К ТЕОРИИ СИНТЕЗА ПРОГРАММ

Н. Н. НЕПЕЙВОДА, Д. И. СВИРИДЕНКО

§ 1. ВВЕДЕНИЕ

В последнее время в языках программирования начинают занимать все большее место дескриптивные структуры, зачастую вытесняя чисто алгоритмические [1, 2]. Кроме того, сама форма представления алгоритмических структур претерпела большие изменения в связи с появлением структурного программирования и осознанием того, что алгоритм существует не сам по себе, а лишь как выражение наших мыслей о разумных способах преобразования некоторой информации [3]. Языки программирования становятся, как это принято сейчас говорить, языками спецификаций алгоритмов, имея в виду модально-повествовательный характер таких спецификаций вместо императивов алгоритмических языков. На наш взгляд, «... рост относительной стоимости программирования и отладки в общей стоимости применения ЭВМ» [4] не есть главная причина такого развития. Скорее причины кроются в необходимости приобщения к ЭВМ как можно большего числа пользователей, для которых чисто «алгоритмический стиль мышления» не всегда является привычным, а также желание выделить в программистской деятельности творческие элементы, оставив рутинные операции машине.

Так или иначе, возникает задача перехода от описания задания, в котором существенное место занимают дескриптивные элементы и алгоритмическая сущность которых неочевидна, к чисто алгоритмическому, которое уже может быть воспринято ЭВМ и выполнено ею. Такую задачу будем называть *задачей синтеза программ.* Примерами задач синтеза могут служить вопросно-ответные системы, решатели задач, генераторы отчетов, трансляторы таблиц решений, системы управления базами данных (СУБД), пакеты прикладных программ и т. п. [5].

Развитие программирования позволяет считать задачу синтеза программ весьма актуальной. Однако имеющиеся подходы к ее анализу показывают нетривиальность такой задачи. В связи с этим возникает необходимость глубокого теоретического исследования задачи синтеза и соответственно создания *теории синтеза программ.*

Первые две работы, посвященные теоретическому исследованию задачи синтеза программ, относятся к началу 70-х годов [6, 7], хотя исследования в области логики построения алгоритмов начались намного раньше. Обе эти работы резко отличаются друг от друга и фактически предопределили два различных пути дальнейших исследований. В основу работы З. Манна и Р. Уолдингера [6] был положен тот факт, что богатый набор методов и приемов синтеза программ по их описанию дает практика программирования. Они продемонстрировали возможность синтеза циклических и рекурсивных программ. Недавно появившаяся работа [8] как бы подводит итог почти десятилетних исследований в этом направлении, которое мы будем называть *эвристическим.*

Другая работа, статья Р. Констейбла [7], указала на возможность иного подхода — назовем его *логическим.* Констейбл отметил, что задача синтеза программ имеет самое непосредственное отношение к интуиционизму, поскольку если на описание задания смотреть как на теорему соответствующей конструктивной теории, то возможность существования требуемого текста программы следует из теоремы Клини о существовании реализации. Необходимо отметить, что еще в 1968 г. А. А. Марков во вводной лекции курса «Конструктивная математическая логика» обосновал возможность и целесообразность применения конструктивной

логики к программированию. Одновременно им были глубоко проанализированы те черты традиционной конструктивной математики, которые мешают ее приложениям, и была намечена программа работ в данном направлении. К сожалению, это выступление осталось неопубликованным.

Оба эти подхода, логический и эвристический, представляют собой крайние варианты решения *дедуктивной* задачи синтеза. Другие подходы к данной задаче фактически являлись либо модификациями одного из вышеуказанных, либо попыткой учесть сильные стороны каждого из них [8—22]. Из-за того, что зачастую сама задача ставится в операционных терминах, возникают еще два подхода к синтезу: *трансформационный* и *индуктивный*, реализирующие переход от одних функциональных спецификаций к другим. Так, трансформационный [23] реализует переход от вычислимости к вычислениям, от абстрактной функциональной спецификации к конкретной. Индуктивный подход [24—30] реализует обратный переход: от вычислений к вычислимости, от конкретных примеров действия программы к алгоритму. Отметим, что оба эти подхода в последнее время приобрели заметную дедуктивную окраску. В дальнейшем речь будет идти о дедуктивной задаче синтеза, а точнее — о логическом подходе к ее решению.

§ 2. ЗАДАЧА СИНТЕЗА ПРОГРАММ

Назначение программы — служить инструментом решения задачи. Поэтому естественно предварительно проанализировать понятие «задачи» и процесса ее решения.

Задача предполагает прежде всего существование определенной *цели* и необходимость поиска средств ее достижения. Будем называть процесс нахождения такого средства *решением* задачи, а само найденное средство — *реализацией*. Полная формулировка задачи включает в себя следующие компоненты:

а) характеристику природы *исходных данных* и соотношений между ними;

б) характеристику *результатов*, их взаимосвязи и связи с исходными данными;

в) описание *средств*, при помощи которых должна быть построена реализация; это описание, в свою очередь, распадается на описание исходных операций, их взаимосвязи и наших знаний о них, допустимых способов их композиции;

г) *предписания*, ограничивающие класс программ, которые могут служить реализацией решения, путем навязывания нам структуры будущего решения, методов, которые должны быть использованы и т. д.;

д) *ограничения*, накладывающиеся на ресурсы либо полностью запрещающие их использование; ресурсами могут быть время, память, спецпроцессоры и т. п.

