Åñëè íå ñóùåñòâóåò íóëüìåñòíûõ ñèìâîëîâ ôóíêöèé, òî ìíîæåñòâî WΣ ïóñòî.

Åñëè èìååòñÿ âû÷èñëèòåëüíàÿ ñòðóêòóðà À ñ ñèãíàòóðîé Σ, òî îñíîâíûå òåðìû â À äîïóñêàþò èíòåðïðåòàöèþ. Ïåðåõîä îò îñíîâíîãî òåðìà t (ïðåäñòàâëåíèÿ) òèïà **s** ê ñîîòâåòñòâóþùåìó ýëåìåíòó à ìíîæåñòâà À íàçûâàþò *èíòåðïðåòàöèåé t â À*. Èíòåðïðåòàöèÿ IÀ ïîýòîìó îçíà÷àåò îòîáðàæåíèå

IÀ : WΣ → {à∈sÀ : s∈S}.

Äëÿ êàæäîãî îñíîâíîãî òåðìà t çàïèñü IÀ[t] îáîçíà÷àåò èíòåðïðåòàöèþ t â À. Ïèøóò òàêæå tÀ âìåñòî IÀ[t]. Èíòåðïðåòàöèÿ ïîëó÷àåòñÿ çàìåíîé â îñíîâíîì òåðìå ñèìâîëîâ ôóíêöèé íà ñîîòâåòñòâóþùèå ôóíêöèè:

IÀ [ ƒ(t1,...,tn) ] = ƒÀ ( IÀ [t1],..., IÀ [tn] ).

Â ÷àñòíîñòè, ({à∈sÀ: s∈S}, WΣ, IÀ) îáðàçóåò èíôîðìàöèîííóþ ñèñòåìó.”

"Если имеется вычислительная структура A с сигнатурой S, то основные термы S допускают интерпретацию в A" [Broy]

Таким образом, в учебнике М. Брой фактически предлагаются АТД с инициальными моделями.

В преддверии совместной работы с М. Брой по подготовке компьютерного учебника-задачника перечислим недостатки:

1. Явно не говорится об АТД и инициальных моделях.
2. Плохо предъявляется семантика неподвижной точки для типов (то есть для множеств).
3. Не демонстрируется работа теоремы для интересных случаев - взаимная рекурсия, "циклические типы".
4. Недостатки синтаксиса: обсуждается параметрический тип, но нет синтаксиса для его предъявления (например, можно было бы предъявить через W - грамматики).
5. По видимому, наряду с СПТ (алгоритмы Маркова) для описания алгоримтов следовало бы применить теорему Черча-Россера для объяснения состоятельности алгоритмов на основе лямбда-теории.
* *Денотационная семантика.*

При рассмотрении языков программирования М. Брой говорит достаточно ясно о двух семантиках.

“Существенно различаются две крайние точки зрения в связи с установлением семантики:

*Функциональная семантика*: описание функций программы, то есть установление отношения между входными и выходными данными (экстенсиональное или наблюдаемое отношение), называют *функциональной семантикой.*

*Операционная семантика*: описание последствий отдельных шагов вычислений, которые имеют место при выполнении программы, называют *операционной семантикой”.*

То есть фактически рассматривается денотационная семантика (аппликативный стиль) и операционная семантика (императивный стиль программирования). В дальнейшем изложении аппликативного стиль рассматривается именно в свете этих двух семантик. Но автор не показывает ясно, на примерах отличия этих двух семантик со стороны стиля программирования, их достоинства и недостатки. А ведь это очень важно для понимания стиля.

Уместно было бы в этом случае привести примеры, показывающие легкость задания денотационной семантики в аппликативном языке, недостатки семантики императивных языков (побочный эффект). У М. Броя, конечно говорится об этом, но в различных главах, и акцент на это не делается.

Хочется привести пример книги Филд, Харрисон “Функциональное программирование”, где авторы совершенно замечательно выделяют данный случай.

Так в этой книге отмечается, что в последнее время появилось довольно большое число языков, преуспевших в отходе от формы традиционного программирования; примером такого рода могут служить *аппликативные*, или *функциональные,* языки. Функциональные программы строятся из “чистых” математических функций, которые по сравнению с функциями многих императивных программ свободны от побочных эффектов (т.е. их вычисление не может изменить среду вычислений). Из-за этого нет возможности программировать “помощью эффекта”, так что сама величина, которая должна быть вычислена программой, и сама программа редуцируются к одному и тому же результату. Выполнение программы тогда становится процессом изменения формы требуемой итоговой величины так, что “8+1” можно заменить на “9”; при этом обе они будут обозначать одну и ту же величину.

Программирование с помощью функций не только возможно, но также вполне естественно и приводит к программам почти всегда более кратким, нежели аналогичные императивные программы, более простым в написании и объяснении.

Авторы проводят сравнения вычислений в императивных и аппликативных языках, показывая сущность различий:

“Функциональность (прозрачность по ссылкам - вычисление выражения просто изменяет форму выражения, но не изменяет его величину) определяет различие между математическими функциями и функциями, которые можно написать на императивных языках программирования, таких, как Паскаль, поскольку эти языки дают функциям возможность ссылаться на глобальные данные и разрешают применять (разрушающее) присваивание, что может привести к изменению значения функции при повторном ее вызове. Такие изменения часто именуются *побочными эффектами.* Это приводит к тому, что функцию трудно использовать, поскольку необходимо рассмотреть текущую величину глобальных данных. Это же в свою очередь требует рассмотрения *истории* вычисления. Говорят, что императивные языки являются *нефункциональными*. Для иллюстрации их нефункциональности рассмотрим пример программы, написанной на языке Паскаль:

**program** example(output);

**var** flag : **boolean**;

**function** f (n : **integer**) : **integer**;

**begin**

 **if** flag **then** f:=n

 **else** f:=**2**\*n;

 flag:=**not** flag

**end**;

**begin**

 flag:=**true**;

 writeln(f(1)+f(2));

 writeln(f(2)+f(1))

**end.**

После выполнения этой программы на терминал будет выведены два числа: 5 и 4. Однако, это довольно странно, поскольку с математической точки зрения коммутативность сложения позволяет заменять x+y на y+x для любых x и y.

Однако, дело здесь не в изменении самой функции ‘+’. Проблема состоит в том, что функция f сильно отличается от математических функций. Следует обратить внимание на то, что величина глобальной переменной flag в нашей программе на языке Паскаль имеет возможность изменяться, и именно это уничтожает свойство функциональности языка. Значение же элементарной функции ‘+’ не изменяется. Источником таких проблем в программах на языке Пвскаль является операция *присваивания* flag:=**not** flag, изменяющая величину flag.

Тот факт, что в программе не содержится операции присваивания, является характеристикой функциональных программ. Вместо представления о переменной как о величине, которая может периодически изменяться путем присваивания ей различных значений, переменные в функциональной программе рассматриваются как переменные в математике: если они существуют, то, следовательно, имеют какую-то величину, и эта величина не может измениться. Вместо программы, являющейся последовательностью императивов, описывающих, **как** компьютер должен решать задачу, основываясь, на состоянии, изменяемом шаг за шагом (т.е. на изменении переменных в результате присваивания), функциональная программа описывает, **что** должно быть вычислено, то есть является просто выражением, определенным в терминах заранее заданных функций и функций, определенных пользователем. Величина этого выражения является результатом программы. Таким образом, тут отсутствует понятие состояния программы и предыстории ее вычислений.”

# 4 Класс обучаемых Универсал: предъявление понятия Выразимость.

В поставленной задаче дипломной работы необходимо было на основании теоремы компактности Мальцева-Геделя выявить ключевые понятия для организации среды саморазвития категории начинающего. Анализ этой задачи, а главное, коллективный итог работы по предметной области [31][32] позволили точнее определиться с моделью ПО в компьютерном учебнике-задачнике (рис. 2, п. 1). Конкретная работа по увязыванию теории предмета с прагматикой задания для разных стилей (аппликативный, логический, объектно-ориентированный) привели к четкому увязыванию на уровне учебного материала математического (реализаторского) аспекта прагматика с классом профессионала, а системологического аспекта прагматика с классом универсала. Отсюда последовали серьезные уточнения в распределении материала. Именно, среда саморазвития для начинающего профессионала выстраивается на общих свойствах понятия алгоритм, а для начинающего универсала - на общих свойствах понятия метатеории (то есть теории, учитывающей язык, определение истинности, определение канонической модели). Наконец, нашла свое место тема - сравнение стилей по выразимости. Её заполучил программистский аспект универсала. Самое забавное, что теория моделей (с теоремой компактности) вышла на максимальный уровень обучения - системологический аспект универсала. Такое положение кажется вполне подходящим, учитывая нашу нацеленность на базовое обучение.

Таким образом, пункт 4 удается сформулировать шире, чем требовалось в задаче дипломной работы. Рассматривается в целом класс обучаемого универсал. Структура пункта 4 построена по настоянию научного руководителя, который хотел иметь итоговое описание состояния дела с учебным материалом для универсала на 96г.

## 4.1 Программистский аспект.

Так как этот вопрос не входил в задачу диплома, то мы обращаемся к перечислению рабочих материалов по вопросу сравнения стилей по выразимости:

В дипломе [27] рассмотрен императивный стиль, но безотносительно к заданию практикума.

В дипломе [29] рассмотрена среда саморазвития для логического стиля. Предъявлен также материал для информационной системы логического стиля.

В дипломе [28] рассмотрена среда саморазвития для аппликативного стиля. Предъявлен материал для информационной системы аппликативного стиля.

В дипломе [32] рассмотрена среда развития для логического стиля, а также элементы учебного материала для профессионала.

В дипломе [31] рассмотрена среда развития для ООП

В моем дипломе рассматривается среда развития для аппликативного стиля, а также элементы учебного материала для универсала.

В дипломе [30] общее состояние компьютерного учебника-задачника на 95г.

## 4.2 Математический аспект.

Для класса универсал обсуждается понятие Выразимость на основании Современного Аксиоматического Метода (САМ). Тогда релизационным аспектом являются вопросы, связанные с существованием реальных вычислителей, работающих для языков, ориентированных на “Что делать”. Следующие два пункта являются дополнениями к п. 3.4 диплома [32] и пунктов 3.2.2, 3.2.3, 3.2.4 диплома [31].

