ЛОГИЧЕСКИЙ ПОДХОД К ПРОГРАММИРОВАНИЮ

Непейвода Н.Н.
Ижевск, СССР

§I. Общая характеристика подхода

Одним из фундаментальных вопросов программирования является: "Откуда берутся программы?". Первоначально программы рассматривались как уже данный нам алгоритм и изучались с точки зрения их функционирования и функционально эквивалентных преобразований. Позднее они рассматривались с точки зрения синтаксиса и методов трансляции с одного алгоритмического языка на другой. Сейчас изучается проблема их корректности. В итоге на первый план закономерно выходит сам процесс конструирования правильной программы.

Резко возросший объем математического обеспечения и расширившаяся сфера применения ЭВМ предъявили большие требования как к надежности получаемых программ, так и к производительности труда программистов и их коллктивов. В программировании накоплен громадный опытный материал, который порою противоречит сложившимся теоретическим традициям и настоятельно требует осмысления. В частности, интересные примеры таких противоречий приведены в работах Г.С.Цейтина (см.,напр.,[I]).

С другой стороны, современная математика вновь начинает превращаться в экспериментальную науку, но экспериментальную науку "богов". Создаваемые нами программы - новые и часто безумные миры. Мы - их творцы и исследователи. Интересно здесь, что наиболее абстрактные области математики, такие, как математическая логика (теория доказательств и теория моделей), общая алгебра(теория категорий, универсальная алгебра), теория множеств и т.п., оказываются широко применимыми и содержат гро-дное богатство практически важных структур.

Автор убежден, что стена между теоретическими и практическими знаниями, зачаотую трудно преодолимая даже внутри человека, в совершенстве владеющего и теми, и другими методами,

– главное препятствие на пути к решению многих проблем. О существовании этой стены говорили, в частности, Г.С.Цейтин и Д.Кнут, последний ярко показал трудности, возникшие в свое время у него при сломе этого внутреннего барьера.

Прочность этой стены обусловлена многими причинами. Главной из них является искусственно сторванная от жизни традиция изложения математических понятий. Она мешает понять многие интуитивно ясные и хорошо приложимые идеи, представленные в искусственно усложненной форме. Не менее неприятно пристрастие многих прикладников к работе "одним топором", только традиционными методами, без анализа особенностей исходной задачи. Мешают также спортивный дух теоретической математики, где больше всего ценится решение давно поставленной проблемы; деляческий дух прикладников, когда стремятся (часто подталкиваемые заказчиком) выдать какое угодно решение, лишь бы побыстрее; страсть теоретиков пережевывать, усиливая и обобщая до ненужности, одни и те же освященные традицией и принесшие когда-то пользу задачи; некритическое восприятие и бесконечное повторение практиками решений, однажды случайно принятых, и т.д.

Формированию своей позиции автор в значительной степени обязан работам [I-5] , которые помогли ему сломать стену у себя в голове и понять, насколько интересные теоретические задачи выдвигает нынешняя практика и насколько серьезно необходимо подходить к теории, если она предназначена для приложений. Автор рассматривает свою работу как в некотором смысле противостоящую статье Г.С.Цейтина, публикуемой в этом же томе. Он испытал в практических задачах противоположный переход – от процедурализма к логицизму. Интересно отметить, что в неоднократных личных дискуссиях с Г.С.Цейтиным обе стороны опирались на одни и те же факты, но делали из них различные выводы.

Логический подход к программированию [8-II] характеризуется следующими чертами:

– Во всех случаях принципиально признается первичность рассуждения и вторичность алгоритмической записи.

– Используемая логика, как содержательная, так и формальная, варьируется в зависимости от характера задачи (и, возможно, даже на разных этапах работы над одной и той же задачей).

– Понятия алгоритма и логического вывода должны слиться в единый объект. Поэтому и то, и другое понятие пересматриваются, и большое внимание уделяется их естественности и согласованности.

– Формальные теории рассматриваются как "метапрограммы" для узко проблемно ориентированного класса задач.

– Процесс построения программы состоит из нахождения конструктивного доказательства поставленной цели и очищения его от неалгоритмических частей.

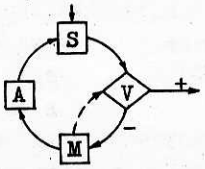
– Основное внимание уделяется слабым концепциям вычислимости.

Со стороны логики наш подход – это непосредственное развитие структурной теории доказательств, со стороны программирования – концепции структурированного программирования.

§2. Соотношение с другими подходами

Сравним развиваемый здесь логический подход с подходами Манны [I2–I4] , трансформационным подходом А.П.Ершова [2, I7–I9] , индуктивным подходом Я.М.Барздиня [20] . Отметим, что здесь мы сослались на те работы, где наиболее полно раскрыты особенности соответствующего подхода.

Анализ работ [I7–I9] позволяет выделить следующий, неоднократно проходимый при синтезе программы, цикл Манны:



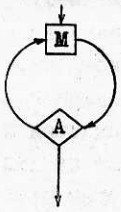
S – синтез,
A – анализ,
M – модификация,
V – верификация.

Вначале мы синтезируем каким угодно методом кусок программы. Затем проверяем, не получили ли мы случайно полного решения. Если нет, то модифицируем созданный кусок (возможно, каждый раз возвращаясь на верификацию), стремясь удовлетворить возможно большему куску главной цели. После этого анализируем построенную частичную программу, выявляем новую цель и возвращаемся на новый частичный синтез. Этот процесс построения мало зависит от качества алгоритмического языка и логического формализма. Он соответствует процессу построения прог-

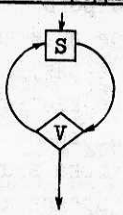
раммы снизу вверх. В случае заведомо плохо поставленной задачи, уточняемой в процессе решения, этот подход - наилучший. Главным его недостатком в нынешней форме является беспомощность при порождении условных предложений, но этот недостаток может быть устранен, если будет отброшена оправданная лишь традициями привязка логического формализма к классической логике. Но второй его недостаток, хотя он до сих пор и не очень мешал развитию, фундаментален и неустраним. Этот подход не может дать нам возможность забежать вперед практики программирования и дать рекомендации по ее перестройке, поскольку он в принципе базируется на фонде приемов программирования.

При трансформационном, индуктивном и логическом подходах также возникают соответствующие циклы построения программы.

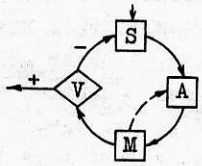
Цикл Ершова:



Цикл Барздиня:



Цикл логического подхода:



Трансформационный подход действует, если уже сама постановка задачи дает нам "теоретический" алгоритм ее решения, заданный на языке, возможно, очень высокого уровня. Например, им может быть теоретико-множественное определение понятия сортировки [I7]. Затем этот практически негодный алгоритм модифицируется с целью получить практически пригодный. Каждый новый этап модификации оценивается с точки зрения эффективности. После достижения нужного качества и достаточной детализированности синтез кончается.

Этот подход требует наличия непрерывного спектра хорошо согласованных между собой языков от высокого до низкого уровня (см. Бауэр [29]). Он базируется на предположении, что

алгоритмы появляются лишь из алгоритмов. Если в некоторой ситуации он применим, то работает великолепно. Он обеспечивает переход от теоретической вычислимости к практическим вычислениям.

При индуктивном подходе мы опираемся на некоторое множество примеров вычислений и стремимся угадать логическую структуру программы, их описывающей. Созданная программа испытывается на новых примерах. Если испытания выявляют неправильность, мы должны модифицировать (или даже синтезировать заново) программу. Здесь верификация понимается как практическое тестирование. Впрочем, и в подходе Манны ее естественно понимать подобным образом. Этим подходом обеспечивается обратный переход: от вычислений к вычислимости.

Логический подход применим, если имеется точно поставленная цель и хорошее описание вычислительной обстановки. Однако здесь не требуется, чтобы принципиальная вычислимость решения была уже установлена строго (нас всегда интересует вычислимость в некотором классе вычислимых функционалов, возможно, гораздо меньшем, чем все рекурсивные функционалы). Синтез программы здесь сразу создает принципиально правильный текст. Однако полученная программа может не ложиться в прокрустово ложе данного диапазона значений, объема памяти, времени работы. С этой точки зрения мы и анализируем программу, выявляя дополнительные спецификации. Затем мы пытаемся модифицировать полученное решение так, чтобы удовлетворить новым ограничениям. Если достичь этого не удается, мы должны повторить синтез с учетом новых спецификаций.