Наиболее традиционный случай — когда части г) и д) отсутствуют и часть в) задается некоторой *вычислительной обстановкой*, т. е. конструктивной (в традиционном смысле [31]) формальной теорией T в языке (пополненного) исчисления предикатов, снабженной библиотекой стандартных процедур, реализующих конструктивные аксиомы. В этом случае задаче можно соотнести следующую цель:

«для каждого набора значений исходных данных, удовлетворяющих некоторым требованиям, найти набор значений неизвестных переменных, удовлетворяющих заданным требованиям, а также условию, характеризующему связь между исходными данными и неизвестными».

В языке исчисления предикатов эта цель может быть формализована так:

«Доказать предложение ψ в теории T, где ψ имеет вид

$$\psi = \forall \bar{x}(A(\bar{x}) \Rightarrow \exists \bar{y} B(\bar{x}, \bar{y})),$$

$\bar{x}$, $\bar{y}$ — конечные наборы переменных».

Однако речь идет не просто о демонстрации того, что формула ψ выполняется на некоторой модели $\mathfrak{A}$, а о конструировании алгоритма f, который по набору значений $[\bar{x}] \in |\mathfrak{A}|^n$, для которого верно $A([\bar{x}])$, вычисляет такое значение $f([\bar{x}]) \in |\mathfrak{A}|^m$, что выполняется условие $B([\bar{x}], f([\bar{x}]))$. Такая постановка, как легко убедиться, в точности соответствует понятию реализуемости в интуиционистской логике. В том случае, когда интерпретацию типов данных можно считать нумерованными множествами [32], мы можем дать следующее определение реализуемости, копирующее идею определения Клини [33] для арифметики.

Пусть $\mathfrak{N}$ — совокупность нумерованных множеств, замкнутая относительно прямой суммы и прямого произведения. Предположим также, что для любых $\mathfrak{A}, \mathfrak{B} \in \mathfrak{N}$ существует главная вычислимая нумерация семейства морфизмов Мог $(\mathfrak{A}, \mathfrak{B})$. Примером такой совокупности $\mathfrak{N}$ может служить λ-модель C^*_{20} (см. [32]).

Пусть L — язык исчисления предикатов. Сопоставим каждой формуле A некоторое нумерованное множество $\mathfrak{A} \in \mathfrak{N}$ и отношение $p \vdash A$ («p подтверждает A»), где $p \in \mathfrak{A}$. Сопоставление определим индуктивно:

1. Для каждой элементарной формулы A множество $\mathfrak{A}$ и отношение $p \vdash A$ считается заданным.

2. Если C имеет вид $A_1 \& \ldots \& A_n$, то сопоставим ей нумерованное множество $L = \mathfrak{A}_1 \times \ldots \times \mathfrak{A}_n$ и $\langle p_1, \ldots, p_n \rangle \vdash C \rightleftharpoons \forall i \leqslant n (p_i \in \mathfrak{A}_i \Rightarrow p_i \vdash A_i)$.

3. Если C имеет вид $A_1 \vee \ldots \vee A_n$, то сопоставим ей нумерованное множество $L = \bigoplus_{i=1}^{n} \mathfrak{A}_i$ (прямая сумма), где для каждых $i \neq j$ $\mathfrak{A}_i \cap \mathfrak{A}_j = \varnothing$ и

$$p \vdash C \rightleftharpoons \exists i \leqslant n (p \in \mathfrak{A}_i \& p \vdash A_i)$$

4. Если C имеет вид $(A \Rightarrow B)$, то сопоставим ей нумерованное множество L, состоящее из всех вычислимых морфизмов из $\mathfrak{A}$ в $\mathfrak{B}$, $L = $ Мог $(\mathfrak{A}, \mathfrak{B})$, снабженное главной вычислимой нумерацией, и

$$p \vdash C \rightleftharpoons \forall p' \in \mathfrak{A} (p' \vdash A \to p(p') \vdash B).$$

5. Если C имеет вид $\neg A$, то сопоставим этой формуле $L = \{\bot\}$ и

$$\bot \vdash A \rightleftharpoons \neg \exists p \in \mathfrak{A} (p \vdash A).$$

6. Если $C = \forall x A(x)$, то сопоставим ей $L = $ Мог $(V, \mathfrak{A})$, где V — множество значений переменной x, и

$$p \vdash \forall x A(x) \rightleftharpoons \forall v \in V (p(v) \vdash A(v)).$$

7. Если $C = \exists x A(x)$, то $L = V \times \mathfrak{A}$ и

$$\langle v, p \rangle \vdash \exists x A(x) \rightleftharpoons p \vdash A(v).$$

Если функция f может быть представлена в терминах некоторого алгоритмического языка, то это представление и является искомой *программой — реализацией*. Задача синтеза предполагает нахождение такой программы конструктивно. Поэтому ставится следующая задача:

«Указать тот класс ситуаций, в которых возможно эффективное нахождение программы — реализации формулы ψ».