### 4.2.1 Èíèöèàëüíàÿ àëãåáðà.

Инициальные модели позволяют отвлечься от какого-либо конкретного представления, то есть обеспечивают *абстракцию*. В частности, концепция инициальности позволяет с большой гибкостью и удобством обращаться с абстрактными типами данных (см. Сашин диплом), в том числе и в языках логического программирования. Её можно использовать и для определений функций и отношений над встроенными типами. Инициальные модели создают также основу концептуального мира программы, будучи “замкнутыми мирами” или “стандартными моделями”. В частности, они дают реализатору четкий стандарт корректности, а программисту - ясную модель того, что можно ожидать от программы. Кроме того, стандартная интерпретация в конкретных семантических областях некоторых сортов, функций и операций позволяет использовать стандартные алгоритмы для решения конкретных задач в таких областях. Это дает большую гибкость, поскольку для других сортов можно по-прежнему использовать инициальность.

Наконец, инициальность является “универсальным” свойством (в смысле существования единственного отображения, удовлетворяющего определенным условиям). Между тем хорошо известно, что использование универсальной характеризации объектов часто позволяет построить более ясную математическую теорию с более простыми и концептуальными доказательствами по сравнению с использованием конкретных конструкций для этих объектов.

### 4.2.2 Óïîðÿäî÷åííî-ñîðòíûå òåîðèè.

Определяя истинность за счет сужения моделей, пользуемся тем фактом, что для описания программистских конструкций достаточно многообразия. Однако, чтобы приблизиться к реальному программированию, необходимо ввести многосортные модели. Гоген [9] показал, что использование в логическом программировании *упорядоченно-сортной логики и модели* формирует новый мощный подход, альтернативный синтаксическим и операционным подходам, используемым обычно в логическом программировании. Более того, упорядоченно-сортная теория моделей поддерживает современный аксиоматический метод языком спецификации EQLOG, являющимся также и языком программирования сверхвысокого уровня. Также существуют языки спецификаций OBJ2 и FOOPS, объединяющие различные языки программирования и основанные на упорядоченно-сортных теориях.

Оказывается, что логическое программирование может очень много выиграть от использования таких понятий теории моделей первого порядка, как

* упорядоченно-сортная логика и модели;
* инициальные модели;
* интерпретация определенных фиксированных сортов, функций и отношений в фиксированных моделях;
* настоящее семантическое равенство.

Эти инструменты (стандартные для теории абстрактных типов данных) составляют привлекательную альтернативу подходам, которые обычно применяются в логическом программировании. Они формируют новый мощный подход, в рамках которого можно использовать

* абстрактные типы данных, определяемые пользователем;
* встроенные типы данных;
* комбинацию логического и функционального программирования;
* программирование с ограничениями, позволяющее применять стандартные алгоритмы для стандартных задач, таких, как линейное программирование.

Различные виды логик охватывают различные аспекты языков программирования. Функциональный аспект программирования охватывается эквациональной логикой. Строгая типизация - многосортной логикой. Логическое программирование охватывается хорновой логикой. Объектно-ориентированное программирование охватывается рефлексивной логикой, в которой тексты программ образуют встроенный абстрактный тип данных.

Развитие логического программирования наводит на мысль, что правильный путь к объединению различных парадигм программирования заключается в выявлении лежащих в их основе логик, объединении их и использовании объединенной логики в качестве основы языка. Это позволяет выбирать нужные черты различных ЯП и смешивать их. Например, комбинация функционального и логического программирования охватывается хорновой логикой с равенством. Комбинация функционального и объектно-ориентированного программирования охватывается рефлексивной эквациональной логикой. В настоящее время существует мнение, что рефлексивная упорядоченно-сортная хорнова логика с равенством хорошо подходит для объединения функциональной, реляционной и объектно-ориентированной парадигм в едином простом языке программирования, обладающем также мощными возможностями для работы с базами данных.

Многие из этих комбинаций уже реализованы в языках FOOPS, OBJ2, Eqlog. Язык OBJ2 основан на упорядоченно-сортной эквациональной логике первого порядка. Язык Eqlog основан на упорядоченно-сортной хорновой логике с равенством. Язык FOOPS является комбинацией языков функционального и объектно-ориентированного программирования и основан на рефлексивной упорядоченно-сортной эквациональной логике первого порядка.

Сущность упорядоченно-сортной логики заключается в том, что на сортах задается частичное упорядочение, определяющее подсорта, которое семантически интерпретируется как теоретико-множественное включение носителей модели. Кроме того, предусматривается перегрузка операций, которая интерпретируется как ограничение функций на подсорта. Упорядоченно-сортная логика не намного сложнее многосортной, и практически все результаты, полученные для многосортного случая, без особых сложностей обобщаются на упорядоченно-сортный.

Итак, в рамках системологического аспекта мы воспринимаем логический стиль через более общее понимание, заключающееся в том, что существуют языки, поддерживающие современный аксиоматический метод .

Тогда уровнем программирования на этих языках для логического стиля может быть программирование ЛАС (пользуясь аксиоматическим методом), а уровнем спецификации может выступать *упорядоченно-сортная теория моделей* как расширение логического стиля без добавления ограничений.

Итак, мы понимаем современный аксиоматический метод как:

* возможность варьировать истинностью, рассматривая только определенный класс моделей;
* наличие языка, его поддерживающего.

Можно сказать, что современный аксиоматический метод - это сущность математики, которая изучает свои собственные методы предъявления знания, рассматривающая саму себя как язык.

## 4.3 Системологический аспект.

Ñîâðåìåííûé àêñèîìàòè÷åñêèé ìåòîä ëîãèêî-àëãåáðàè÷åñêèõ ñïåöèôèêàöèé ïîääåðæèâàåòñÿ â ïðîãðàììèðîâàíèåì ÿçûêàìè OBJ, EQLOG, FOOPS. Ïîýòîìó, çäåñü îñìûñëèâàòåñÿ â êà÷åñòâå áóäóùåãî ó÷åáíîãî ìàòåðèàëà êëàññè÷åñêàÿ òåìà òåîðèè ìîäåëåé äëÿ ÿçûêà èñ÷èñëåíèÿ ïðåäèêàòîâ ïåðâîãî ïîðÿäêà - òåîðåìà êîìïàêòíîñòè Ãåäåëÿ-Ìàëüöåâà.

Цель этого пункта - поговорить о вычислительных возможностях языка исчисления предикатов, используя минимальный багаж знаний. Например, в серии “Популярные лекции по математике” нам очень интересно было посмотреть выпуски Успенского В. А. “Теорема Геделя о неполноте” и“Машина Поста”**.** Действительно, в первой книге отмечается, что для понимания доказательства теоремы Геделя о неполноте необходимо знакомство с простейшей терминологией теории множеств (словами “множество”, “функция”, “область определения” и т.п.) и некоторая привычка к восприятию математических рассуждений. Так что “оно доступно и подготовленному школьнику” [26]. Во второй книге понятие программирования вводится через понятие машины Поста, через ее функционирование. При этом по ходу дела вводятся понятия алгоритма и программы, диаграмм и блок-схем, состояний и переходов из одного состояния в другое. Рассматриваются проблемы разрешимости и применимости алгоритма.

Попробуем следовать этому примеру, то есть, понимая уровень обучаемого (универсал), мы должны показать ему границы выразимости языка при использовании простейших понятий - нам нужно найти эквивалентные формулировки соответствующих понятий на более простом уровне. Например, Барвайс [21]: излагает теорему компактности сначала для логики высказываний. Там все очень просто и легко для понимания, поэтому уже на таком уровне мы можем показать то, что хотим.

Главное, язык предикатов хорош, но он ограничен, с одной стороны, но есть хороший вычислитель, с другой стороны. Предполагаем также, что уже существует описание среды саморазсвтия для идеи вывода [29].

### *4.3.1 Теория моделей*

Теория моделей представляет собой раздел математической логики, изучающий связи между формальным языком и его *интерпретациями*, или *моделями*.

*Пример*.

Приведем пример моделей, представляющих собой структуры, встречающиеся в математике. Циклическая группа пятого порядка, поле рациональных чисел - модели рассматриваемого типа.

Можно было бы прямо сейчас заняться изучением этих моделей, не привлекая формального языка. Тогда мы окажемся в области универсальной алгебры, предметом которой являются гомоморфизмы, подструктуры, прямые произведения и т. п.

Чтобы добраться до теории моделей, вводится в рассмотрение *формальный язык* - язык логики предикатов первого порядка с равенством*.* Формальный язык требуется нам для того, чтобы в наших рассуждениях о моделях мы могли использовать и предложения соответствующих формальных систем. С этой целью мы каждой паре, состоящей из предложения и модели, ставим в соответствие одно из истинностных значений - *истинно* или *ложно.* Вводимое таким образом *понятие истинности* играет роль моста, связывающего формальный язык с его интерпретацией посредством моделей.

Типичные результаты теории моделей содержат некоторую информацию негативного характера о выразимости логики предикатов первого порядка. Один из классических результатов - *теорема компактности*, принадлежащая Гёделю[1930] и Мальцеву[1936]: если каждое конечное подмножество имеет модель, то и все множество имеет модель. С помощью этой теоремы было показано, что многие интересные свойства моделей нельзя выразить, используя только множества предложений логики первого порядка. Надо заметить, что техника теории моделей развивалась большей частью именно для логики первого порядка.

Единство теории моделей придает различение синтаксиса и семантики, проводимое во всей этой теории. Синтаксис имеет дело с чисто формальной структурой языка (например, длина предложения или совокупность символов, входящих в него - вопрос семантический). Семантика же изучает интерпретацию, или значение формального языка. Истинность или ложность предложения в модели - вопрос семантический.

Большая часть теории моделей исследует взаимодействие синтаксических и семантических идей. Именно поэтому теория моделей важна для информатики, так как мы вкладываем в компьютер синтаксические вещи, а предъявляем знания (семантику). Мы видим реальную связь формального языка и его интерпретации:

*Пример*.