Из этих рассмотрений видно, что трансформационный подход относится к логическому так же, как подход Барздиня к прагматическому подходу Манны.

§3. Основные задачи и модели

Известно (см., напр., [6]), что в общей задаче целенаправленного поведения можно выделить следующие крайние подслучаи.

I. Чистое программирование. Перед нами стоит задача построения выходных данных, удовлетворяющих заданным спецификациям, по входным данным из некоторого множества. Входные

и промежуточные данные задаются полностью и могут храниться неограниченное время без искажения. Ограничения на объем хранимой информации несущественны.

Как неоднократно было показано ранее (см., напр., [3], естественная логика данной задачи - интуиционистская. Более того, аппарат, подготовленный к использованию на практике, был создан еще в 30-40-е годы (интуиционистское исчисление естественного вывода и понятие реализуемости по Клини [7]). Однако как практики, так и теоретики проходили мимо него. В частности, голословные утверждения о невозможности использовать интуиционистскую логику как практический инструмент были сделаны в работах [I5,I6].

Практическая задача, на которую ориентировано рассмотрение,- в данном случае задача сборки программ из отлаженных программ некоторого пакета. В настоящее время задача чистого программирования исследована лучше всего, и ей мы уделим основное внимание.

2. Ограниченное программирование. Задача перед нами стоит та же, что и в прошлом пункте, но ограничения на ресурсы являются существенными. Для этой задачи естественная конструктивная логика в готовом виде не существовала. Первыми шагами на пути к ней являются работы [2I,22] . Появляющаяся конструктивная логика также базируется на интуиционистской, но существенно ее видоизменяет.

3. Простейшее планирование. Имеется пассивный мир и активный робот, каждое действие которого изменяет мир. Задача состоит в построении плана действий для перехода от состояний, описываемых данным предусловием, к состояниям, описываемым данным постусловием. Количество хранимых промежуточных данных может быть ограничено, а действия с ними сводятся лишь к проверке условий, чтению из внешней среды и записи туда. В рамках таких задач естественно описывается программирование в машинных кодах и на алгоритмических языках традиционного типа - с присваиваниями. Конструктивные логики простого планирования принадлежат классу релевантных логик, но прямых предков у них там не было. Здесь мы покажем элементарный фрагмент одной из таких логик, которую естественнее всего

считать логическим оформлением понятия продукционной грамматики и методов, использованных в GPS (см., напр., [6]).

Рассмотрим простейшую модель задачи планирования. Она состоит из универса (множества возможных состояний мира) U и набора подмножеств U , сопоставленных каждой пропозициональной букве (предикату). Каждому элементарному действию сопоставляется функция из U в PU , т.е. недетерминированная функция на U . На основе этой модели можно построить пропозициональную логику, процесс нахождения доказательства в которой соответствует построению разветвленной композиции элементарных действий для достижения заданной цели (см.[23]).

Центральным в этой логике является понятие программистской триады: действие, посылка, обещание. Для описания действий вводятся два набора связок: конструктивные, требующие действий, и дескриптивные, аналогичные связкам булевой логики.

Приведем пример использования программистских триад для описания системы предписаний для открывания и закрывания двери. Программистскую триаду будем обозначать $f \textcircled{r} A = B$, где f - элементарное действие; A - его посылка (условие применимости); B - его обещание. Эта запись читается "f применимо при условии A и обещает дать B" или "f реализует переход от A к B". Дескриптивные связки будем обозначать and, or, not, $\Rightarrow$, конструктивные - &, $\vee$, $\neg$, $\Rightarrow$.

Пример I. Пусть наш универс - множество состояний ключа и замка. Пусть у нас заданы предикаты "ключ в кармане", "ключ в замке", "замок открыт", "замок закрыт" и элементарные действия "достать ключ", "вставить ключ в замок", "вынуть ключ", "положить ключ в карман", "повернуть ключ (сокращенные обозначения КК,КЗ,ЗО,ЗЗ, ДК,ВКЗ,ВК,ПКК,ПК соответственно).

Опишем эту модель следующей системой аксиом, большинство из которых будут триадами:

A1. ДК $\textcircled{r}$ КК $\Rightarrow$ КК and not КЗ.

A2. ВКЗ $\textcircled{r}$ not КК and not КЗ $\Rightarrow$ КЗ.

A3. ВК $\textcircled{r}$ КЗ $\Rightarrow$ not КК and not КЗ.

A4. ПКК $\textcircled{r}$ not КЗ and not КК $\Rightarrow$ КК.

A5. ПК $\textcircled{r}$ (ЗЗ and КЗ $\Rightarrow$ ЗО and КЗ) and (ЗО and КЗ $\Rightarrow$ ЗЗ and КЗ).

А6. КК $\Rightarrow$ not КЗ.
А7. ЗЗ $\Rightarrow$ not ЗО.

В этом описании мы предполагаем, что предикаты разделены на два семантических класса: {КК,КЗ} и {ЗЗ,ЗО}. В дескриптивные аксиомы А6,А7 входили предикаты из одного и того же класса. Это - общее условие на дескриптивные аксиомы. Каждая из них должна связывать между собой предикаты одного семантического класса. По умолчанию,предполагается, что предикаты семантических классов, не входящих в конструктивную аксиому, ею не изменяются.

Пример 2. Доказательство и программа для примера I.

Рассмотрим цель КК and ЗЗ $\Rightarrow$ КК and ЗО (т.е., имея ключ в кармане и закрытую дверь, отпереть ее и положить ключ обратно). Доказательство осуществимости этого перехода мы запишем в обозначениях, принятых в функциональной системе правил вывода [9].] означает "допустим", вертикальная черта отмечает вспомогательный вывод, опирающийся на сделанное допущение, утверждения, записанные на одной и той же строке, считаются верными одновременно.

КК; ЗЗ
КК, КЗ;
КК, КЗ;
КЗ; ЗО
КК, КЗ;
КК;
КК ЗЗ КК ЗО

Программа: begin ДК; ВКЗ; ПК; ВК; ПКК; end .

Отметим, что при построении доказательства действия, реализующие конструктивные аксиомы, нам не нужны. Однако некоторые характеристики реализующих действий могут существенно помочь при поиске вывода. Их целесообразно добавлять в аксиомы в виде конструктивных модальностей, фиксирующих расход ресурсов на исполнение действий (см., напр.,[2,22]).

4. Наблюдения. Имеется активный внешний мир и датчики информации. Объем и точность получаемой информации существенно меньше необходимых для создания полной модели мира. Вспо-

могательная информация может терять точность в процессе хранения или вообще храниться лишь ограниченное время. Задача состоит в распознавании некоторого состояния мира. Логика таких задач – конструктивная временная логика, разработка которой только начинается.

5. Общие задачи, целенаправленного поведения естественно разбиваются на подзадачи этих трех видов. Например, управление манипулятором – на планирование и вычисление, контроль технологического процесса – на вычисления и наблюдения, и т.д.

6. Антилогическое направление. После увлечения логическими методами у части исследователей в области искусственного интеллекта наблюдается сейчас реакция. Наиболее остро и аргументированно доводы против логического подхода высказывались Г.С.Цейтиным [30]. Однако в еще более резкой форме те же недостатки логики, которые критикуются как препятствия для ее применения, критиковались Г.Крайзелем [4] как препятствия для ее развития. Статья Г.Крайзеля отличалась тем, что, помимо критики, показывались накопленные и пропадающие втуне богатства и указывались пути преодоления возникших недостатков.

Суммируя возражения против логики, можно отметить, что все они касаются ее традиционных областей в традиционной их форме. Критики отождествляют логику с классической логикой, причем в одной из традиционных форм; формализацию с ее традиционной методикой, словом, науку с ее традициями (в данном случае – предрассудками).

В качестве замены логических методов предлагается аппарат атрибутов и фреймов. Однако атрибут – это лишь одна из трех ипостасей, в которых выступает предикат в программе, а фрейм – одна из форм конструктивной импликации. Автор считает, что вообще никакого алгоритмического мышления не существует, но зато формы логического гораздо разнообразнее, чем принято считать.