Заметим, что прежде всего формула ψ должна быть выполнимой на модели $\mathfrak{A}$. Синтаксической гарантией выполнимости предложения является его доказательство в некоторой теории T, при этом такая гарантия в определенных случаях может быть эффективной. Если, кроме того,

потребовать *конструктивность* теории T, то тогда мы будем иметь в своем распоряжении алгоритм, строящий по любому доказательству формулы φ другой алгоритм, являющийся реализацией этой теоремы. Другими словами, здесь имеют место два результата, являющиеся непосредственным обобщением теорем Клини — Нельсона [33, 34] и Минца [35] на случай «нумерованных реализаций», устанавливающих существование реализации доказуемой формулы и эквивалентность получаемых реализаций для разных понятий реализуемости.

Сами по себе эти принципиальные результаты, которые обобщены также для анализа и для теории множеств [36, 37], недостаточно конструктивны для практического синтеза программ. Во-первых, уже для арифметики получается слишком высокая оценка сложности извлекаемого алгоритма (ε_0-рекурсивные функции). Во-вторых, в традиционных формализациях структура доказательства дает лишь отдаленное представление о действии извлекаемого алгоритма. Однако даже такая постановка задачи была принята некоторыми исследователями [9—11] как базис конкретных разработок. Но, как показывает анализ указанных работ, на этом пути возникли существенные трудности. Традиционное понятие конструктивности оказалось слишком общим и слишком узким для задачи синтеза программ. Узость объясняется сильной ориентацией на теорию чисел и привязкой к нумерациям объектов, т. е. к конкретному представлению данных. Чаще всего нам важны лишь некоторые свойства действий над данными. Кроме того, при таком подходе нельзя учесть ограничения на ресурсы, предписания и такие распространенные ситуации, как наличие операций, изменяющих «мир».

Учитывая эти обстоятельства, укажем более адекватную точку зрения на исходные понятия — языка, интерпретации, логики и т. д. Так, для нас недостаточно понимать логику как исчисление. Формальные конструкции имеют ценность лишь постольку, поскольку они имеют интерпретацию. Таким образом, на первом плане должны быть семантические вопросы. В то же время легче поддаются полному, конечному и эффективному решению как человеком, так и машиной синтаксические проблемы, а значит, естественно решать семантические проблемы, предварительно сформулировав их как синтаксические. Поэтому в дальнейшем будем придерживаться следующего методологического принципа: *синтаксическая правильность текстов, записанных на формальном языке, должна убедительно гарантировать семантическую.*

Поскольку здесь речь идет о некотором подходе к анализу проблемы синтеза программ, приводимые ниже определения схематичны, поэтому содержат такие общие понятия, как, например, «простыми средствами» и т. п., которые обретают точный смысл при конкретизации ситуации.[1)]

Определение 1. *Язык* — это разрешимое простыми средствами множество конечных слов некоторого алфавита, называемых *формулами*, образующихся применением конечного числа неукорачивающих *правил порождения.*

Последнее означает, что любая достаточно сложная формула может быть просто разбита на составные части.

Определение 2. *Интерпретированный язык* — это язык вместе с заданным классом возможных интерпретаций, где каждая интерпретация предполагает множество степеней достоверности. Определено отношение «формула A со степенью достоверности α подтверждается в интерпретации e» такое, что степень достоверности A доста-

[1)] Так, например, под «простыми средствами» в различных ситуациях можно понимать алгоритмы полиномиальной или экспоненциальной сложности или, скажем, примитивно рекурсивные или общерекурсивные и т. п.

точно просто и однозначно определяется степенями достоверности ее компонент.

Например, для обычной классической логики множество степеней достоверности для каждой интерпретации есть {0, 1}, в то время как для многих программных логик это множество состоит из множеств путей в программе [38].

Определение 3. *Логика* — это пара ⟨интерпретированный язык, исчисление⟩ такая, что отношение P — вывод формулы A из $A_1, \ldots, A_n$ по правилам логики» разрешимо простыми средствами и в любой интерпретации степень достоверности заключения правильно построенного вывода ненамного меньше степени достоверности посылок.

В классической логике истинность формул при выводе сохраняется, а в программной — область истинности заключения есть пересечение областей истинности посылок.

Однако не все логики нас устраивают в качестве инструмента. Прежде всего потребуем, чтобы каждый шаг рассуждения имел естественную интерпретацию, так как в противном случае структура доказательства не дает нам полного представления о структуре строящегося объекта. Далее, подтверждение заключения должно равномерно зависеть от подтверждений посылок, т. е. должно существовать эффективное правило сборки реализации заключения из реализаций посылок. Заметим, что требование полноты относительно выбранной семантики мы на логику не накладываем. Полнота — это приятная дополнительная особенность.

Определение 4. *Теория* есть тройка ⟨алфавит, логика, множество аксиом⟩.