Знания о решении систем линейных алгебраических уравнений могут быть представлены в компьютере в виде программы на некотором языке программирования (используем формальный язык).

Универсала интересуют прежде всего границы инструмента информационного моделирования, то есть *выразимость*: что можно выразить с помощью данного инструмента, а что нельзя. В нашем случае инструмент - это логика предикатов первого порядка.

### *4.3.2 Теория моделей логики высказываний*

Теперь изложим теорию моделей для одного очень простого формального языка - логики высказываний. На этой “игрушечной” теории моделей мы сможем легче уяснить для себя очень важный результат - теорему компактности - на основе простейшего формального языка и классических примеров, облегчающих понимание теории для более сложных логик.

Язык логики высказываний Ф - множество простых утверждений + построенные из них составные утверждения. Интуитивная интерпретация для этих утверждений - “возможный мир”, где каждое утверждение либо истинно, либо ложно. Мы хотим заменить интуитивные интерпретации совокупностью точных математических объектов. Таким образом возникает отображение F: Ф→Boolean. Вместо этого мы будем рассматривать в качестве модели - произвольное подмножество А множества Φ: S∈А означает, что простое высказывание S истинно, S∉А - что простое высказывание S ложно.

Опишем логику высказываний как формальный язык. Наш язык содержит следующие символы: логические связки ∧(и) и ¬(не), скобки (,), непустое множество Ф высказывательных символов.

*Определение (высказывания относительно множества Ф).* Можно предложить различные определения высказываний. Мы приведем некоторые из них.

|  |
| --- |
| **Индуктивное** |
| 1. S∈Φ - высказывание
2. ϕ - высказывание ⇒ ¬ϕ - высказывание
3. ϕ, ψ - высказывания ⇒ ϕ&ψ - высказывание
4. ничто другое не является высказыванием
 |
| **Рекурсивное** |
| . ϕ - высказывание |
| **Теоретико-множественное** |
|  |

L - язык, построенный на основе высказывательных символов из Φ.

Теперь мы готовы посредством определения истинности высказывания в модели перекинуть мост между языком L и его моделями.

*Определение*.Истинность высказывания ϕ в модели. (A|=ϕ: *ϕ выполняется в А*, или *А - модель для ϕ*).

1. ϕ∈Φ **⇒** A|=ϕ ⇔ S∈A
2. ϕ = ¬ψ **⇒** A|=ϕ ⇔ A|=ψ не имеет места
3. ϕ = θ&ψ **⇒** A|=ϕ ⇔ A|=ψ, A|=θ

Особенно важным видом высказываний являются *истинные высказывания.* Высказывание ϕ называется *истинным* (|=ϕ), если ϕ - истинно во всех моделях языка L. На первый взгляд кажется, что, отвечая на вопрос “Является ли высказывание ϕ бесконечного языка L истинным?”, нужно исследовать в общем случае несчетное множество бесконечных моделей А. Это происходит из-за того, что истинность - семантическое понятие, определяемое с помощью моделей. Однако, существует простой и регулярный способ распознавать в конечное число шагов, является ли это высказывание истинным или нет .

Эта процедура основана на синтаксическом понятии *тавтология*.

Пусть ϕ высказывание такое, что все высказывательные символы содержатся среди n+1 символов S0,S1,…Sn. Пусть a0,a1,…an - последовательность из t и f. Назовем эту последовательность - *интерпретацией*.

*Определение*. Значение высказывания ϕ при интерпретации a0,a1,…an.

1. ϕ - высказывательный символ Sm, m≤n **⇒** значение ϕ есть am
2. ϕ = ¬ψ **⇒** значение ϕ - противоположно значению ψ
3. ϕ = θ&ψ **⇒** ϕ=t, если ψ=t и θ=t, иначе ϕ=f

Отметим, что два последних определения сходны. Единственное существенное различие между ними состоит в том, что первое определение оперирует с бесконечной моделью А, в то время как это - только с конечной интерпретацией a0,a1,…an.

*Определение*. ϕ - тавтология, (|=ϕ),

если ϕ имеет значение t для любой интерпретации a0,a1,…an.

Отметим, что знак ‘|=‘ оперирует с семантическими понятиями, а ‘|-’ - с синтаксическими.

А та процедура, о которой мы говорили выше, очень проста - нужно просто построить таблицу истинности! Применение этих таблиц позволяет построить простую и чисто механическую процедуру, определяющую, является ли высказывание ϕ тавтологией, - надо просто выписать таблицу истинности для ϕ и проверить, при всех ли интерпретациях ϕ принимает значение t.

**Теорема** (о полноте).

|-ϕ ⇔ |=ϕ

(т.е. высказывание является тавтологией тогда и только тогда, когда оно истинно. Эта теорема фактически показывает связь синтаксиса с семантикой.

*Доказательство*:

Пусть ϕ - высказывание и его высказывательные символы находятся среди S0,S1,…Sn.Рассмотрим произвольную модель А.

Для m=0,1,...,n положим am=t, если Sm∈A

am=f, если Sm∈A.

Таким образом, получим некоторую интерпретацию. Докажем теперь вспомогательное утверждение.

*Утверждение 1*. А|=ϕ ⇔ высказывание ϕ принимает значение t при интерпретации a0,a1,…an.

*Доказательство*: по индукции определения высказывания.

1. ϕ- высказывательный символ ⇒ очевидно.
2. Если утверждение верно для ψ и θ, то оно верно и для ¬ψ, θ&ψ.

⇒ Если ϕ - тавтология, то по утверждению ϕ - истинно.

⇐ т.к. любая интерпретация a0,a1,…an может быть получена из некоторой модели А, то заключаем, что если ϕ истинно, то оно - тавтология.

Таким образом, с помощью нашей разрешающей процедуры для отношения |-ϕ можно также выяснять, истинно ли |ϕ.

### *4.3.3 Теорема компактности в логике высказываний*

Введем теперь понятие формального вывода в нашей логике. *Правило отделения* (или *модус поненс*) гласит:

Из высказываний ψ и ψ→ϕ выводится ϕ.

Рассмотрим теперь конечное или бесконечное множество высказываний Σ в языке L.

*Определение*. Множество высказываний Σ *выполнимо*, если имеет хотя бы одну модель.

*Определение*. Множество высказываний Σ *противоречиво* (несовместно), если Σ|=ϕ для всякого высказывания ϕ, иначе - *непротиворечиво* (совместно).

*Утверждение 2*. Σ противоречиво <=> Σ |- S&¬S для произвольного S∈L.

Покажем теперь самую важную теорему логики высказываний, дающую критерий выполнимости множества Σ.

**Теорема. (**Обобщенная теорема о полноте.)

*Множество высказываний Σ языка L непротиворечиво ⇔ когда оно выполнимо*.

Из этой теоремы можно вывести чисто семантическое следствие теорему компактности.

*Следствие***. (Теорема компактности.**)

*Если любое конечное подмножество Σ выполнимо, то всё Σ выполнимо (Если любое конечное подмножество Σ имеет модель, то и все Σ имеет модель. )*

*Доказательство*:

Пусть любое конечное подмножество Σ выполнимо, а Σ - невыполнимо.

Тогда по обобщенной теореме о полноте Σ противоречиво. Тогда по утверждению 2 Σ |- S&¬S . Но в выводе высказывания S&¬S участвует только конечное множество Σ0⊂Σ. Таким образом, Σ0 |- S&¬S.

Но тогда по утверждению 2 Σ0 - противоречиво => по обобщенной теореме полноте Σ0 невыполнимо. Получили противоречие.

Заметим, что обратное верно. ( Для любого выполнимого множества - любое его конечное подмножество выполнимо).

*Альтернативная формулировка теоремы компактности.*

Если Σ|=ϕ, то ∃ конечное Σ0⊆Σ: Σ0|=ϕ.

**Стандартный пример простого применения теоремы компактности для логики высказываний.**

Рассматривается задача о раскраске бесконечной карты k красками так, чтобы каждая страна на карте имела цвет, отличный от всех ее соседей.

*Утверждение*.

Если бесконечная карта не может быть раскрашена k красками, то некоторая конечная подкарта не может быть раскрашена k красками.

*Доказательство*:

Каждой стране на карте присваивается к первичных формул (по одной для каждого цвета).

Выписываем тривиальные аксиомы такие, что каждая страна делает истинным ровно один цвет, и соседние страны делают истинными разные цвета.

Получаем бесконечное количество высказываний. Доказательство проведем от противного. Пусть бесконечная карта не может быть раскрашена k красками, а все конечные подкарты могут быть раскрашены соответствующим образом. Тогда, используя теорему компактности, бесконечная карта также может быть раскрашена k красками. Получили противоречие.

### *4.3.4 Что и как можно выразить в логике I порядка.*

Формулы логики первого порядка - это конечные цепочки символов, состоящих из символов переменных x, y, z, …, логических связок &, ∨, ¬, →, =, кванторов всеобщности и существования ∀ и ∃, скобок (,).

Кроме этих логических символов можно ввести множество *L* примитивных нелогических символов. Например, если мы работаем с абелевыми группами, то множество *L* состоит из функционального символа + для группового сложения и константного символа 0 для нулевого элемента. При работе с множеством *L* имеет символ принадлежности ∈.

*Определение. Теория* - множество предложений.

*Определение*. Теория называется *конечно аксиоматизируемой*, если она обладает конечным множеством аксиом.

*Определение*. Множество Δ называется *множеством аксиом* теории Г, если Г и Δ имеют одни и те же следствия.

*Определение.* ϕ следствие Σ, если любая модель для Σ является моделью и для ϕ.

Для нас интересны следующие вопросы:

* Какие теории можно описать с помощью конечного числа аксиом?
* Какие теории нельзя описать с помощью конечного числа аксиом, но можно - с помощью бесконечного?
* Существуют ли теории, которые вообще нельзя описать с помощью логики предикатов первого порядка?

Таким образом эти вопросы приближают нас к пониманию границы применимости, выразимости логики первого порядка.