Уход в сторону фреймов и атрибутов обладает тем же недостатком, что и подход З.Манны. Обобщать накопленный практический опыт необходимо, но коллекционирование приемов не есть

обобщение. Настоящая теория должна преобразовывать практику и стремиться идти вперед нее. За набором практических наблюдений необходимо искать малое число весьма абстрактных понятий и принципов, всех их порождающих. К таким обобщениям не видно другого пути, как через логику.

Однако при этом необходимо помнить, что в традиционной форме почти ничего из современной логики применено быть не может. Если содержание в максимальной степени нужно стремиться брать из теории, то форму - из хорошей практики.

Отметим одну из наиболее трудных проблем новой конструктивной логики. Это - проблема взаимодействия различных теорий, зачастую основанных на разных логиках. Например, в случае планирования действий мы даже в случае неизменной конструктивной теории имеем все время изменяющуюся дескриптивную.

§4. Предпосылки логического языка программирования

До конца статьи мы будем заниматься лишь задачей чистого программирования.

Развитие логического подхода к данной задаче прошло три этапа. Вначале среди множества формализаций логического вывода и программы были выбраны два, теснее всего связанные друг с другом. Это естественный вывод в форме Фитча и язык Алгол-68. Критический анализ этих понятий показал, что уровень системы вывода был несколько ниже уровня алгоритмического языка, и логический формализм был модифицирован. В результате получилась полная система правил естественного вывода для интуиционистской логики, состоящая из трех правил: описания процедур, применения процедуры и разбора случаев. Каждое из них имеет естественную программную интерпретацию, а фактическое построение формального вывода в этой системе идет более естественно, чем в традиционной. Эта система изложена в [9,II], поэтому повторять здесь мы ее не будем.

При первом же анализе с конструктивной точки зрения стали ясны как те концепции Алгола-68, которые делают его наиболее логически стройным из языков программирования, так и его концептуальные недостатки. Встала задача их устранения и построения нового - логического языка программирования.

Здесь мы покажем процесс перехода от правил конструктивной логики к алгоритмическому языку.

Выделим явно базисные аналоги, лежащие в основе алгоритма извлечения программы из доказательства [8-II]. Доказательство представляется как сеть формул и вспомогательных выводов. Каждая начальная формула в этой сети является аксиомой, все остальные помечены правилами вывода, позволяющими их получить. Вспомогательные выводы имеют такую же структуру, но начальными вершинами могут быть также и допущения. Получить информацию из вспомогательного вывода может лишь та формула, которая непосредственно следует из него. Формулы вспомогательного вывода, передающие информацию наружу, называются результатами. Вспомогательный вывод может быть посылкой лишь одного применения правила вывода.

Правила вывода делятся на описывающие (из логических правил таким является лишь правило описания процедуры) и строящие. Каждое строящее правило может создавать несколько новых объектов, обозначаемых при помощи вспомогательных констант c_i.

Процесс извлечения программы делится на два этапа: классификация объектов вывода на действующие и бездействующие и сборка нужной нам программы из действующих объектов.

Отметим две особенности используемого логического языка. Во-первых, используется язык многосортного исчисления предикатов первого порядка без равенства, сорта переменных (= исходные типы данных алгоритмического языка) частично упорядочены отношением $\leq$, и объект типа $m \leq n$ автоматически приводится к типу n в соответствующих позициях; непустота типов данных не предполагается. Во-вторых, не все используемые предикаты предполагаются вычислимыми. Допускается наличие теоретических предикатов, которым не ставится в соответствие вычислимая булева функция. Первая особенность широко используется в практических формализациях, но в примерах мы иногда будем ограничиваться одним сортом объектов - ob. Вторая особенность вездесуща. В разработанной автором методике формализации даже рекомендуется минимизировать число вычислимых предикатов, и в особенности их использование. Вычислимы должны быть лишь члены исходных дизъюнкций, за исключе-

нием, быть может, одного.

Пример 3. Пусть некоторая вычислительная обстановка [IO] характеризуется следующей системой аксиом (перед каждой конструктивной аксиомой поместим обозначение ее реализации).

AI. $\forall x(A(x) \Rightarrow B(x) \vee C(x) \vee D(x) \vee E(x) \vee F(x))$

A2. $f \textcircled{r} \forall x(B(x) \Rightarrow \exists y K(x,y))$

A3. $g \textcircled{r} \forall x,y(A(x)\&K(x,y) \Rightarrow \exists z L(x,z))$

A4. $h \textcircled{r} \forall x(C(x) \Rightarrow \exists y M(x,y))$

A5. $j \textcircled{r} \forall x(D(x) \Rightarrow \exists y K1(x,y))$

A6. $j1 \textcircled{r} \forall x(A(x) \Rightarrow \exists y K2(x,y))$

A7. $j2 \textcircled{r} \forall x,y,z(K1(x,y)\&K2(x,z) \Rightarrow \exists u L(x,u))$

A8. $k \textcircled{r} \forall x(E(x) \Rightarrow \exists y H1(x,y))$

A9. $k1 \textcircled{r} \forall x,y(H1(x,y) \Rightarrow \exists z(M(x,z)\&P(x,z))$

AIO. $k2 \textcircled{r} \forall x(F(x) \Rightarrow \exists y Q(x,y))$

AII. $k3 \textcircled{r} \forall x,y(Q(x,y) \Rightarrow \exists z Q1(x,y,z))$

AI2. $\forall x,y,z(Q1(x,y,z) \Rightarrow \neg A2(x))$

AI3. $l1 \textcircled{r} \forall x,y(L(x,y)\&A1(x) \Rightarrow \exists z R(x,z))$

AI4. $l2 \textcircled{r} \forall x,y(M(x,y)\&A2(x) \Rightarrow \exists z R1(x,z))$

AI5. $l3 \textcircled{r} \forall x,y(R1(x,y) \Rightarrow \exists z H1(x,z))$

AI6. $l4 \textcircled{r} \forall x,y(P(x,y) \Rightarrow \exists z R(x,z))$

Пусть предикаты B, C, D, E, F вычислимы, остальные – теоретические. Докажем теорему (= цель программы):

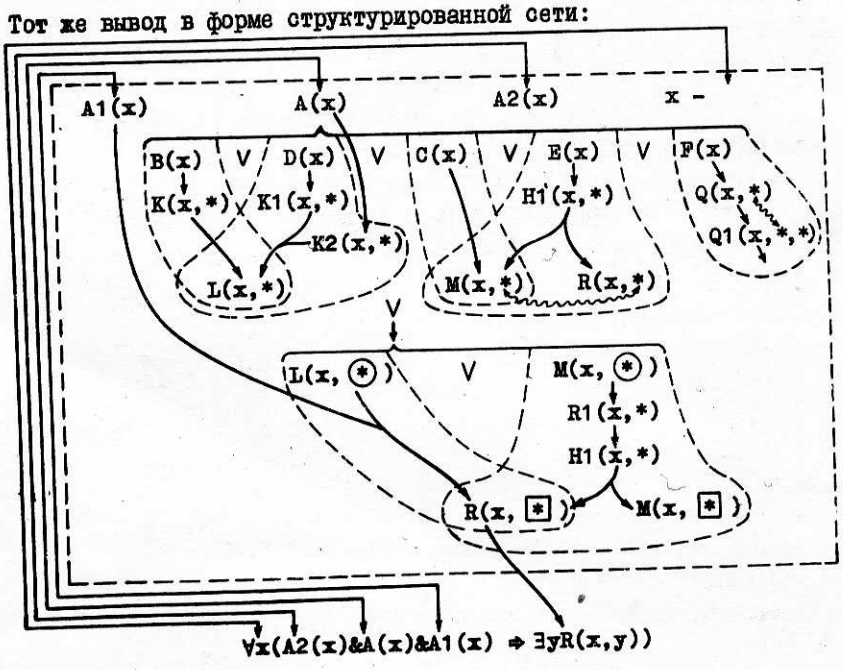
$$\forall x(A(x)\&A1(x)\&A2(x) \Rightarrow \exists y R(x,y)).$$

Приведем доказательство цели в двух формах – в форме Фитча и структурированной сети. Вывод в форме Фитча:

I.	]A(x),A1(x),A2(x)	x – произвольный
2.	|B(x) ∨ C(x) ∨ D(x) ∨ E(x) ∨ F(x)	по AI
3.	|]B(x)	первый случай
4.	||K(x,c1)	по A2
5.	||L(x,c2)	по A3
6.	|]C(x)	второй случаи
7.	||M(x,c3)	по A4
8.	|]D(x)	третий случай
9.	||K2(x,c4)	по A6
IO.	||K1(x,c5)	по A5
II.	||L(x,c6)	по A7
I2.	|]E(x)	четвертый случай

I3.	|H1(x,c7)	по A8
I4.	|M(x,c8),P(x,c8)	по A9
I5.	]F(x)	пятый случай
I6.	|Q(x,c9)	по AI0
I7.	|Q1(x,c9,c10)	по AI1
I8.	|¬A2(x)	по AI2
I9.	|⊥	противоречие
20.	L(x,c11) M(x,c11)	вспомогательное построение
2I.	]L(x,c11)	первая альтернатива
22.	|R(x,c12)	по AI3
23.	]M(x,c11)	вторая альтернатива
24.	|R1(x,c13)	по AI4
25.	|H1(x,c14)	по AI5
26.	|M(x,c15),P(x,c15)	по A9
27.	|R(x,c16)	по AI6
28.	R(x,c17)	что и требовалось
29.	∀x(A(x)&A1(x)&A2(x) ⇒ ∃yR(x,y))	цель

Тот же вывод в форме структурированной сети:



Здесь * обозначают вспомогательные константы, стрелки – логические зависимости, волнистые стрелки – совпадение объектов. Две самые важные для нас константы мы выделили. ⊛ означает c11 , ⊡ – c17 . Анализируя информационные связи, видим, что программа должна зафиксировать построение c11, а затем выдать c17 . Помимо значения c11 , мы должны запомнить, какой из случаев: L или M имел место, поскольку прямо проверить эти предикаты мы не можем. Это – частный случай общей ситуации, когда мы вынуждены восстанавливать невычислимую информацию по косвенным вычислимым признакам.

Ю.М.Сметанин (частное сообщение) отметил наличие аналогии между подобными выводами и конечными автоматами.

§5. Основы логического языка программирования

Имевшиеся до сих пор попытки представления доказательств как программ были неудачны. Причиной неудач чаще всего было либо игнорирование преобладания невычислимых теоретических объектов в логических теориях, либо стремление прямо использовать существующий логический аппарат и языки программирования. Естественное соответствие между доказательствами и программами требует прежде всего наличия прямой связи между правилами вывода логической теории и операторами алгоритмического языка. Каждая конструкция алгоритмического языка должна быть образом некоторой конструкции в логическом выводе. Нельзя допускать ни двусмысленных конструкций, ни реализации одного и того же элементарного шага построения разными путями.

Основные особенности логического языка программирования, вытекающие из анализа конструктивных доказательств, следующие.

1) Отсутствуют понятия переменной и присваивания. Промежуточные результаты запоминаются при помощи описаний констант. Необходимую экономию памяти осуществляет механизм блочной структуры. Поскольку одно правило может выдавать несколько вспомогательных констант, вводятся совместные описания констант, например: `real a,b = sqrt(x↑2+y↑2),x+y;` или `real a, bool b = a1,b1 of f(x,y);`.

2) Концепция, согласно которой каждый оператор выдает значение, доводится до логического конца.

3) Блочная структура доказательства отображается в блочную структуру программы.

4) Иерархия типов данных соответствует иерархии формул в доказательстве. Значениями исходных типов соответствуют вспомогательные константы, сложных типов - формулы, содержащие действующие связанные переменные или действующие невычислимые дизъюнкции (соответствующие размеченным объединениям).

5) Действующие правила вывода взаимно-однозначно отображаются в операторы программы.

6) Оператор языка программирования возникает тогда, когда мы взбираемся на вершину "окольного пути". Напомним, что окольный путь возникает в доказательстве тогда, когда мы применяем формулу, которую сами же доказали при помощи описываемого правила.

7) Бездействующие объекты переходят в "значения-признаки", необходимые для верификации.

8) Информационная и логическая структуры программы неразрывно связаны друг с другом.

Перейдем к описанию некоторых конструкций языка. Они являются результатом оформления в стиле Алгола-68 действующих объектов доказательства. Язык настраиваем на конкретную вычислительную обстановку, поэтому исходные данные и операции (за исключением типов bool и void и логических операций) не фиксируются. Они соответствуют множеству сортов переменных рассматриваемой теории, и, так же, как и сорта, упорядочены отношением $\subseteq$.

Более сложные виды описываются, исходя из исходных, при помощи описаний, аналогичных описаниям вида Алгола-68.Поля структур не упорядочиваются; в объединении каждый его член снабжается идентификатором случая (соответственно, пропадают ограничения на объединение приводимых друг к другу видов).

Пример 4. Программа, извлеченная из примера 3.

```
proc aim=(ob x)ob:
   (union(ob p,q)c11=if B(x) then p:g(x,f(x)) □
                     C(x) then q:h(x) □
                     D(x) then p:j2(x,j(x),j1(x)) □
                     E(x) then q:k1(x,k(x)) □
                     F(x) then error
                  fi;
```

```
new case c11 in p:l1(x,c11) □
                q:l4(x,k1(x,l3(x,l2(x,c11))))
    esac
);
```

Отметим, что символ new отмечает вырабатываемое значение в блоке.

Исполнение программы определяется при помощи трансформационной семантики [2]. Например, исполнение описания констант состоит в исполнении предложения, вырабатывающего значения констант, и замене во всем блоке этих констант на выбранные значения.

В частности, идентификатору процедуры в качестве значения придается изображение процедуры с поставленными значениями глобальных констант. Это процедурное значение может быть выдано как результат блока. Такой подход эквивалентен порождению новых процедур при помощи частичной параметризации. Условные выражения имеют ту форму, которую придал им Дейкстра. Однако мы допускаем и часть else, соответствующую невычислимому члену дизъюнкции. Возражения Дейкстры против такого способа умолчания начисто разрушаются тем фактом, что он сам использовал его в таком центральном месте, как критерий окончания цикла.

Подробнее мы изложим сейчас концепцию неожиданности, ранее отсутствовавшую в языках программирования. Она является обобщением исключений и завершителей цикла в Аде и многих других аналогичных концепций структурного goto.

При построении доказательства мы, как правило, начинаем с фиксации допущений вспомогательного вывода и определения логической структуры его результатов (которые пока что могут содержать пустые места для подстановки еще неизвестных значений). Однако иногда бывает так, что во внутреннем вспомогательном выводе мы натыкаемся на следствие, которое может служить результатом объемлющего. В программе в этом случае целесообразно бросить ту черновую работу, которой мы занимались, и перейти к концу самого крупного из тех блоков, которым удовлетворяет данная цель. При этом необходимо передать нужные значения. Это - приятная неожиданность.

Второй случай неожиданности - когда в ходе исполнения программы мы попали в ситуацию, которая в доказательстве была исключена как приводящая к абсурду. Здесь неприятная неожиданность показывает нам, что мы либо получили неправильные исходные данные, либо сбилась машина, либо наша формализация вычислительной обстановки неверна. Для неприятности целесообразно, наоборот, локализовать ее действие и перейти к ближайшей возможной программе ошибок, опять-таки передав ей нужные данные.

Рассмотрим программное оформление неожиданностей. Естественно напрашивается решение оформлять обработчики неожиданностей как тела процедур, а возбуждение неожиданностей - в том же виде, как и вызов процедуры. Однако при этом теряется главная идея неожиданности, поскольку нас устраивает любая ситуация, в которой предусмотрена реакция на неожиданно появившиеся у нас предикаты. В разных целях могут использоваться разные предикаты, а одни и те же могут появляться в различном контексте.

Поэтому целесообразнее всего отметить возникновение неожиданности, указав предикаты, которые неожиданно выполнились. Их аргументы естественно считать данными, которые в случае необходимости могут быть переданы программе обработки. Предикаты в данном случае выступают как атрибуты данных.

$$(\underline{\frac{good}{bad}})\ \underline{surprise}\ \ P_1(\bar{t}_1),\ldots,P_n(\bar{t}_n);$$

где P_i - идентификаторы предикатов; $\bar{t}_i$ - векторы термов.