Определение 5. *Текст проеграммы* — это конечная комбинация заданных исходных предписаний при помощи фиксированного конечного множества правил их соединения, где и предписания, и правила их комбинирования есть конструкции некоторого фиксированного языка.

Конкретизируя «кто» и «как» должен выполнять указанные предписания, мы приходим к следующим понятиям.

Определение 6. *Алгоритмический язык* — это разрешимое простыми средствами множество текстов программ вместе с категорией интерпретаций, в которой выделены класс допустимых интерпретаций исходных предписаний и совокупность функторов, одно-однозначно сопоставленных правилам сочленения предписаний.

Согласно этому определению действие (семантика) программы понимается как морфизм, составленный регулярным и равномерным образом из морфизмов, реализующих исходные предписания, из которых состоит текст программы.

Определение 7. Логика (теория) L называется *конструктивной относительно алгоритмического языка* ALG, если выполнены следующие требования.

а) Каждой логической связке сопоставлен фактор, преобразующий объекты, являющиеся интерпретациями аргументов этой связки. В этом случае будем говорить, что задана иерархия типов данных и каждой формуле A соответствует тип данных $\underline{a}$ (формула A типа $\underline{a}$), являющийся объектом категории.

Предположим, что имеется «пустой» тип данных *empty*.

б) Каждой формуле типа *empty* (*дескриптивной формуле*) сопоставлено условие на объекты категории интерпретаций, степень выполнения которого считается степенью ее достоверности.

в) Каждой формуле типа, отличного от empty (*конструктивной формуле*), сопоставлено условие на объектах данного типа; объект, удовлетворяющий данному условию в той или иной степени, считается реализацией формулы той же степени достоверности.

г) Каждому доказательству конструктивной формулы эффективно сопоставляется текст программы языка ALG, действие которой представляет реализацию указанной формулы и конструирование данной реализации базируется на реализациях конструктивных посылок (аксиом), используемых в доказательстве.

Отметим, что в этом определении все преобразования естественны в том смысле, что определяются индукцией по построению объектов и сводятся к сопоставлению правилу построения функтора комбинирования. Все упоминаемые в определении условия могут быть расплывчаты в смысле [39] — это наиболее типичный случай.

Пример. Рассмотрим обычную гильбертовскую формулировку позитивного импликативного исчисления [40]:

Схемы аксиом:

$$(A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C));$$

$$(A \Rightarrow (B \Rightarrow A)).$$

Правило вывода:

$$\frac{A,\ A \Rightarrow B}{B}.$$

Каждой формуле сопоставим тип ob (обы в смысле [41]). Программы будут комбинациями переменных и исходных обов K, S по правилам комбинаторной логики. Правило вывода соответствует правилу конверсии, схемы аксиом — исходным обам:

$$Sabc = (ac)(bc),$$

$$Kab = a.$$

Рассмотрим вывод $A \Rightarrow C$ из $A \Rightarrow B$ и $B \Rightarrow C$ и построим соответствующий ему комбинатор, осуществляющий композицию двух функций:

$$\underline{(((A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))) \Rightarrow ((A \Rightarrow (B \Rightarrow C) \Rightarrow (A \Rightarrow B)) \Rightarrow (A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow C)))} \quad \# S \#$$

$$\downarrow$$

$$\underline{(A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))} \quad \# S \#$$

$$\downarrow$$

$$\underline{((A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow B)) \Rightarrow ((A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow C))} \quad \# SS \#$$

$$\downarrow$$

$$\underline{(A \Rightarrow B) \Rightarrow ((A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow B))} \quad \# K \#$$

$$\downarrow \qquad \underline{(A \Rightarrow B)} \quad \# y \#$$

$$\downarrow$$

$$\underline{(A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow B)} \quad \# Ky \#$$

$$\downarrow$$

$$\underline{(A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow C)} \quad \# SS(Ky) \#$$

$$\underline{(B \Rightarrow C) \Rightarrow (A \Rightarrow (B \Rightarrow C))} \quad \# K \# \qquad \underline{(B \Rightarrow C)} \quad \# x \#$$

$$\downarrow \qquad \downarrow$$

$$\underline{(A \Rightarrow (B \Rightarrow C))} \quad \# Kx \#$$

$$\underline{(A \Rightarrow C)} \quad \# SS(Ky)(Kx) \#.$$

Комбинатор $SS(Ky)(Kx)$, где x — реализация $(B \Rightarrow C)$, y — реализация $(A \Rightarrow B)$, является искомым. Действительно, применим его к реализации A:

$$(SS(Ky)(Kx))a = ((S(Kx))(Ky)(Kx))a = S(Kx)(Ky(Kx))a =$$

$$= S(Kx)ya = ((Kx)a)(ya) = x(y(a)),$$

т. е. получили реализацию C, что и требовалось доказать.

Отметим, что в примере дескриптивных формул не было. Все формулы считались конструктивными и все наши доказательства транслировались непосредственно в текст программы на соответствующем алгоритмическом языке.

§ 3. ПОСТРОЕНИЯ, СВЯЗИ И ПРИЗНАКИ

Рассмотрим в качестве содержательной иллюстрации вышесформулированных определений так называемый логический подход к программированию [42—46].