Так что же можно выразить в логике первого порядка? Ответы на поставленные вопросы мы можем получить, используя стандартный метод для доказательства неаксиоматизируемости теорий - теорему компактности.

**Теорема компактности** (Гедель - Мальцев)

*Пусть Т - произвольное множество аксиом логики первого порядка. Если для каждого конечного подмножества Т0 множества Т существует модель для всех аксиом из Т0, то существует модель для всех аксиом из Т.*

**Альтернативная форма теоремы компактности:**

*Если T∪{ψ} - множество предложений логики первого порядка, и T|=ψ (ψ - логическое следствие из T), то существует конечное T0⊆T: T0|=ψ.*

Это следует из основной формулировки теоремы компактности.

Данная формулировка яснее показывает смысл теоремы, фактически нам говорят:

***“Если вы из бесконечной теории выводите ψ, то вы на самом деле пользуетесь только конечным числом аксиом.”***

Теорема компактности не верна для логики высказываний II порядка или даже для слабой логики II порядка.

**Применение теоремы компактности. Группы**

Рассмотрим пять математических понятий (а)-(f) и попробуем ответить на поставленные вопросы об аксиоматизируемости теорий, используя теорему компактности.

1. группа,
2. абелева группа,
3. абелева группа, в которой порядок любого элемента ≤ N,
4. полная группа,
5. периодическая группа.

Мы можем утверждать, что

(а) - (с) - легко задаются несколькими аксиомами логики первого порядка,

(d) требует бесконечного списка аксиом.

(f) - не является понятием логики первого порядка.

Итак,

1. группа: **G** = (G, +, 0), где ‘+’:

∀x∀y∀z [x+(y+z) = (x+y)+z] (1)

∀x [x+0 = x] (2)

∀x∃y [x+y = 0] (3)

Или иначе можем сказать, что **G** является моделью для (1),(2),(3) и написать **G**|=(1),(2),(3) вместо того ,чтобы говорить, что **G** удовлетворяет (1),(2),(3).

1. абелева группа - группа, удовлетворяющая аксиоме:

∀x∃y [x+y = y+x] (4)

1. Введем соглашение: заменяем формальный терм (x+x) термом 2x, ((х+х)+х) => 3х и т.д. (Nx+x)=>(N+1)x.

абелева группа, в которой порядок любого элемента ≤ N: если **G** является моделью для выражения

∀x [x=0∨2x=0∨…∨Nx=0] (5)

где N - константа

1. Абелева группа **G** является полной, если

**∀**n≥1 ∀x∃y[ny=x] (6)

*Пример* - рациональные числа.

(6) не является аксиомой логики первого порядка, т.к. квантор ∀ действует на множестве положительных натуральных чисел, а не рассматриваемой области G. Попытаемся имитировать (6).

⇒ Заменим (6) на бесконечный список аксиом:

∀x∃y [x = y] (6.1)

∀x∃y [2y = x] (6.2)

. . .

∀x∃y [ny = x] (6.n)

*Утверждение 3.* Любое конечное множество, истинное во всех полных абелевых группах, истинно и в некоторой неполной абелевой группе. Иными словами, понятие полной абелевой группы не является конечно аксиоматизируемым в логике I порядка.

1. Абелева группа G является периодической группой, если

∀x∃n≥[nx=0] (7)

Вновь попытаемся имитировать (7); но получится

∀x [x=0∨2x=0∨…∨nx=0 …] (7′)

Здесь n - переменная величина, может быть неограниченна для всех x в целом.

Изучение таких бесконечных формул является невозможным. Проясняет ситуацию

*Утверждение 4.* Множество предложений логики I порядка истинных во всех периодических абелевых группах, истинно некоторой непериодической абелевой группе **H**.

(На самом деле мы покажем, что если **G** - абелева группа, множество порядков элементов которой не ограничено, то существует непериодическая группа **Н**: **G≡**H. Последнее означает, что каждое предложение логики первого порядка , истинное в **G**, также иcтинно и в **H** и наоборот. Поэтому класс периодических групп не может быть охарактеризован даже множеством аксиом I порядка конечным или бесконечным.

Доказательства утверждений проводится с помощью теоремы компактности [21].

## 4.4 Элементы среды саморазвития.

В соответствии с моделью ПО в компьютерном учебнике-задачнике среда саморазвития строится на понятии метатеории в рамках современного аксиоматического метода (логико-алгебраические спецификации). Так как в таком ракурсе этот вопрос не входит в задачу дипломной работы, то предлагается только перечень ключевых понятий, который составлен также на основании дипломов [31][32].

Ключевые понятия: *универсальная алгебра, логика, модель, язык исчисления предикатов, язык исчисления высказываний, многообразия, клаузальная логика (хорновская), инициальная модель, многосортная логика, метатеория.*

# 5 Реализация задания практикума для аппликативного стиля.

Наличие в системе задания практикума - нетривиальной классической задачи программирования - обусловлено

* во-первых, необходимостью обеспечить прагматическую основу, на которой могут быть предъявлены выбранные идеи и ориентиры;
* во-вторых, соответствием модели обучения и принципам РО (нацеленность на самостоятельную созидающую деятельность обучаемого);
* в-третьих, необходимостью обеспечить связность материала по выбранной теме.

Постановка задачи определена во **Введении** и **Приложении F**.

В данной дипломной работе приводится реализация задания практикума для аппликативного стиля.

## 5.1 Макеты задания.

Ïðèâåäåì âàðèàíòû ôîðìóëèðîâîê ðàçëè÷íîãî óðîâíÿ ñëîæíîñòè:

1. Íåîáõîäèìî ðåøèòü в рамках группы óðàâíåíèå âèäàÅ1•õ•Å2=À, ãäå Å1, Å2 è À ïðîèçâîëüíûå òåðìû (âûðàæåíèÿ), õ-íåèçâåñòíîå äëÿ çàäàâàåìûõ èíòåðïðåòàöèé.
2. Äàíà ãðóïïà öåëûõ ÷èñåë ñ îïåðàöèåé ñëîæåíèÿ. Èñïîëüçóÿ ßÏ Ëèñï

 ïîçâîëèòü ïîëüçîâàòåëþ

* çàïèñàòü óðàâíåíèå âèäà Å1•õ•Å2=À, ãäå Å1, Å2 è À ïðîèçâîëüíûå âûðàæåíèÿ, ñîñòîÿùèå èç ÷ëåíîâ ýòîé ãðóïïû è çíàêà îïåðàöèè,
* ïðåîáðàçîâàòü ýòî óðàâíåíèå ê óïðîùåííîìó âèäó, èñïîëüçóÿ ñâîéñòâà ãðóïïû,
* âû÷èñëèòü âûðàæåíèå â ïðîèçâîëüíîé òî÷êå.
1. Ðåøàÿ ïîñòàâëåííóþ çàäà÷ó - ðåàëèçîâàòü âçàèìîäåéñòâèå ñ "ãðóïïîé öåëûõ ÷èñåë ñ îïåðàöèåé ñëîæåíèÿ", ìû äîëæíû ïîíèìàòü, ÷òî ñìîæåì ðåøèòü òàêæå è áîëåå îáùóþ çàäà÷ó - ñ ãðóïïîé âîîáùå, ãäå ÷ëåí ãðóïïû - íå îáÿçàòåëüíî öåëîå ÷èñëî, îïåðàöèÿ - íå îáÿçàòåëüíî ñëîæåíèå, ïðîòèâîïîëîæíûé ýëåìåíò '-x', íåéòðàëüíûé ýëåìåíò - '0'. Â ðåøàåìîé â äàííûé ìîìåíò çàäà÷å èñïîëüçóåòñÿ êîíêðåòíàÿ èíòåðïðåòàöèÿ ãðóïïû, òî åñòü öåëûå ÷èñëà, ñëîæåíèå, '-x', '0'. Óòî÷íåíèå èíòåðïðåòàöèè ãðóïïû â óñëîâèè çàäà÷è ïîÿâèëîñü ëèøü çàòåì, ÷òîáû ìû ìîãëè ðåàëüíî ïî÷óâñòâîâàòü è äðóãèå ïðîáëåìû, óæå ñâÿçàííûå ñ ôóíêöèîíàëüíûì ñòèëåì è Ëèñïîì. Ýòó èíòåðïðåòàöèþ ìîæíî çàìåíèòü íà ëþáóþ äðóãóþ, èñïîëüçóÿ ñìåøàíûå âû÷èñëåíèÿ.

Èòàê,

1. Âû÷èñëèòü âûðàæåíèå â ëþáîé çàäàííîé òî÷êå, èñïîëüçóÿ âû÷èñëèòåëüíûå âîçìîæíîñòè ßÏ Ëèñï, ãäå âûðàæåíèå ïîíèìàåì êàê ÁÍÔ:

 <âûðàæåíèå>::=<âûðàæåíèå><çíàê îïåðàöèè><âûðàæåíèå>

 |(<âûðàæåíèå>)

 |<id>

 |<öåëîå ÷èñëî áåç çíàêà>

 <çíàê îïåðàöèè>::= + | - .

Îáúÿñíèì öåëü ðàçáèåíèÿ çàäà÷è íà ïîñëåäîâàòåëüíûå ìàêåòû. Îíà ñîñòîèò â òîì, ÷òîáû îáåñïå÷èòü îáó÷àåìîìó:

1. ÿñíîå âèäåíèå çàäà÷è,
2. âûäåëåíèå ýëåìåíòàðíûõ ñîñòàâëÿþùèõ è ïîíÿòèé,
3. è, íàêîíåö, âîçìîæíîñòü ðàçíåñòè âî âðåìåíè âûïîëíåíèå âñåãî çàäàíèÿ ïðàêòèêóìà (ðàáîòà íàä âûïîëíåíèåì îäíîãî ìàêåòà çàíèìàåò â ñðåäíåì âðåìÿ îäíîãî “åñòåñòâåííîãî” ñåàíñà ðàáîòû çà êîìïüþòåðîì).