Соответственно, чтобы "поймать" неожиданность, необходимо указать предикаты, при выполнении которых может работать программа обработки. Предикаты могут содержать либо выражения, значения которых заведомо известны в момент возникновения неожиданности, либо свободные переменные, которые отождествляются с соответствующими значениями при поимке неожиданности.

$$\underline{when}\ (\underline{\frac{good}{bad}})\ P_1(\bar{r}_1),\ldots,P_k(\bar{r}_k)\ \underline{with}\ \underline{m1}\ x1,\ldots,\underline{ml}\ xl$$

Здесь все члены $\bar{r}_i$ - либо термы, либо переменные из списка x_j.

give S evig

Оператор **when** может стоять в обычном блоке лишь после **new** и выдаваемого значения в обычной ситуации, в операторе

if - в качестве альтернативы.

При обработке приятной неожиданности ищется, начиная с самого внешнего блока, оператор when,список предикатов в котором может быть отождествлен как подсписок списка, указанного в surprise.В случае невозможности отождествления возникшая приятная неожиданность игнорируется.

При неприятной неожиданности поиск ведется, начиная с внутреннего блока, и при отсутствии годного обработчика программа аварийно завершается.

Уже в этом базисном подмножестве логического языка имеются операторы, конструктивная трактовка которых еще не полностью формализована. В частности, существующие правила порождения рекурсивных процедур [9,II] еще неудовлетворительны.

§6. Циклы и массивы

Циклы порождаются из правил математической индукции. В частности, прямым переносом правила индукции по регулярным формулам [8] (т.е. формулам вида $\exists \bar{x}\, \mathcal{A}(\bar{x})$, где $\mathcal{A}$ - нормальна), получаем следующий оператор цикла типа прогрессии

for i, m1 x1,...,mn xn from i0,y1,...,yn by step to lim
give t evig {W} do S rof

Здесь i - главный параметр цикла (целого типа),step - его шаг, lim - конечное значение, xj - побочные параметры, mj - их типы, i0, yj - начальные значения параметров, t - значение, выдаваемое после завершения цикла, {W} - возможная совокупность обработчиков неожиданностей, S - оператор, выдающий новые значения параметров xj .

Пример 5. Числа Фибоначчи.

```
proc fibonacci = (nat n) nat:
fof i,nat x,y from 1,0,1 by 1 to n give y evig do x:y,y:x+y rof
```

При исполнении цикла вначале вычисляются начальные знавения lim и step , затем проверяется условие окончания цикла и, если оно не выполнено, копируется S с заменой i , xj на начальные значения, вычисляется полученная копия и ее результаты заменяют yj , а i0 + step - i0 . После этого выполнение цикла возобновляется. Если значение i превзошло lim, выполнение цикла завершается,и оператор цикла заменяет-

ся на значение t . Аналогично, если возникшая внутри цикла неожиданность поймана одним из его обработчиков, вычисляется соответствующее значение, и цикл завершается. Здесь аппарат обработчиков неожиданностей делает ненужным традиционные условия while. Впервые эта возможность была последовательно использована в языке ЯРМО [24].

Однако более глубокий конструктивный анализ показывает, что логически условия, приводящие к while, выражаются иначе, чем общий случай неожиданности. А именно, они порождаются следующим принципом индукции, сводимым к обычному, но зачастую более удобным.

$$\frac{\begin{array}{l|l} A(0, \overline{\text{co}})\vee B_1 \vee \ldots \vee B_k & \left.\begin{array}{l} A(n, \bar{x}),\ n, \bar{x} - \\ \vdots \\ A(n+1, \bar{t})\vee B_1 \vee \ldots \vee B_k \end{array}\right. \end{array}}{A(m, \bar{c})\vee B_1 \vee \ldots \vee B_k}$$

Здесь все $B_1, \ldots, B_k$ - вычислимы. Проверку условий B_i естественно помещать там же, где и обработку неожиданностей, но выделять более короткой записью: when B_i give r_i evig.

Первым естественным обобщением цикла является замена области значений счетчика на произвольное, заранее заданное, вполне упорядоченное множество. При этом часто главный параметр исчезает вообще, поскольку необходимые проверки легче выполнять, пользуясь неожиданностями и явными условиями цикла. Здесь мы сталкиваемся с программистской интерпретацией принципа Маркова [25] : тело цикла необходимо строить конструктивно, а его завершаемость можно доказывать и классически.

Пример 6. Функция-91 Дж.Маккарти:

```
proc f91 = (nat x)nat:
  for nat n,z from 1, x  when n = 0 give Z evig do
   if z > 100 then n - 1, z - 10 else n + 1, z + 11 fi
  rof
```

Теперь рассмотрим случай, когда множество значений главного параметра проходится нелинейным образом. Здесь возникает много различных видов циклов, один из них - цикл Брауэра, мы рассмотрим подробнее.

Этот цикл соответствует принципу бар-индукции Брауэра (см.,напр., [26]). Пусть X – сеть с конечными путями, множеством начальных элементов Y и множеством конечных элементов Z, ← – отношение непосредственного предшествования. Тогда $\forall x \in Y A(x) \;\&\; \forall x \in X(\forall y \in X(y \leftarrow x \Rightarrow A(y)) \Rightarrow A(x)) \Rightarrow \forall x \in Z A(x)$. Для реализации таких циклов и многих других логически естественных конструкций целесообразно ввести новые сложные типы данных. Например, ими могут быть множества элементов типа m (тип set m), частично-упорядоченных множеств poset m, сетей net m, и т.д.

Над этими типами данных ведем естественные операции. В частности, мы используем ограничение множества set union (b1 m1,...,bk mk)x на фиксированный атрибут bi, обозначаемое x1↑bi, отношения $\subseteq$, =, $\in$, $\leq$ (для частично-упорядоченных множеств), ← (для сетей).

Теперь можно записать общую форму цикла Брауэра, или цикла типа потока данных:

bar for x∈X,m1 x1,...,mk xk from Y give t evig{W} do S rof

Здесь Y – множество начальных значений; X – сеть значений главного параметра; S – предложение, вырабатывающее новые значения побочных параметров; W – обработчики неожиданностей, t – выдаваемое в случае нормального завершения значение.

Опишем теперь исполнение цикла. Y должно быть предложением, вырабатывающим массив, чьими элементами являются структуры значений xi, а индексами – начальные элементы сети X. Вслед за вырабатываемым значением Y мы имеем право поставить явную проверку соответствия его множества индексов требуемому множеству X0 : compare X0 bad surprise...,возбуждающую в случае неравенства сравниваемых множеств плохую неожиданность.

После инициализации и проверки Y мы заменяем Y на его значение, выбираем некоторое X, не входящее в range Y, но такое, что все непосредственно предшествующие ему элементы сети входят в него. Если X оказался конечным элементом сети, исполнение цикла завершается исполнением t, в противном случае исполняется S. Внутри S на все x_i ссылаемся как на элементы массивов, индексируемых y ← x. В случае, если цикл еще не завершается, выработанные значения присоединяются к Y

Y как элементы с индексом x, и из Y удаляются те элементы, индексы которых имели единственную использованную связь - с x .

Пример 7. Вычисление значения арифметического выражения, заданного как сеть, при данных значениях переменных.

```
mode aex = net union
     (string var, struct (ref a,b) plus, minus, mult, dev);
proc compute = (aex a, [set string] int enriron)int:
     (set string avar = a ↑ var;
      new
      bar for  x ∈ a, int i from environ {avar}
          compare aver bad surprise q(avar, range environ)
          give i evig do
            case x in plus: i[a of x] + i[b of x]
                     minus: i[a of x] - i[b of x]
                      mult: i[a of x] × i[b of x]
                       div: i[a of x] ÷ i[b of x]
          esac
      rof
      when bad q(x1, x2) withe set string x1, x2 give
          signal ("не заданы значения", x1-x2); skip evig
     );
```

Здесь range a - множество индексов массива a, a{z} - подмассив a, получаемый ограничением множества индексов до z .

Этот пример иллюстрирует некоторые приемы работы с массивами. Как правило, массивы обрабатываются специально приспособленными для этого операторами. Однако, в соответствии с общей концепцией ортогональности, диктуемым логическим подходом, массивы являются полноправными значениями и могут использоваться и в обычных операторах. Кроме того, мы обязаны иметь возможность создавать начальное значение массива.