Цель излагаемых ниже теоретических построений — создание инструмента логического конструирования вычислительных программ из готовых подпрограмм. Другими словами, требуется создание теории пакетов программ для вычислительных задач. При этом предполагается, что

— спецификации задаются в описательной форме;

— нас интересуют лишь достаточно простые программы, являющиеся в некотором смысле простыми комбинациями имеющихся подпрограмм;

— знание об имеющихся программах допускает точную формализацию на языке логики предикатов;

— синтезируемые программы могут содержать циклы либо на самом верхнем уровне, либо внутри стандартных подпрограмм (это предположение весьма радикальное, но сложившаяся практика конструирования пакетов программ ему не противоречит);

— ограничения на объем требуемой памяти и время работы программы не являются существенными [1].

Таким образом, центральная наша задача — синтез достаточно хороших программ без циклов над заданной совокупностью абстрактных типов данных, стандартных подпрограмм и знаний о них (вычислительной обстановкой). Отметим, что чисто алгебраический подход к задаче синтеза явно игнорирует последнюю составляющую — знание. Однако необходимо учитывать следующее: ни одним свойством программы мы не сможем воспользоваться, если его не знаем, и в то же время объем знаний о стандартных подпрограммах обычно крайне ограничен. Последнее обстоятельство играет положительную роль. Это фактически один из аспектов концепции модульного программирования — ограничивать информацию о модуле, которая доступна снаружи.

В данном параграфе приводится доказательство одного результата, очерчивающего важный класс разрешимых конструктивных теорий. По пути формулируются многие понятия, являющиеся центральными в индуцируемой логическим подходом новой технике формализации.

Для достижения конструктивности логики необходимо обеспечить естественное соответствие между правилами вывода и операторами алгоритмического языка. С этой целью формализация естественного вывода [47] была несколько перестроена [43]. В результате получившаяся формализация оказалась гораздо ближе к обычным человеческим рассуждениям. Кроме того, она оказалась лучше и с точки зрения машинного поиска доказательства.

Система состоит из трех конструктивных правил вывода.

Правило применения процедуры. Как уже было сказано, функциональная формула

$$\forall \bar{x}((\mathfrak{A}_1 \& \ldots \& \mathfrak{A}_n) \Rightarrow \exists \bar{y}(\mathfrak{B}_1 \& \ldots \& \mathfrak{B}_k))$$

[1] Это соглашение будет значительно ослаблено в дальнейших работах.

выражает описание процедуры. Для получения нужных результатов на вход процедуры необходимо подать конкретные значения аргументов и подтверждение их правильности, т. е. возможности применимости процедуры. Отсюда приходим к следующему правилу вывода:

$$\forall x_1 \ldots x_m(\mathfrak{A}_1 \,\&\, \ldots \,\&\, \mathfrak{A}_n \Rightarrow \exists y_1 \ldots y_l(\mathfrak{B}_1 \,\&\, \ldots \,\&\, \mathfrak{B}_k))$$

$$\left.\begin{array}{l}\mathfrak{A}_1(t_1, \ldots, t_m)\\ \vdots\\ \mathfrak{A}_n(t_1, \ldots, t_m)\end{array}\right\}\ t_1, \ldots, t_m \text{ — конкретные значения аргументов;}$$

$\mathfrak{A}_i$ — условия корректности значений и аргументы высших типов

$$\left.\begin{array}{l}\mathfrak{B}_1(t_1, \ldots, t_m, c_1, \ldots, c_l)\\ \vdots\\ \mathfrak{B}_k(t_1, \ldots, t_m, c_1, \ldots, c_l)\end{array}\right\}\ c_i \text{ — обозначения полученных результатов;}$$

$\mathfrak{B}_i$ — условия связи «вход-выход».

Здесь c_i — новые вспомогательные константы, обозначающие промежуточные результаты. В процессе вывода нам абсолютно безразлично, с помощью какой функции они получены, важно лишь наличие требуемых условий. Поэтому и выбран нарочито безличный способ отметки построенных значений.

Правило описания процедуры. Данное правило вводит предположение, а затем его устраняет. Чтобы доказать функциональную формулу, необходимо указать способ построения $\bar{y}$ по $\bar{x}$, используя имеющиеся у нас средства, и его обоснование. Необходимо вывести $\mathfrak{B}$ из $\mathfrak{A}$ при помощи итерационного процесса применения правил вывода (в том числе, возможно, и правил описания процедуры). Это заставляет нас ввести понятие *вспомогательного вывода*, начинающегося с объявления временно истинных некоторых фактов и заканчивающегося результатами:

$]\mathfrak{A}_1, \ldots, \mathfrak{A}_n; x_1, \ldots, x_m$ — произвольны} «допустим, что $\mathfrak{A}_i$ истинны и ...»