Ìàêåòàìè äëÿ ðåàëèçàöèè çàäàíèÿ ïðàêòèêóìà ìîãóò áûòü:

1. Ââîä óðàâíåíèÿ.
2. Âåðèôèêàöèÿ óðàâíåíèÿ: ðàñïîçíàòü óðàâíåíèå; ïðîâåðèòü ïðàâèëüíîñòü âûðàæåíèé (íàïðèìåð, õ ìîæåò âñòðå÷àòüñÿ òîëüêî îäèí ðàç).
3. Ðåøåíèå óðàâíåíèÿ â ñèìâîëüíîì âèäå. Âûðàæåíèÿ Å1, Å2, À óïðîùàþòñÿ (èñ÷åçàþò íåéòðàëüíûå è âçàèìíîîáðàòíûå ýëåìåíòû), óðàâíåíèå ñâîäèòñÿ ê âèäó õ=Expr, ãäå Expr - íîðìàëüíàÿ ôîðìà ðåøåíèÿ: ~Å1•À•~Å2.
4. Íàéòè çíà÷åíèå íåèçâåñòíîãî õ â çàäàâàåìîé ïîëüçîâàòåëåì êîíêðåòíîé (íî, âîçìîæíî, íåïîëíîé) èíòåðïðåòàöèè I: xI=ValI (Expr).

Ïîäðîáíåå:

1. Äëÿ íîðìàëüíîé ðàáîòû ñ ââåäåííûì âûðàæåíèåì (à â íàøåì ñëó÷àå ýòî ïðîñòåéøåå àðèôìåòè÷åñêîå âûðàæåíèå - ñì. Óñëîâèå) íóæíî ïðåîáðàçîâàòü åãî â áîëåå óäîáíóþ ôîðìó. Òî åñòü ñòðîêó, ãäå âñå ñèìâîëû íå íåñóò ñìûñëîâîé íàãðóçêè è ðàâíîçíà÷íû, íóæíî ïðåîáðàçîâàòü â ïîñëåäîâàòåëüíîñòü ëåêñåì. Òàêèì îáðàçîì íóæíî âûïîëíèòü çàäà÷ó ëåêñè÷åñêîãî àíàëèçàòîðà, à òàêæå ïðîèçâåñòè ñòðóêòóðèçàöèþ âûðàæåíèÿ. "Ëåêñè÷åñêèé àíàëèç äàíûõ".
2. Òàê êàê íàøå âûðàæåíèå - ñòðóêòóðèðîâàííàÿ åäèíèöà (âàæåí ïîðÿäîê âûïîëíåíèÿ îïåðàöèé), òî âîçíèêàåò ïîòðåáíîñòü ïðåäñòàâèòü âûðàæåíèå â âèäå äåðåâà. Çàäà÷à "Âûðàæåíèå â âèäå äåðåâà".
3. Âû÷èñëèòü âûðàæåíèå â òî÷êå, â êà÷åñòâå ðåçóëüòàòà ïîëó÷èâ íîâîå âûðàæåíèå - çàäàåì íåïîëíóþ èíòåðïðåòàöèþ ìíîæåñòâà ïåðåìåííûõ â âûðàæåíèè (êàððèçàöèÿ). Ñì. çàäà÷ó "Âû÷èñëåíèå âûðàæåíèÿ â òî÷êå".
4. Óïðîùåíèå âûðàæåíèé çàòðàãèâàåò î÷åíü âàæíîå ïîíÿòèå äëÿ Ëèñïà ­ñèìâîëüíûå ïðåîáðàçîâàíèÿ, òàê êàê Ëèñï èñïîëüçóåòñÿ ïðåæäå âñåãî äëÿ ñèìâîëüíûõ âû÷èñëåíèé ïðè ðåøåíèè çàäà÷ èñêóññòâåííîãî èíòåëëåêòà. Çàäà÷à "Ñèìâîëüíûå ïðåîáðàçîâàíèÿ".

Îòìåòèì, ÷òî ìàòåìàòè÷åñêèé àñïåêò íàãðóæåí ìàòåðèàëîì, êîòîðûé âûäåëåí íà îñíîâàíèè ñïåöèôèêàöèé ïðîãðàìì, âîøåäøèõ â ìàêåòû.

Âûäåëåíèå óðîâíåé (ïîíÿòèé) ñïåöèôèêàöèè äëÿ ïðàêòè÷åñêîãî ìàòåðèàëà ïî çàäàíèþ ïðàêòèêóìà íåîáõîäèìî äëÿ íåðàçðûâíîãî ñóùåñòâîâàíèÿ â ñèñòåìå FLINT ïðàãìàòèêè è òåîðèè ïðåäìåòà. Îíè âûäåëÿþò òåîðåòè÷åñêèé ïëàí, êîòîðûé ïîçâîëèò îáó÷àåìîìó ðàçîáðàòüñÿ ñ ñóùåñòâîì ìåòîäîâ ðåøåíèÿ çàäà÷è.

Ââåäåíèå óðîâíÿ ñïåöèôèêàöèè äëÿ çàäàíèÿ ïðàêòèêóìà ïðåñëåäóåò öåëü “ðàçîðâàòü” ñåìàíòèêó, îñíîâàííóþ íà ìåõàíèçìå âûâîäà, è ñåìàíòè÷åñêîå ñëåäñòâèå.

Â ïóíêòå 3 âûäåëåíû ïîíÿòèÿ óðîâíÿ ñïåöèôèêàöèé, è òåïåðü âîïðîñ çàêëþ÷àåòñÿ â ðåàëüíîì åãî ïðåäúÿâëåíèè (äëÿ çàäàíèÿ ïðàêòèêóìà) àêñèîìàòè÷åñêîãî ìåòîäà. Ïðåäñòàâëÿåòñÿ ðàöèîíàëüíûì ïðåäúÿâèòü:

1. *äëÿ ïðîãðàììèñòñêîãî àñïåêòà - íåïîäâèæíàÿ òî÷êà êàê ïîäõîäÿùåå îáîáùåíèå ïðîáëåìíîé îðèåíòèðîâàííîñòè àïïëèêàòèâíîãî ñòèëÿ.;*
2. *äëÿ ìàòåìàòè÷åñêîãî àñïåêòà - ëÿìáäà-èñ÷èñëåíèå è êîìáèíàòîðíàÿ ëîãèêà êàê òåîðåòè÷åñêàÿ îñíîâà äëÿ ðåàëèçàöèé àïïëèêàòèâíûõ ÿçûêîâ;*
3. *äëÿ ñèñòåìîëîãè÷åñêîãî àñïåêòà - ìîäåëè Ä. Ñêîòòà..*

## 5.2 Понятия.

***Интерпретация***

При формулировке задания от обучаемого требуют задания конкретной интерпретации группы. Понятие интерпретации охватывает все области информатики.

В [28] сделана попытка создания среды саморазвития для идеи интерпретации (см Приложение D).

***Накапливающий параметр.***

Припомним исторический казус с Алонсо Черчем, который на подступах к теории алгоритмов λ-исчисления бился с λ-определимостью n-1. Пришлось дождаться Стефана Клини, который будучи аспирантом, решил задачу посредством накапливающего параметра.

F(i,n,r)=n=0→0

 i+1=n→r

 ⎜---⎜→F(i+1,n,r+1)

Обращение для n-1=F(0,n,0). Накапливающим параметром является i.

Покажем действенность нашего подхода совместного рассмотрения стилей, чтобы показать как, между делом, воспринимается техника накапливающего параметра посредством логического стиля. Рассмотрим порождающее предъявление n-1 в логическом стиле.



Запишем это представление посредством рекурсии, тогда в обращении к функции должно передаться базовое значение и аргумент n. Итак, обращение -1(1,n,0). Схема рекурсии с учетом -1(0,0), будет

-1(i,n,r)=

Сравнение реализации с реализацией Клини показывает, что получили эквивалентные схемы с точностью до обращения. Верификация на частном случае:

-1(1,3,0)→-1(2,3,1)→(3,3,2)→2

Рассмотрим пример с числами Фибоначчи. Порождающая схема будет выглядеть следующим образом:



Обращением к рекурсивной функции может быть F(1,1,n+r). Тогда схема рекурсии имеет вид

Fib(p,q,n,r)=

Так как параметр р не несет информационной нагрузки, то упростим схему, допустив обращение F(1,1,n,1). Схема рекурсии примет вид

Fib(i,p,n,q)=

Обратимся с Fib(1,1,5,1). Должно получиться 8.

Fib(1,1,5,1)→Fib(2,1,5,2)→Fib(3,2,5,3)→Fib(4,3,5,5)→Fib(5,5,5,8)→8

Итак, за счет согласованного рассмотрения логического и аппликативного стилей легко добывается знание, которое называют техникой накапливающего параметра или эффективной схемой МакКарти.

***Неподвижная точка***

Неподвижная точка является подходящим средством, чтобы показать возможности аппликативных языков. Рассмотрим абстракцию неподвижной точки

Y=λh.(λx.h(x x))(λx.h(x x))

Приспособим нашу функцию S к действию функционала Y.

Σ=λf.λn.

Развернем рекурсию

YΣ3→(λx.Σ(x x))(λx.Σ(x x))3

→Σ((λx.Σ(x x))(λx.Σ(x x)))3

Чтобы выделить существо дальнейших преобразований, изучим действие Y.

YΣ→Σ(YΣ)

Если предположить аппликативный (энергичный, вызов по значению) порядок вычислений, тогда зациклимся. Это произойдет из-за невозможности посчитать аргумент YΣ.

YΣ→Σ(ΥΣ)→Σ(Σ(ΥΣ))→Σ(Σ(Σ(ΥΣ)))→…

Если будут действовать отложенные вычисления (ленивые, вызов по имени), то есть будет нормальный порядок вычислений, тогда развернется рекурсия.

Перепишем наш пример.

YΣ3→Σ(ΥΣ)3→

+3→ +(ΥΣ(- 3 1))3→ +(Σ(ΥΣ)(- 3 1))3→

Обозначив через S=λn.

+(S(- 3 1))3→

+(+(ΥΣ(- 2 1))2)3→

+(+(1 2) 3)→

+ 3 3→6

Мы исходим из того, что примитивная функция + является строгой по отношению к своим аргументам. То есть она не будет выполняться пока не будут вычислены до конкретного числа оба аргумента.