Общеизвестно, что массив формализуется как функция, сопоставляющая индексам значения. Однако в обычных, дескриптивных, формализациях остается неясно, чем же массив отличается от процедуры. С конструктивной точки зрения процедуре соответствует формула $\forall\bar{x}(A \Rightarrow \exists\bar{y}B)$. Известное правило свертки

позволяет от такой формулы перейти к $\exists \bar{f} \forall \bar{x}(A(\bar{x}) \to !\bar{f}(\bar{x}) \,\&\, B(\bar{x}, \bar{f}(\bar{x}))$, где функция выбора задана явно. Однако в §8 будут показаны недостатки явного задания функций по сравнению с неявным. Эти недостатки несущественны лишь в том случае, если вычисление функции почти ничего не стоит, т.е. она предвычислена и превращена в массив. Естественно, что это возможно лишь в том случае, когда $A(\bar{x})$ есть $\bar{x} \in A_1$, где A_1 – финитное множество.

Приведенные выше примеры достаточно ясно показывают обоснованность выдвижения следующей гипотезы. Каждое разумное логическое правило имеет простую и естественную программистскую интерпретацию, и наоборот, каждый достаточно общий программистский прием может быть выведен из конструктивного логического правила. Однако поиск таких аналогий часто требует необычной точки зрения.

§7. О разрешимости конструктивных теорий

Проблема поиска вывода в конструктивных теориях пока что абсолютно неразработана. Как будет показано в следующем параграфе, вариации на тему метода резолюций здесь бесперспективны. Однако прежде чем искать выход, нужно хорошо понять, что искать, зачем искать и где искать. Поэтому проблема поиска вывода тесно связана с поиском хорошей методики формализации. Здесь мы покажем некоторую классификацию формул, дающую возможность задавать теории в "стандартизованной" форме и выделять некоторые разрешимые подклассы. Отметим, что вся эта классификация основана на семантических соображениях, но выражена синтаксически.

Пусть $\mathcal{A}$, $\mathcal{B}$, $\mathcal{L}$ – конъюнкции элементарных формул, $\mathcal{A}(\bar{\bar{x}})$ означает, что в любой предикат из $\mathcal{A}$ входят все переменные из вектора $\bar{x}$, $\mathcal{A}(\bar{x})$ – хотя бы одна из $\bar{x}$.

Выделим следующие классы формул.

1. Факты. Элементарные замкнутые формулы и их отрицания.
2. Связи. $\forall \bar{x}(\mathcal{A}(\bar{x}) \to \mathcal{B}(\bar{\bar{x}}))$.
3. Классификация. $\forall \bar{x}\,(\mathcal{A}(\bar{x}) \to \mathcal{B}_1(\bar{\bar{x}}) \vee \ldots \vee \mathcal{B}_k(\bar{\bar{x}}))$.
   Все $\mathcal{B}_1, \ldots, \mathcal{B}_k$, кроме, быть может, одного вычислимы.
4. Опровержения. $\forall \bar{x}\, \neg \mathcal{A}(\bar{\bar{x}})$.
5. Построения. $\forall \bar{x}\,(\mathcal{A}(\bar{x}) \to \exists \bar{y} \mathcal{B}(\bar{x}, \bar{y}))$.

5а. Сложные построения. $\forall\bar{x}(\mathcal{A}(\bar{x})\&\mathcal{V}\to\exists\bar{y}\mathcal{B}(\bar{x},\bar{y}))$, где $\mathcal{V}$ – конъюнкция (возможно, пустая) сложных построений.

6. Признаки первого рода. $\forall\bar{x}\bar{y}(\mathcal{A}(\bar{x},\bar{y})\to\mathcal{B}(\bar{x}))$.

7. Признаки второго рода. $\forall\bar{x}(\forall\bar{y}\,\mathcal{A}(\bar{x},\bar{y})\to\mathcal{B}(x))$.

8. Признаки третьего рода. $\forall\bar{x}((\mathcal{A}(\bar{x})\to\mathcal{B}(\bar{x}))\to\mathcal{L}(\bar{x}))$.

Теорию, каждая аксиома которой имеет один из девяти перечисленных выше видов, назовем стандартизованной. Приведем без доказательства несколько результатов, показывающих роль стандартизованных формул и теорий.

Лемма 1. Если дан вывод, состоящий лишь из стандартизованных формул, то при извлечении из него программы связи полностью игнорируются, отрицательные факты и опровержения приводят к исключению некоторых альтернатив из условных операторов; построения порождают термы и описания констант; классификации порождают условные операторы; признаки порождают призраки, т.е. бездействующие конструкции, не переходящие в программу.

Лемма 2. Для любой конструктивной теории T можно построить ее стандартизованное консервативное расширение T', не содержащее сложных построений, и такое соответствие между выводами в T и в T', что каждому выводу формулы A в T соответствует вывод в T', длина которого не более чем в три раза больше, чем у исходного.

Однако соответствие между выводами, существование которого доказывается в лемме 2, не обязательно сохраняет извлекаемые программы. Доказать сохранение программ удается лишь при дополнительном, но достаточно естественном, условии на T: все конструктивные аксиомы имеют вид $\forall\bar{x}(\mathcal{A}(\bar{x})\to\exists\bar{y}\mathcal{B}(\bar{x},\bar{y}))$, где $\mathcal{B}$ – нормальна (т.е. полунормальны [9]).

Содержательно это условие означает, что все исходные функции выдают в качестве значения структуры элементов исходных типов.

Лемма 3. Если все аксиомы T нормальны или полунормальны, то можно построить стандартизованное консервативное расширение T' и соответствие между выводами, сохраняющее извлекаемые программы и увеличивающее длину вывода не более, чем в три раза.

Отметим, что мы все время сравниваем между собой не множества выводимых формул различных теорий, а отрезки этих множеств, задаваемые выводами ограниченной сложности. Принципиальная эквивалентность понятий для нас никакой роли не играет, важно иметь простое преобразование (ср. замечания Г.Крайзеля [4]).

Назовем несокращающими такие построения, у которых в любом предикате из $\mathcal{L}$ количество различных переменных не меньше, чем в префиксе $\forall\bar{x}$. Сложное простроение несокращающее, если то же условие выполнено после добавления к количеству членов $\bar{x}$ количества результатов всех построений из $\mathcal{U}$, и все посылки – построения построений из $\mathcal{U}$ также несокращающие.

Теорема. Если T – конечная стандартизованная теория, не содержащая признаков первого рода, все построения и сложные построения в которой несокращающие, то проблема выводимости для построений в T разрешима.

Следствие. Проблема выводимости разрешима и для сложных построений, все посылки–построения которых несокращающие.

Беспризнаковые теории естественно появляются при формализации пакетов прикладных программ, и именно для них естественно строить алгоритмы поиска вывода. Подбор признаков целесообразно возложить на человека, требуя от него подсказок в диалоговом режиме.

§8. Что не нужно делать?

Этот параграф посвящен дискуссии. Но в ней мы будем отремиться приводить в качестве аргументов точные теоремы даже там, где до сих пор сравнение производилось лишь при помощи зравого смысла, философских соображений и ограниченного числа экспериментов.

Обсудим, как и было обещано, метод резолюций с точки зрения пригодности для синтеза программ. Для его нетривиальных применений в данной области существует пять серьезных препятствий.

Первое препятствие. Невычислимые сколемовские функции.

Как известно, перед применением метода резолюций мы должны произвести предварительную обработку формул, а именно,

сначала устранить все кванторы, заменяя их сколемовскими функциями (сколемизация), а затем преобразовать формулу к виду конъюнкции дизъюнктов. Уже эта предварительная обработка может полностью разрушить вычислимость результата.

Пример 8. Рассмотрим формулу $\forall x(\exists y B(x,y) \to \exists z A(x,z))$. Приведя ее к предварительной форме, получаем $\forall x \exists z \forall y (B(x,y) \to A(x,z))$. Сколемизируя ее отрицание и разбивая на дизъюнкты, получаем $B(c,f(z))$, $\neg A(c,z)$. Теперь при синтезе программы методом резолюций мы должны проследить унифицирующую подстановку для z при выводе пустого дизъюнкта. Но в эту подстановку может попасть f, которое нам не дано и которое неизвестно как вычислять.