$$\left.\begin{array}{l}\vdots\\ \mathfrak{B}_1(t_1, \ldots, t_l)\\ \vdots\\ \mathfrak{B}_k(t_1, \ldots, t_l)\end{array}\right\}\ \text{«По правилам вывода получаем ...»}$$

$$\forall x_1, \ldots, x_m(\mathfrak{A}_1 \,\&\, \ldots \,\&\, \mathfrak{A}_n \Rightarrow \exists y_1 \ldots y_l(\mathfrak{B}_1(y_1, \ldots, y_l) \,\&\, \ldots \,\&\, \mathfrak{B}_k(y_1 \ldots, y_l)))$$

Заметим, что интерпретация правил вывода, применяемых к формулам, в которых $\mathfrak{A}_1, \ldots, \mathfrak{A}_n, \mathfrak{B}_1, \ldots, \mathfrak{B}_k$ элементарны, требует лишь наличия функций объектов и операции композиции. Если же применять эти правила к формулам произвольной логической сложности, то необходимо уметь строить иерархии функционалов конечных типов [48—50]. В дальнейшем по мере того, как будут сниматься ограничения на вид синтезируемых программ, понадобятся пространства функционалов более сложной природы. В частности, кажется правдоподобным, что в случае снятия ограничения на циклы адекватными моделями могут оказаться так называемые *аппликативные А-пространства* [51, 52].

Правило разбора случаев. Для того чтобы воспользоваться либо имеющимися с самого начала, либо уже доказанными классификациями, мы должны разобрать по отдельности каждый из получившихся случаев. Соответственно, чтобы доказать классификацию, необходимо для каждого из появляющихся случаев подобрать такие условия его истинности, которые в сумме покрывали бы все возможности, т. е. опять-таки подобрать классификацию. В этом явное преимущество интуиционистской логики, поскольку в ней отсутствуют формулы вида $\mathfrak{A} \vee \neg\mathfrak{A}$. Отсю-

да приходим к следующему правилу вывода, двойственному самому себе:

$$\frac{\mathfrak{A}_1 \vee \ldots \vee \mathfrak{A}_n\} \text{ исходная классификация} \quad \left|\begin{array}{c}\mathfrak{A}_1 \\ \vdots \\ \mathfrak{B}_{i_1}(t_1^1, \ldots, t_k^1)\end{array}\right| \quad \ldots \quad \left|\begin{array}{c}\mathfrak{A}_n \\ \vdots \\ \mathfrak{B}_{i_m}(t_1^m, \ldots, t_k^m)\end{array}\right|}{\mathfrak{B}_1(c_1, \ldots, c_k) \vee \ldots \vee \mathfrak{B}_l(c_1, \ldots, c_k).}$$

Здесь каждое $\mathfrak{B}_{i_j}$ есть либо один из членов выводимой дизъюнкции, либо ложь. Отметим, что некоторые $\mathfrak{B}_i$ могут получаться по нескольку раз, другие ни разу.

Данное правило требует для своей реализации булевых значений, если $\mathfrak{A}_i$ элементарны, и конструкций типа прямой суммы в общем случае [39].

Подробнее все эти правила вывода вместе с примерами доказательства рассмотрены в [43].

Запись $\mathfrak{B}(\bar{x})$ означает, что в любой предикат из $\mathfrak{B}$ входят все переменные из вектора $\bar{x}$, $\mathfrak{B}(\tilde{\bar{x}})$ — что в любой предикат из $\mathfrak{B}$ входит хотя бы одна переменная из вектора $\bar{x}$. Пусть $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$ (возможно, с индексами) — конъюнкции элементарных формул; A, B, C, ... — исходные предикаты. Выделим следующие классы формул:

а) *факт* — формула вида $\mathfrak{A}(\bar{c})$ или $\mathfrak{A}(\bar{c})$, где $\bar{c}$ — вектор констант;

б) *связь* — формула вида $\forall \bar{x}(\mathfrak{A}(\bar{\bar{x}}) \Rightarrow \mathfrak{B}(\bar{\bar{x}}))$;

в) *классификация* — формула вида

$$\forall \bar{x}(\mathfrak{A}(\bar{x}) \Rightarrow \mathfrak{B}_1(\bar{\bar{x}}) \vee \ldots \vee \mathfrak{B}_n(\bar{\bar{x}}));$$

г) *опровержение* — формула вида $\forall \bar{x}(\mathfrak{A}(\bar{x}) \Rightarrow \neg \mathfrak{B}(\bar{\bar{x}}))$;

д) *построение* — формула вида $\forall \bar{x}(\mathfrak{A}(\bar{x}) \,\&\, \mathscr{Y} \Rightarrow \exists \bar{y} \mathfrak{B}(\bar{x}, \tilde{\bar{y}})$, где $\mathscr{Y}$ — конъюнкция (возможно, пустая) построений (здесь $\bar{y}$ — результаты построения, $\bar{x}$ — аргументы, $\mathscr{Y}$ — процедуры-параметры);

е) *гиперпостроение* — формула вида

$$\forall \bar{x}(\mathfrak{A}(x) \,\&\, \mathscr{Y} \Rightarrow [(\exists \bar{y}_1(\mathfrak{B}_1(\bar{x}, \tilde{\bar{y}}_1) \,\&\, \mathfrak{C}_1(\tilde{\bar{y}}_1)) \vee \ldots$$

$$\ldots \vee (\exists \bar{y}_n(\mathfrak{B}_n(\bar{x}, \tilde{\bar{y}}_n) \,\&\, \mathfrak{C}_n(\tilde{\bar{y}}_n)))],$$