***Смешанные вычисления***

Смешанные вычисления являются существом аппликативного программирования. Посредством обобщенной рекурсии (не примитивной) и бестиповости “произрастают” возможности интерпретации. Эта роль смешанных вычислений рассматривалась в дипломных работах [27], [28]. Здесь приводятся рассуждения на упрощенном примере.

Пусть задана примитивная функция +1=λx.(x+1).

Тогда можем посчитать +15→λx.(x+1)5→6.

Нарастим возможности за счет обобщения примитива до

+λx.y.(x+y)

и введения функции высшего порядка

**F**λf.λx.(f x)

Тогда, используя карринг, можем посчитать требуемое обращение F(+ 1) 5. Действительно

F(+1)5→λx.((+1)x)5→(+1)5→λy.(1+y)5→6

Введем двойную суперпозицию

****

К +1 вернулись для простоты записи. Попробуем посчитать +45 с помощью (+1) и F2. Можем действовать двумя способами. Обычный способ, соответствующий императивному стилю, представляется таким образом:

⇒

+1(+17)→+18→9

Второй способ соответствует технике смешанных вычислений. Конструкцию предыдущего обращения F2+1 (+1(+15)) можем получить в виде F2+1(F2+15) посредством простого обращения F2F2+15. Это обращение ближе находится к формулировке “Что делать”. Действительно F2F2→λx.(F2(F2x)) то есть имеем выражение, которое двойную суперпозицию функции х, именно F2x, применит посредством двойной суперпозиции F2 (F2x).

F2 F2+1⇒ F2 (F2+1)→λx.(F2+1)(F2x)

F2 F2+15⇒ (F2+1)(F2+15)

Получили обещанное ранее обращение императивного случая. В дипломе [28] было показано, что пониманию “Что делать” также начинает помогать техника алгебры комбинаторов.

***Денотационная семантика***

За счет функционалов аппликативный стиль достигает хорошего уровня для формулировок “Что делать”. Приведем пример на языке FP. Функция факториал - рекурсивная

*def* fac=eq0→1, /\*n=0→1

 \*Ο[id, facΟ(-Ο[id, 1])] /\*\*(n, fac(n-1))

где Ο- функционал суперпозиции. Фактически запись передает посредством суперпозиции известное рекурсивное определение. Функция факториал нерекурсивна. Использует генератор τ. Например,

 τ: 5→<1,2,3,4,5>

*def* facn=/\*Οτ

Означает а) сгенерировать последовательность чисел <1,...,n>; b) расставить (reduce) операцию \* между числами последовательности слева направо 1\*(2\*(3\*…\*(n-1 n)...)

Функция вычисления длины последовательности:

*def* len=null→0; /\*nil→0

 +Ο[1, lenΟ tl] /\*+(1, len(tl(l)))

Нерекурсивная функция:

*def* len\_n=/Ο+α1

Означает, а) заменить члены последовательности 1; в) суммировать их со значением, равным нулю в базовом случае. Например,

len\_n:[a,b,c]→

/Ο+:[1,1,1]→

+(1,+(1,+(1,0)))→3

Знаменитая функция member

*def* mem=nullΟI2→F;

 eqΟ[I1, hdΟI2]→T;

 memΟ[I1, tlΟI2]

Нерекурсивная функция поинтересней.

*def* mem\_n=/or Ο α eq Ο distl

Конечно, без знания обращения к ней трудно догадаться, что происходит. Итак

mem\_n:<1, <3,2,1>>→

/or Ο α eq:<<1,3>, <1,2>, <1,1>>→

/or Ο <eq:<1,3>, eq:<1,2>, eq:<1,1>>→

or(F, or(F,T))→T

***Программирование без переменных***

Вопервых, будем различать аппликативное и функциональное программирование. Аппликативный стиль выстраивается на построении текста - абстракции (функции) и аппликации (применение функции). Поэтому функционалы в нем моделируются. Функциональный стиль выстраивается на функциях и заданных функционалах. Т.к. это программирование без переменных, то вычисление осуществляется применением функционалов к функциям и значениям.

Воспользуемся теорией рекурсивной функции Клини, чтобы познакомить с функциональным стилем. Например, покажем, как можно обойтись без объектных переменных, что немыслимо для императивного стиля. Рассмотрим классический пример n! Зададим необходимые примитивы:

O=λx.O

J=λx.(x+1)

=λx1,...,λxn.xm

Зададим необходимые функционалы:

суперпозиции 

рекурсии



Определим х+у.



Таким образом, x+y через функционалы представляется следующим образом:



Посчитаем 3+3

3+0=

3+1=J=1(3)=4

3+2=J=J(4)=5

Определим x\*y



Таким образом, x×y через функционалы представляется



Посчитаем 3×2

3×0=О(о)=0

3×1=+(3,0)=3

3×2=+(3,3)=6

# 6 Заключение.

Ïîçâîëèì ñåáå ïåðåéòè ê ÿçûêó “ñâîáîäíîé öâåòèñòîé ïðîçû, îáåùàþùåé íè÷åì íå îãðàíè÷åííîå óäîâîëüñòâèå” [5]. Ýòî âûçâàíî íåîáõîäèìûì â âåê ñèñòåì ôèêñàöèåé öåëåïîëàãàíèÿ ïðîåêòà.

Âî-ïåðâûõ, õîòåëîñü áû îáúÿñíèòüñÿ ïî ïîâîäó âûáîðà çàäà÷è - áàçîâîå îáó÷åíèå èíôîðìàòèêå, ïðè÷åì ñ âûäåëåíèåì â íåé íàóêîåìêîãî ÿäðà. Äåëî â òîì, ÷òî íå ìû âûáèðàåì çàäà÷è, à îíè íàñ. Â 1969 ãîäó Ì. Ìèíñêèé [6] íà÷àë ñâîþ òüþðèíãîâñêóþ ëåêöèþ ñëîâàìè: “Ïðîáëåìà ñîâðåìåííîé èíôîðìàòèêè ñîñòîèò â íàâÿç÷èâîé îçàáî÷åííîñòè ôîðìîé â óùåðá ñîäåðæàíèþ”. Ìîæåì ñ óäîâëåòâîðåíèåì êîíñòàòèðîâàòü çíà÷èòåëüíîå ïðîäâèæåíèå, òàê êàê ïðîáëåìà óæå äîñòèãëà ïîëÿ çðåíèÿ îáðàçîâàíèÿ. Ïîñìîòðèì øèðå. Åñëè îïåðåòüñÿ íà òàêèå àâòîðèòåòû êàê À. Í. Óàéòõåä, Ê. Ïîïïåð è øêîëó ýâîëþöèîíèñòîâ (Í. Í. Ìîèñååâ, À. Í. Ñåâåðöîâ), òî ñëåäóåò ïðèçíàòü, ÷òî ñóùåñòâóþò ýïîõè ñìåíû ïîäñîçíàòåëüíîãî ñòèëÿ ìûøëåíèÿ (ñìåíû êàòåãîðèéíîãî àïïàðàòà). Áîëåå òîãî, óæå ïîäõîäèì ê òîìó, ÷òîáû ñ÷èòàòü íåîáõîäèìûì ôîðìèðîâàíèÿ óìåíèÿ ñàìèì íàñòðàèâàòüñÿ íà ðàçíûå êàòåãîðèéíûå ñòèëè (Ý. Â. Èëüåíêîâ, Â. Ñ. Áèáëåð). Ïðèçíàêîì ñìåíû ïîäñîçíàòåëüíîãî ñòèëÿ ÿâëÿåòñÿ óâåëè÷èâàþùàÿñÿ ðîëü ðàöèîíàëüíîãî. Ýòî åñòåñòâåííî, òàê êàê áàçà (êàê ÄÍÊ) äîëæíà ñîäåðæàòü ñâîè áóäóùèå ðåçóëüòàòû. Ñèíòåçèðóþùàÿ îñîáåííîñòü èíôîðìàöèîííîãî ìîäåëèðîâàíèÿ, çàìûêàåìàÿ ê òîìó æå íà êîíñòðóèðîâàíèè â ïðîãðàììèðîâàíèè, ôèêñèðóåò â íàñ êàòåãîðèéíûé àïïàðàò êëàññè÷åñêîãî íàó÷íîãî çíàíèÿ (È. Êàíò), íî ñ îñîáûì óïîðîì íà èíòåãðèðóþùóþ îñîáåííîñòü ñîçèäàíèÿ (òâîð÷åñòâà). Òåì ñàìûì ïðîèñõîäèò èçìåíåíèå íàñ êàê âèäà çà ñ÷åò ðàöèîíàëèçàöèè ñ íåîáõîäèìûì íàïîëíåíèåì òâîð÷åñêèì, à íå óçêîïðîôåññèîíàëüíûì. Êàê áû ìû íè ïðîòèâèëèñü ýòîìó, â êóëüòóðå äåéñòâóåò èíñòðóìåíò ïðèîáùåíèÿ (ñðàâíèì ñ ïèñüìåííîñòüþ) - ýòî êîìïüþòåð. Ïîýòîìó çàäà÷à âûáðàëà íàñ, íî ìû åå ðåøàåì â êîíêðåòíîé èíòåðïðåòàöèè: êîìïüþòåð êàê ÿçûêîâîå ñðåäñòâî ïðåäúÿâëåíèÿ çíàíèÿ äëÿ åãî èññëåäîâàíèÿ (âñïîìíèì: “ðåçóëüòàòîì âû÷èñëåíèÿ ÿâëÿåòñÿ ïîíèìàíèå à íå - ÷èñëî”).