В данном случае для устранения препятствия достаточно было избрать другой порядок выноса кванторов, а именно: $\forall x \forall y \exists z(B(x,y) \to A(x,z))$, но в следующем примере ничто не поможет.

Пример 9. Рассмотрим формулу $\forall x \exists y \forall z\, A(x,y,z)$. После предобработки она переходит в $\neg A(c,y,f(y))$. Здесь f принципиально вычислима, неустранима, но может войти в решение.

Способ борьбы с этим препятствием: ограничть класс унифицирующих подстановок для результатов создаваемой процедуры и строго кодифицировать порядок выноса кванторов при сколемизации.

Второе препятствие. Неадекватность вычислимых сколемовских функций.

Пример 10. Для формулы

$$\forall x(\forall y \exists z\, A(x,y,z) \,\&\, \forall y' \exists z'\, B(x,y',z') \Rightarrow \exists u C(x,u))$$

мы имеем три существенно различных варианта предваренной формы:

$$\forall x \exists y \forall z \exists y' \forall z' \exists u (A(x,y,z) \& B(x,y',z') \to C(x,u))$$

и два других с префиксами $\forall x \exists y, y' \forall z, z' \exists u$ и $\forall x \exists y' \forall z' \exists y \forall z \exists u$. После предобработки получаем либо $A(c,y,f(y)), B(c,y',g(y,y')), \neg C(c,u)$, либо $A(c,y,f(y,y')), B(c,y',g(y,y')), C(c,u)$, либо $A(c,y,f(y,y')), B(c,y',g(y)), \neg C(c,u)$. Хотя бы одна из сколемовских функций содержит лишний аргумент.

Пример 11. Смысл формулы $\forall x((\forall y \exists z A(x,y,z) \to \exists u B(x,u)) \to \exists v C(x,v))$ состоит в том, что v должен быть построен как результат процедуры, параметром которой служит процедура по-

строения u, а ее параметром - процедура построения z по y при данном x. Следовательно, конструктивная расшифровка (т.е. правильная сколемизация) должна быть

$$\exists\phi\forall x,\varphi(\forall f(\forall yA(x,y,f(y)) \Rightarrow B(x,\varphi(f)(x))) \Rightarrow C(x,\phi(x,\varphi))).$$

Классическая предваренная форма имеет кванторный префикс $\forall x,y\,\exists z\forall u\,\exists v$. Обработка приводит к дизъюнктам $\neg C(c_1, v)$, $A(c_1,c_2,z)\vee B(c_1,f(z))$, Но, как было показано, f должно применяться не к самому z, а к процедуре, строящей z (возможно, при этом необходимо варьировать значение y).

Эту трудность преодолеть тяжело. Однако следующая теорема показывает, что это возможно в принципе.

Теорема о конструктивной сколемизации.

а) Можно построить такой естественный (т.е. определяемый рекурсией по построению формулы) алгоритм $\mathscr{P}$, преобразующий любую формулу A к виду $\exists\alpha_A\,\exists\beta_A\forall_{\gamma_A}SA(\alpha_A,\beta_A,\gamma_A)$, где SA - бесквантовая формула языка работы [28], а α_A, β_A, γ_A - переменные по функционалам (возможно, высокого уровня).

б) A интуиционистски выводима в теории T тогда и только тогда, когда можно найти такие аксиомы $B_1,\ldots,B_n \in T$, такие термы $t_i^1, \ldots, t_i^{k_i}$, соответствующие по типу γ_{Bi}, такой терм r, соответствующий α_A, и зависящий лишь от x_i, терм s, соответствующий β_A и не зависящий от c, что $SA(r,s,c)$ выводимо из $SB_1(x_1,y_1,t_1^1),\ldots,SB_1(x_1,y_1,t_1^k);\ldots;SB_n(x_n,y_n,t_n^1)$, $\ldots,SB_n(x_n, y_n, t_n^{kn})$ в классическом исчислении высказываний с правилами конверсии термов.

Эта теорема была опубликована в [27]. Для уяснения значения ее формулировки нам достаточно знать, что исчисление термов из [28] подобно исчислению λ-конверсий с типами, отличаясь добавлением типов прямых произведений и прямых сумм и условных термов.

Пример I2. Конструктивная сколемизация формулы примера II есть

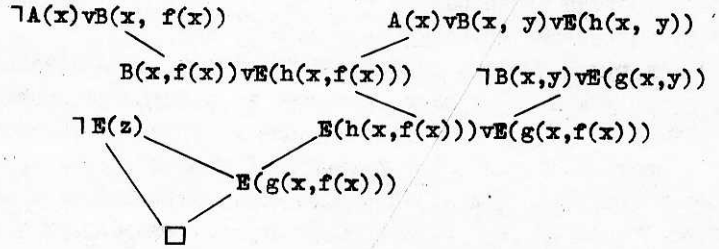
$$\exists\Psi\,\exists v_5, v_1, v_4\forall x,\varphi, v_2, v_3$$
$$((A'(x, v_3(v_5(x,\varphi)), v_5(x,\varphi)(v_3(v_5(x,\varphi))),$$
$$v_1(x,\varphi)(v_3(v_5(x,\varphi)))) \Rightarrow$$
$$B'(x,\varphi(v_5(x,\varphi))(x),v_2(v_5(x,\varphi))(v_1(x,\varphi)))) \Rightarrow$$
$$C'(x, \Psi(x,\varphi), v_4(x,\varphi)(v_2, v_3)))$$

Здесь новые аргументы, введенные в предикаты, имеют тип ghost, который считается отсутствующим в исходном языке.

Итак, плата за желание сохранить в конструктивной обстановке привычные методы работы может быть очень высока: необходимость работы с произвольными λ-термами.

Третье препятствие. Не всюду определенные функции. Часто конкретные процедуры, реализующие сколемовские функции, определены лишь для некоторых значений аргумента. При этом проверка применимости функции может быть не легче ее применения (например, проверка невырожденности системы линейных уравнений). В этом случае у нас может возникнуть дизъюнкт ¬A(x)vB(x, f(x)), где f(x) применимо (!f(x)),лишь если A(x).

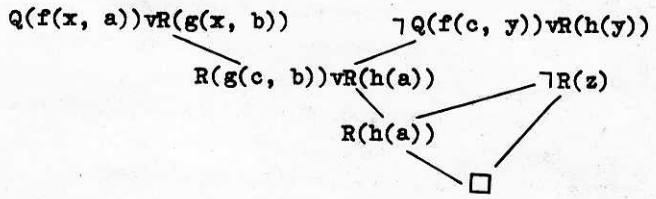
Пример I3. Рассмотрим вывод (E(z) - цель, z - переменная-ответ).



Синтезируемая программа if B(x,f(x)) then g(x,f(x)) else h(x,f(x)) fi.Однако проверка условия невозможна из-за возможной неопределенности f(x).

Четвертое препятствие. Наличие теоретических предикатов.

Пример I4. Пусть Q - теоретический предикат, z - переменная-ответ. Рассмотрим вывод

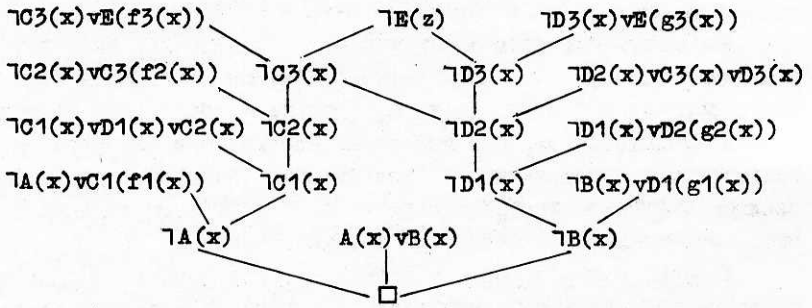


Синтезируемая программа для z здесь есть
if Q(f(c, a)) **then** h(a) **else** g(c, b) **fi**, но проверять Q мы не можем.