где $\mathscr{Y}$ и $\mathfrak{C}_i$ — конъюнкции (возможно, простые) гиперпостроений;

ж) *признак 1-го рода* — формула вида

$$\forall \bar{x}((\mathfrak{A}(\bar{x}) \Rightarrow \mathfrak{B}(\bar{x})) \Rightarrow \mathfrak{C}(\bar{\bar{x}}));$$

з) *признак 2-го рода* — формула вида

$$\forall \bar{x}(\forall \bar{y}(\mathfrak{A}(\bar{x}, \bar{y}) \Rightarrow \mathfrak{B}(\bar{\bar{x}}));$$

и) *признак 3-го рода* — формула вида $\forall \bar{x}, \bar{y}(\mathfrak{A}(\bar{x}, \bar{y}) \Rightarrow \mathfrak{B}(\bar{\bar{x}}))$ (здесь $\bar{y}$ — непустой вектор).

Эти классы формул естественно появляются при анализе процесса извлечения программы из доказательства. Будем называть перечисленные выше формулы *стандартными*. Назовем сумму логических длин, входящих в некоторый вывод формул, *длиной* этого вывода.

Теорема 1. *По любой интуиционистской теории T эффективно можно построить такое ее консервативное расширение T', что*

i) все аксиомы T' стандартны;

ii) если дан вывод $\mathfrak{R}$ в T, то он с увеличением длины не более чем в три раза и сохранением извлекаемой программы эффективно может быть перестроен в вывод $\mathfrak{R}$ в T';

iii) если $\mathfrak{A}$ — формула сигнатуры теории T' и дан вывод $\mathfrak{A}$ в T', то с увеличением длины не более чем в три раза этот вывод можно перестроить в вывод формулы $\mathfrak{A}$ в теории T с сохранением извлекаемой программы.

Теорема доказывается последовательной заменой сложных подформул в аксиомах на новые предикаты и добавлением аксиом — определений для них. Она демонстрирует реальную возможность ограничиться при формализации только аксиомами указанных выше видов — это не испортит ни доказательства, ни извлекаемые из них программы.

Таким образом, задача построения программы из подпрограмм формализуется как задача поиска вывода в интуиционистской логике формулы — построения из аксиом некоторой элементарной теории, описывающей наши исходные функции, предикаты и знания о них.

Заметим, что при всей рутинности своего доказательства эта теорема заставляет обратить внимание на одно интересное обстоятельство. Иногда можно вводить построения, которым не соответствуют реализованные подпрограммы. Но это возможно лишь в том случае, когда эти построения являются «тупиковыми»: они не должны быть существенно использованы при доказательстве реально требующихся программ. Другими словами, предикаты исходного языка можно делить не только на вычислимые и теоретические, но и на внешние предоставляемые пользователю и внутренние (вводимые для наших собственных нужд).

Для задачи синтеза программ особый интерес представляют те теории, в которых проблема выводимости построений разрешима. Примером может служить так называемая беспризнаковая теория.

Определим количество входных переменных в построении $\forall \bar{x}(\mathfrak{A}(\bar{x}) \,\&\, \&\, \mathscr{V} \Rightarrow \exists \bar{y}(\mathfrak{B}(\bar{x}, \bar{y})))$ следующим образом: «количество элементов в наборе $\bar{x}$ + сумма числа элементов в наборах — результатах построений из $\mathscr{V}$». Построение назовем *несокращающимся*, если в любом предикате из $\mathfrak{B}$ количество различных переменных не меньше, чем количество входных переменных построения, и все посылки — построения его посылок — построений также несокращающиеся.

Определение 8. Стандартизованная теория T *беспризнаковая*, если она не содержит гиперпостроений, не являющихся построениями, и признаков и, кроме того, все ее построения несокращающиеся.

Теорема 2. *Проблема выводимости для построений, в которых все члены из $\mathscr{V}$ несокращающиеся, разрешима в любой конечной беспризнаковой теории.*

Замечание. Отметим, что под сокращающее построение можно легко замаскировать признак 3-го рода. А именно, если дана аксиома — признак $\forall(\bar{x}, \bar{y}(\mathfrak{A}(\bar{x}, \bar{y}) \Rightarrow \mathfrak{B}(\bar{x}))$, то достаточно заменить в $\mathfrak{B}$ все предикаты $P(\bar{x})$ на $P'(\bar{x}, a)$ и аксиому — на формулу $\forall \bar{x}, \bar{y}(\mathfrak{A}(\bar{x}, \bar{y}) \Rightarrow \exists a \mathfrak{B}(\bar{x}, a))$. Поэтому теория без признаков, но содержащая сокращающие построения, может оказаться неразрешимой даже относительно теорем вида $\exists \bar{x} \mathfrak{B}(\bar{x})$. В частности, такой пример легко построить в классе теорий полугрупп с неразрешимой проблемой тождества.