Âî-âòîðûõ, îáúÿñíèìñÿ ïî ïîâîäó âûáîðà ìåòîäà ðåøåíèÿ - ïðèìåíåíèÿ Ðàçâèâàþùåãî Îáó÷åíèÿ. Â åãî îñíîâå ëåæàò äâå òåîðèè ðàçâèòèÿ: *ãåíåòè÷åñêàÿ ïñèõîëîãèÿ è ãåíåòè÷åñêàÿ ýïèñòåìîëîãèÿ* Æ.Ïèàæå - âûñòðàèâàåìàÿ îò èíòåíñèîíàëà, îò ×åëîâåêà; êóëüòóðíî-èñòîðè÷åñêàÿ òåîðèÿ Ë. Ñ. Âûãîòñêîãî - âûñòðàèâàåìàÿ îò ýêñòåíñèîíàëà, îò êóëüòóðû ñîöèàëüíîãî ×åëîâåêà. Èõ ïðåäòå÷àìè â ôèëîñîôèè ñëåäóåò âûáðàòü ñîîòâåòñòâåííî È. Êàíòà è Ã. Â. Ô. Ãåãåëÿ. Îïÿòü ñëåäóåò ïîâòîðèòü, íå ìû âûáèðàåì ìåòîä, à îí íàñ. Óñëîæíÿþùàÿñÿ ñîöèàëüíàÿ ñðåäà (èíôîñôåðà, íîîñôåðà) ñòàâèò êàê ýêîëîãè÷åñêóþ ïðîáëåìó ñóùåñòâîâàíèå ÷åëîâåêà â ñëîæíîé, ñàìîîðãàíèçóþùåéñÿ ñèñòåìå áûñòðî ìåíÿþùåãîñÿ îáùåñòâà. Ðåøåíèå ïðîáëåìû - âûéòè íà ìåòàóðîâíü îáó÷åíèÿ, êîãäà çàáîòÿòñÿ î ñîñòîÿòåëüíîñòè ýâîëþöèîíèðóþùåãî ñîñòîÿíèÿ èíòåëëåêòà. ÐÎ ïðåîäîëåâàåò ïðîòèâîðå÷èå, ñóùåñòâóþùåå ìåæäó íåóäåðæèìûì îíòîãåíåçîì èíòåëëåêòóàëüíîé ïðèðîäû homo sapiens (ñóáúåêò òâîð÷åñòâà â êóëüòóðå) è âûìèðàþùèìè ïðèìèòèâíûìè ñîöèàëüíûìè ñðåäàìè ñóùåñòâîâàíèÿ ÷åëîâåêà ñ èõ íîðìàìè íà ïðîôåññèîíàëüíóþ ïðèãîäíîñòü (îáúåêò âîçäåéñòâèÿ ñîöèàëüíûõ èíñòèòóòîâ, â òîì ÷èñëå îáðàçîâàíèÿ). ÐÎ ðàçðóáàåò ãîðäèåâ óçåë, îòêàçûâàÿ â ñîñòîÿòåëüíîñòè âñÿ÷åñêèì ïåäàãîãèêàì, íå âûñòðàèâàåìûì íà ïðèçíàíèè ãåíèÿ (ñàìî)ðàçâèâàþùåãîñÿ ÷åëîâåêà â êóëüòóðå. Ïîä÷åðêíåì ÿâíî íåóäèâèòåëüíîå ñîâïàäåíèå âî âðåìåíè äâóõ ñòîðîí óíèâåðñàëèçàöèè ×åëîâåêà: óñëîæíåíèÿ ðàöèîíàëüíîãî â âèäå è ìåòîäà â ïåäàãîãèêå, ðåøàþùåãî çàäà÷è îáó÷åíèÿ íà ïóòè ïðåäîñòàâëåíèÿ ñðåä äëÿ òðåíàæà òåîðåòè÷åñêîé ñàìîîðãàíèçàöèè èíòåëëåêòà.

Â-òðåòüèõ, íàñêîëüêî ìû ñóçèì ïðà-ëîãè÷åñêîå (ïî Ëåâè-Áðþëü) â íàøåì ìûøëåíèè, âî ìíîãîì, çàâèñèò îò ÿñíîãî ïîíèìàíèÿ äâóõ îïàñíîñòåé. Ïåðâàÿ îïàñíîñòü - â ïîçèòèâèçìå “êëàññè÷åñêàÿ ëîãèêà (òî åñòü ëîãèêà ó÷åáíèêîâ) è íàèâíûé ðåàëèçì çäðàâîãî ñìûñëà - äâà ñìåðòåëüíûõ âðàãà çäîðîâîé ïñèõîëîãèè ïîçíàíèÿ, âðàãà òåì áîëåå îïàñíîãî, ÷òî ÷àñòî óäàåòñÿ èçáåãíóòü îäíîãî òîëüêî äëÿ òîãî, ÷òîáû ïîïàñòü â îáúÿòèÿ äðóãîãî” [22]. Âòîðàÿ îïàñíîñòü - â íåäîîöåíêå ïðà-ëîãè÷åñêîãî ìûøëåíèÿ, î êîòîðîì ïðåäóïðåæäàåò Ëåâè-Áðþëü: “Åñëè ïîçíàâàòåëüíàÿ ôóíêöèÿ â ïðîöåññå ýâîëþöèè îáùåñòâà ñòðåìèòñÿ äèôôåðåíöèðîâàòüñÿ, îòäåëèòüñÿ îò äðóãèõ ýëåìåíòîâ, ñîäåðæàùèõñÿ â êîëëåêòèâíûõ ïðåäñòàâëåíèÿõ, òî îíà òàêèì ïóòåì, êîíå÷íî, ïðèîáðåòàåò ñâîåãî ðîäà íåçàâèñèìîñòü, îäíàêî íå äàâàÿ ýêâèâàëåíòà èñêëþ÷åííûõ åþ ýëåìåíòîâ. ×àñòü ýòèõ ýëåìåíòîâ ïðîäîëæàþò ñîõðàíÿòüñÿ ãäå-òî âíå åå è íàðÿäó ñ íåé”. Êàæåòñÿ ñïðàâåäëèâûì, ÷òî ýëåìåíòû ñîçèäàíèÿ (â èíôîðìàòèêå) - ýòî òî, ÷òî ìîæåò íåïðåðûâíî ïðîòèâîñòîÿòü íåñîñòîÿòåëüíîñòè ïðà-ëîãè÷åñêîãî è ïîñòåïåííî çàìåíÿòü åãî ýëåìåíòû íà ðàöèîíàëüíîå. Íî íàäî âñå âðåìÿ äåìîíñòðèðîâàòü (â îáó÷åíèè), ÷òî íåò íè÷åãî áîëåå ïðàêòè÷åñêîãî äëÿ ñîçèäàíèÿ â âåê ñèñòåì, ÷åì òåîðåòè÷åñêèé âçãëÿä íà âåùè.

Òàêîâû ïðåäïîñûëêè, â òîì ÷èñëå ñîöèàëüíûå, äèêòóþùèå öåëåïîëàãàíèå ïðîåêòà.

* Ôàêòè÷åñêè ìû ñòðåìèìñÿ ê òîìó, ÷òîáû äîáàâèòü ê âîëå ó÷åíèêà - óñâîèòü ïðåäìåò - òî öåëåïîëàãàíèå, êîòîðîå ïîçâîëèò åìó âûéòè çà ïðåäåëû óñâîåííîãî.
* Ó÷èòûâàÿ çàïðîñû âðåìåíè, ïðåîáðàçóþòñÿ ê âîçìîæíîìó êîíñòðóêòèâíîìó óðîâíþ äëÿ îáó÷åíèÿ òåîðèÿ ïîçíàíèÿ È. Êàíòà, êîíöåïöèÿ èíòåëëåêòóàëüíîãî ðàçâèòèÿ Æ.Ïèàæå è Ë. Âûãîòñêîãî, ýâîëþöèîííàÿ òåîðèÿ ðàçâèòèÿ âèäà Ã. Ãåêêåëÿ è À. Ñåâåðöîâà.
* Íàäååìñÿ, ÷òî ðàáîòà ñïîñîáñòâóåò íåóäåðæèìîé ïîïûòêå “...÷åëîâåêà ïðèîáðåñòè íå òîëüêî ñîáñòâåííîå ÿ, íî è ñïîñîáíîñòü îöåíèâàòü åãî ñî ñòîðîíû” [7]. Ñåãîäíÿ ýòî óæå çâó÷èò èìïåðàòèâíî: “ß äîëæåí ìûñëèòü ìîå ìûøëåíèå, à íå ìîå ìûøëåíèå ìåíÿ” [8].

# 7 Результаты работы.

Итоги выполнения дипломной работы в составе проекта следующие:

1. Предъявлена структура полной модели предметной области для компьютерного учебника-задачника (рис. 2, п. 1).
2. Рассмотрены ключевые понятия аппликативного стиля по теме ИСПОЛНЕНИЕ для класса обучаемых прагматик, выявлены уровни спецификации.
3. предъявлен материал, выявляющий ключевые понятия по идее ВЫПОЛНИМОСТЬ, который будет использован для системологического аспекта универсала; предъявлена полная структура учебного материала для класса универсал.
4. реализовано задание практикума (общий вариант) для аппликативного стиля, рассмотрены теоретические предпосылки введения уровня спецификаций.

Дальнейшим развитием дипломной работы является: во-первых, перенесение семантических понятий спецификации на задание практикума; во-вторых, реальное погружение учебного материала в реализуемую систему FLINT.