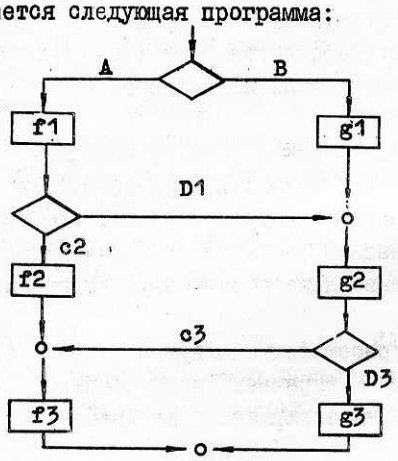
Оба этих препятствия носят менее принципиальный характер, чем изложенные выше, но приносят гораздо больше неприятностей на стадии поиска вывода. Для их обхода необходимо тщательно проверять все унифицирующие подстановки.

Пятое препятствие. Отсутствие структурированности доказательства. В методе резолюций естественное членение вывода на подвыводы отсутствует. С одной стороны, это делает получившуюся программу "плоской" (отсутствует возможность вводить вспомогательные процедуры), с другой стороны, позволяет породить неструктурные программы.

Пример I5. Рассмотрим вывод



Из него извлекается следующая программа:



Эту трудность можно преодолеть подходящей тактикой поиска вывода.

После того, как часть из указанных выше дыр залатана, мы получаем типичную программу синтеза при помощи метода резолюций (примеров здесь можно привести много, но почти все они производят откровенно удручающее впечатление, поэтому ссылок не приведем). Естественно, что, помимо ограничений, явно наложенных автором программы, она будет содержать многие ограничения, о которых он не подозревал (как всегда бывает при работе апробированным, но не очень подходящим к задаче методом). Однако остается еще один вопрос. В методике резолюций предобработка переводит все неявно содержавшиеся в теории и доказываемом утверждении функции в явную форму. В функциональной системе, наоборот, предпочтение отдается неявному представлению функции. Для сравнения этих двух представлений оценим их влияние на вывод и качество извлекаемой программы.

Используя алгоритм конструктивной расшифровки, аналогичный клиниевскому (см., например., [II]), мы можем привести любую формулу A к виду $\exists\alpha_A КА(\alpha_A)$, где КА - нормальная формула.

Получившееся α_A без изменения переходит в алгоритм γ из теоремы о конструктивной сколемизации. Заменим теперь все аксиомы $B \in T$ на $КВ(\xi_B)$, где ξ_B - новые константы того же типа, что и α_B. Получившуюся теорию обозначим КТ.

Теорема о явной форме. Доказательство формулы $\exists\alpha_A КА(\alpha_A)$ в теории КТ без увеличения длины и сложности перестраивается в доказательство формулы A в теории A. Программа, извлекаемая из второго доказательства, является результатом оптимизации программы, извлекаемой из первого, путем предвычисления повторяющихся термов.

Таким образом, неявное задание оказывается лучше.

Аналогично можно показать, что логически легче создавать максимально распараллеленную программу, чем последовательную. Соответственно, задача распараллеливания программ, уже уложенных в прокрустово ложе последовательного языка, несколько дискредитируется.

И, наконец, рассмотрим модную в последнее время задачу верификации. Если A выражает задачу синтеза программы, то КА(t) выражает задачу верификации программы t.

Таким образом, теорема о явной форме влечет, в частности, что задача верификации не проще задачи синтеза, и, более того, единственный путь полной верификации программы – восстановить ее построение, повторить ее синтез. Здесь уже можно прямо сказать, что задача полной верификации практически бесполезна. Как уже было сказано в §2, задача частичной верификации естественно возникает в цикле синтеза программы.

§9. Заключение

Данная статья – первый опыт изложения логического подхода как единой концепции теории программирования. Только сейчас становится ясно, насколько широки неиспользованные возможности и насколько радикальной перестройки программного и математического мышления он требует. В принципе, программирование должно слиться с математикой. Это слияние автор видит на базе возрожденного и перестроенного конструктивного подхода – эмпирического конструктивизма.

Литература

1. Цейтин Г.С. Нематематическое мышление в программировании. "Перспективы системного и теоретического программирования", Новосибирск, 1979, стр. 128–132.
2. Ершов А.П. О сущности трансляции. – Программирование, 1977, №5, стр. 21–39.
3. Марков А.А. О конструктивной математике. Тр. МИАН СССР, вып. 67, 1962, стр. 8–14.
4. Kreisel G. Some uses of proof theory for finding computer programs.Colloq.Intern.Log.,Clermont-Ferrant, 1975, p.151.
5. Дал У., Дейкстра Э., Хоар К. Структурное программирование. – М.: Мир , 1975. 248 стр.
6. Попов Э.В., Фирдман Г.Р. Алгоритмические основы интеллектуальных роботов и искусственного интеллекта. "Наука",М. 1976.
7. Клини С.К. Введение в математику. – М.: ИЛ, 1957, 52 стр.
8. Непейвода Н.Н. Соотношение между правилами естественного вывода и операторами алгоритмических языков высокого уров-

ня. ДАН СССР, т. 239, 1978, №3, стр. 526–529.
9. Непейвода Н.Н. О построении правильных программ. "Вопросы кибернетики", вып. 46, 1978, стр. 88–122.
10. Непейвода Н.Н. Об одном методе построения правильной программы из правильных подпрограмм. "Программирование", 1979, №1, стр. 11–21.
11. Непейвода Н.Н. Применение теории доказательств к задаче построения правильных программ. "Кибернетика", 1979, №2, стр. 43–46.
12. Manna Z., Waldinger R. The logic of computer programming. IEEE Trans. on Software Engincering,vol.SE-4,no.5 (1978).
13. Manna Z., Waldinger R. The synthesis of structure-changing programs.Proc. of the 3-d Intern.Conf. on Software Engineering, May 1978.
14. Manna Z., Waldinger R. A deductive approach to program synthesis.Proc.6-th Intern.Conf.on AI,Tokoy, 1979, p.542.
15. Расева Е., Сикорский Р. Математика математики.–М.:Наука, 1972,–591 с.
16. Тыугу Э.Х. Система программирования с автоматическим синтезом алгоритмов."Тр.Всесоюзного симпозиума по методам реализации новых алгоритмических языков", вып. 2, Новосибирск, 1975, стр. 94–108.
17. Darlington J. A synthesis of several sorting algorithms. Acta Informatica, 1979, vol. 12, no. 1, p. 1.
18. Ершов А.П. Смешанные вычисления: потенциальные применения и проблемы исследования. "Методы математической логики в проблемах искусственного интеллекта и систематическое программирование", ч.2, Вильнюс, 1980, стр. 26–55.
19. Burstall R.M., Darlington J. A transformation system for developting recursive programs.J.of ACM,vol.24,no.1(1977),p.44.
20. Барадинь Я.М. Об индуктивном синтезе программ (В этом сборнике, т. I , стр. 343–364.
21. Непейвода.Н.Н. Конструктивные логики ограниченных построений. "Релевантные логики и теория следования", М.,1979, стр. 76–80.
22. Бельтюков А.П. Формальная теория для порождения правильных программ заданной вычислительной сложности. "Методы математической логики в проблемах искусственного интеллек-

та и систематическое программирование", ч.I, Вильнюс,I980, стр. 64–66.
23. Непейвода Н.Н. Логическое программирование. М., Научн. совет по кибернетике АН СССР, I980. I6 стр.
24. Гололобов В.И.,Чеблаков Б.Г., Чинин Г.Д. Описание языка ЯРМО. Препринт 247, ВЦ СО АН СССР, Новосибирск, I980.
25. Марков А.А. Об одном принципе конструктивной математической логики. "Тр. 3-го Всес. математ. съезда", т.2 АН СССР, I956, стр. I46–I47.
26. Гейтинг А. Интуиционизм. - М.: Мир, I965, 200 стр.
27. Nepeivoda N.N. The connections between the proof theory and computer programming.6-th Intern.Congr.of Logic,Methodology and Philosophy of Science,Hannover,1979,v.1,p.7–11.
28. Nepeivoda N.N. A proof theoretical comparizon of program synthesis and program verification.6th Intern.Congr.of Logic,Methodology and Philosophy of Science,Hannover,1979,
29. Bauer F.L. et al. Report on a mide spectrum language for program specification and development.TUM-I 8104.München, TUM Institut für Informatik, May 1981, 236 p.
30. Цейтин Г.С. От логизма к процедурализму. (В этом сборнике т.2 , стр. I8I–I93.
3I. Кнут Д.Алгоритмы в современной математике и вычислительной науке (В этом сборнике, т.I, стр. 64–99).