# 8 Литература.

1. Ãðîìûêî Â.È., Êó÷åâñêèé Þ.Â., Ïàí÷óê Î.À. Ðàçâèâàþùåå îáó÷åíèå â êîìïëåêñå îáó÷åíèÿ îñíîâàì èíôîðìàòèêè. Ïåäàãîãè÷åñêàÿ èíôîðìàòèêà, 95, ¹2.
2. Êíóò Ä.Å. “Ïðîãðàììèðîâàíèå êàê èñêóññòâî”. Ëåêöèè ëàóðåàòîâ ïðåìèè Òüþðèíãà çà ïåðâûå 20 ëåò. Ìîñêâà, “Ìèð”, 1993
3. Ãðîìûêî Â.È. Áàçîâîå îáó÷åíèå èíôîðìàòèêå. Âåñòíèê ÌÃÓ. Ñåðèÿ 15. Âû÷èñëèòåëüíàÿ ìàòåìàòèêà è êèáåðíåòèêà, 95, ¹2.
4. Ãðîìûêî Â.È., Êîëÿäêî Ì.Â., Òðèôîíîâ Í.Ï. Êîìïüþòåðíàÿ ñèñòåìà â êîìïëåêñå îáó÷åíèÿ îñíîâàì èíôîðìàòèêè. Ïåäàãîãè÷åñêàÿ èíôîðìàòèêà, 95, ¹2.
5. ×. Å. Ð. Õîàð “Ñòàðûå ïëàòüÿ èìïåðàòîðà”. Ëåêöèè ëàóðåàòîâ ïðåìèè Òüþðèíãà
6. Ìèíñêèé “Ôîðìà è ñîäåðæàíèå â èíôîðìàòèêå”.
7. Ìîèñååâ Í. Í. “Àëãîðèòìû ðàçâèòèÿ”
8. Çèí÷åíêî Â. Ï. “Àôôåêò è èíòåëëåêò â îáðàçîâàíèè”
9. Ìàòåìàòè÷åñêàÿ ëîãèêà â ïðîãðàììèðîâàíèè. Ñáîðíèê ñòàòåé ïîä ðåä. Ì.Â. Çàõàðüÿùåâà è Þ.È. ßíîâà, Ìîñêâà, “Ìèð” 1991.
10. Ðîäæåðñ Õ.“Òåîðèÿ ðåêóðñèâíûõ ôóíêöèé è ýôôåêòèâíàÿ âû÷èñëèìîñòü”, Ìîñêâà, “Ìèð” 1972
11. Áóëëîñ Äæ., Äæåôôðè Ð.“Âû÷èñëèìîñòü è ëîãèêà”, Ìîñêâà, “Ìèð” 1994
12. Ãðèñ Ä.“Íàóêà ïðîãðàììèðîâàíèÿ”,Ìîñêâà, “Ìèð” 1984
13. Äàë Ó., Äåéêñòðà Ý., Õîîð Ê. “Ñòðóêòóðíîå ïðîãðàììèðîâàíèå”
Ìîñêâà, “Ìèð” 1975
14. “Ëîãè÷åñêîå ïðîãðàììèðîâàíèå”. Ñáîðíèê ñòàòåé ïîä ðåä. Â.Í.Àãàôîíîâà, Ìîñêâà, “Ìèð” 1988
15. “Äàííûå â ÿçûêàõ ïðîãðàììèðîâàíèÿ”. Ñáîðíèê ñòàòåé ïîä ðåä. Â.Í.Àãàôîíîâà, Ìîñêâà, “Ìèð” 1982
16. “Ñåìàíòèêà ÿçûêîâ ïðîãðàììèðîâàíèÿ”. Ñáîðíèê ñòàòåé ïîä ðåä Â.Ì.Êóðî÷êèíà, Ìîñêâà,”Ìèð” 1980
17. Ýíãåëåð “Ìåòàìàòåìàòèêà ýëåìåíòàðíîé ìàòåìàòèêè”, Ìîñêâà “Ìèð”, 1987
18. Ãýðè,Äæîíñîí “Âû÷èñëèòåëüíûå ìàøèíû è òðóäíîðåøàåìûå çàäà÷è”
19. Àõî À., Õîïêðîôò Äæ., Óëüìàí Äæ.“Ïîñòðîåíèå è àíàëèç âû÷èñëèòåëüíûõ àëãîðèòìîâ”,Ìîñêâà, “Ìèð” 1979
20. Êåéñëåð Ã., ×ýí  ×. ×. “Òåîðèÿ ìîäåëåé”, Ìîñêâà, “Ìèð” 1977
21. Áàðâàéñ “Ñïðàâî÷íàÿ êíèãà ïî ìàòåìàòè÷åñêîé ëîãèêå”. ×àñòü 1, Òåîðèÿ ìîäåëåé. Ìîñêâà, “Íàóêà”, 1987ã.
22. Ë. À. Ñêîðíÿêîâ “Ýëåìåíòû òåîðèè ñòðóêòóð“
23. Õ. Áàðåíäðåãò “Ëÿìáäà - èñ÷èñëåíèå: åãî ñèíòàêñèñ è ñåìàíòèêà“
24. À. Ôèëä, Ï. Õàððèñîí “Ôóíêöèîíàëüíîå ïðîãðàììèðîâàíèå“
25. Â. À. Óñïåíñêèé “Òåîðåìà Ãåäåëÿ î íåïîëíîòå” (Ñåðèÿ “Ïîïóëÿðíûå ëåêöèè ïî ìàòåìàòèêå”, âûïóñê 57), Ìîñêâà “Íàóêà”, 1982.
26. Â. À. Óñïåíñêèé “Ìàøèíà Ïîñòà” (Ñåðèÿ “Ïîïóëÿðíûå ëåêöèè ïî ìàòåìàòèêå”, âûïóñê 57), Ìîñêâà “Íàóêà”, 1988.
27. Êîëÿäêî Ì. Â. Ðàçâèâàþùåå îáó÷åíèå èíôîðìàòèêå â ñèñòåìå FLINT. Äèïëîìíàÿ ðàáîòà. Êàôåäðà Àëãîðèòìè÷åñêèõ ßçûêîâ, ÂÌÊ ÌÃÓ, Ìîñêâà - 93, íàó÷íûé ðóêîâîäèòåëü Ãðîìûêî Â. È.
28. Ïàí÷óê Î. À. Ðàçâèâàþùåå îáó÷åíèå â ñèñòåìå FLINT. Àïïëèêàòèâíûé ñòèëü. Äèïëîìíàÿ ðàáîòà. Êàôåäðà Àëãîðèòìè÷åñêèõ ßçûêîâ, ÂÌÊ ÌÃÓ, Ìîñêâà - 94, íàó÷íûé ðóêîâîäèòåëü Ãðîìûêî Â. È.
29. Ïëåøêîâ Î. À Ðàçâèâàþùåå îáó÷åíèå â ñèñòåìå FLINT. Ëîãè÷åñêîå ïðîãðàììèðîâàíèå. Äèïëîìíàÿ ðàáîòà. Êàôåäðà Àëãîðèòìè÷åñêèõ ßçûêîâ, ÂÌÊ ÌÃÓ, Ìîñêâà - 94, íàó÷íûé ðóêîâîäèòåëü Ãðîìûêî Â. È.
30. Áû÷êîâà Í. Å. Ðàçâèâàþùåå îáó÷åíèå â ñèñòåìå FLINT. Ìåòîä è ïðàêòèêà ïîäãîòîâêè ó÷åáíîãî ìàòåðèàëà. Äèïëîìíàÿ ðàáîòà. Êàôåäðà Àëãîðèòìè÷åñêèõ ßçûêîâ, ÂÌÊ ÌÃÓ, Ìîñêâà - 95, íàó÷íûé ðóêîâîäèòåëü Ãðîìûêî Â. È.
31. Ïðèùåïîâ À. À. Ðàçâèâàþùåå îáó÷åíèå â ñèñòåìå FLINT. Òèïèçàöèÿ â ïðîãðàììèðîâàíèè. Äèïëîìíàÿ ðàáîòà. Êàôåäðà Àëãîðèòìè÷åñêèõ ßçûêîâ, ÂÌÊ ÌÃÓ, Ìîñêâà - 96, íàó÷íûé ðóêîâîäèòåëü Ãðîìûêî Â. È.
32. Íîâèêîâà Ò. Â. Ðàçâèâàþùåå îáó÷åíèå â ñèñòåìå FLINT. Ëîãè÷åñêèé ñòèëü - ñðåäà ðàçâèòèÿ. Ñðåäà ñàìîðàçâèòèÿ íà îñíîâå òåîðåìû Êóêà. Äèïëîìíàÿ ðàáîòà. Êàôåäðà Àëãîðèòìè÷åñêèõ ßçûêîâ, ÂÌÊ ÌÃÓ, Ìîñêâà-96, íàó÷íûé ðóêîâîäèòåëü Ãðîìûêî Â. È.

**Áëàãîäàðíîñòè.**

Â çàêëþ÷åíèè õî÷ó âûðàçèòü ãëóáîêóþ ïðèçíàòåëüíîñòü ñâîåìó íàó÷íîìó ðóêîâîäèòåëþ Âëàäèìèðó Èâàíîâè÷ó Ãðîìûêî çà âåðó â ìîè ñèëû è ïîñòîÿííóþ ïîääåðæêó â ðàáîòå. Òàêæå õî÷ó ïîáëàãîäàðèòü îäíîêóðñíèêîâ Íîâèêîâó Òàòüÿíó è Ïðèùåïîâà Àëåêñàíäðà çà ïîìîùü â ðàáîòå.

Êðîìå òîãî ïîìîãàëè â ðàáîòå è ðåáÿòà ñ òðåòüåãî êóðñà Çàäûìîâ Êîíñòàíòèí, Òàãåð Äìèòðèé è Íàçìèòäèíîâà Àëåíà..

# 9 Приложения.

## А. Ранние требования к предметной области системы (до введения уровня спецификаций).

## B. Несколько слов об энергичных и ленивых интерпретаторах.

## C. SECD-машина.

## D. Среда саморазвития для идеи интерпретации.

## E. Бестиповое лямбда-исчисление. Определения.

## F. Задание практикума.

1. Ïðàâèëî η-ïðåîáðàçîâàíèÿ îñíîâàíî íà òîì, ÷òî âûðàæåíèÿ **(λx. Ex)** è **E** îáîçíà÷àþò îäíó è òó æå ôóíêöèþ ïðè óñëîâèè, ÷òî x íå ÿâëÿåòñÿ ñâîáîäíîé ïåðåìåííîé â Å, òîãäà **(λx. Ex) À = E À** äëÿ ëþáîãî âûðàæåíèÿ À. [↑](#footnote-ref-1)
2. \* Íî ýòî ñóùåñòâî ïðåäëîæåííîãî ìåòîäà. Âñå â ïðåäëîæåííîì èñ÷èñëåíèè êîíâåðñèé. Ïåðâûé øàã íà÷èíàåòñÿ ñ t1||→t11.Если начинаем с S или K, то “стоим на месте”. Фактически доказательство левой части заменили счетом в правой части. [↑](#footnote-ref-2)