Доказательство. Будем проверять выводимость в теории, сводя ее к проверке выводимости конъюнкций «обобщенных фактов», включающих метапеременные, из аксиом теории и допущений. При этом константы, аксиомы, допущения, метапеременные и цели классифицируются по уровням. Уровни будут образовывать дерево, и при доказательстве факта уровня σ или при унификации переменной уровня σ можно использовать лишь объекты уровней $\leqslant \sigma$. Несокращающий ха-

рактер построений, используемых в теории, приведет к тому, что на каждом шаге поиска вывода количество метапеременных может быть меньше либо оставаться прежним, что, в свою очередь, дает возможность проверять только конечное число существенно различных выводимостей.

Чтобы реализовать эту схему, докажем ряд лемм.

Лемма 1. *Если в нормализованном выводе построения в теории без гиперпостроений и признаков 2-го рода встретилось правило — описание процедуры, то его заключение — это либо доказываемое построение, либо малая посылка правила применения процедуры с некоторым построением, т. е. процедурный параметр доказываемого построения.*

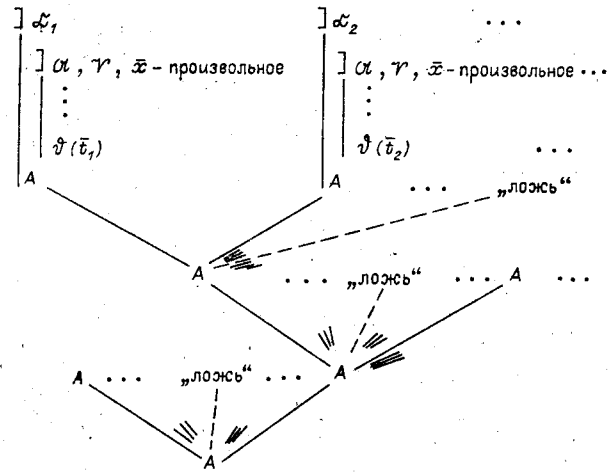
Доказательство. В нормализованном выводе заключение правила — описания процедуры не может выступать одновременно как главная посылка правила применения процедуры. Следовательно, возможны только случаи, указанные в лемме.

Лемма 2. *Нормализованные доказательства построений в теории без гиперпостроений можно модифицировать без увеличения длины таким образом, чтобы все доказуемые построения были заключениями правил описания процедуры.*

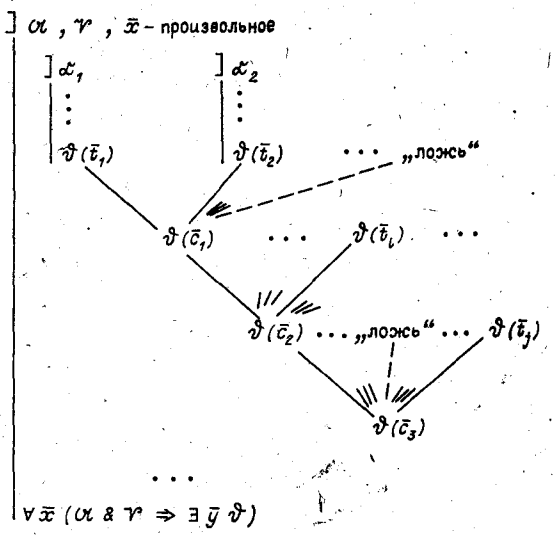
Доказательство. Построение может быть доказано, по свойству подформульности, лишь при помощи правила разбора случаев или правила описания процедуры. Правило разбора случаев в этом случае (опять-таки по свойству подформульности и нормализованности) имеет вид:

$$\frac{\begin{array}{l}\mathfrak{B}_1(t) \vee \ldots \vee \mathfrak{B}_k(\bar{t}) \text{ — заключение некоторой классификации} \\ ]\mathfrak{B}_1(\bar{t}) \quad \ldots \qquad\qquad]\mathfrak{B}_k(\bar{t}) \\ \vdots \qquad\qquad\qquad\qquad \vdots \\ |\forall \bar{x}(\mathfrak{A}(\bar{x}) \& \mathfrak{D} \Rightarrow \exists \bar{y}\mathfrak{D}(\bar{x}, \bar{y})) \text{ или «ложь»} \quad |\forall \bar{x}(\mathfrak{A}(\bar{x}) \& \mathfrak{D} \Rightarrow \exists \bar{y}\mathfrak{D}(\bar{x}, \bar{y})) \text{ или «ложь»}\end{array}}{A \rightleftharpoons \forall \bar{x}(\mathfrak{A}(\bar{x}) \& \mathfrak{D} \Rightarrow \exists \bar{y}\mathfrak{D}(\bar{x}, \bar{y}))}.$$

Отсюда можно построить дерево вывода для каждого из предков построения, имеющих тот же вид, что и доказуемое построение, такое, что заключительные вершины этого дерева соответствуют правилам описания процедуры, а остальные — правилам разбора случаев. Например,



Здесь пунктиром помечены случаи, исключенные ввиду того, что в результате были получены противоречия. Перестроим теперь этот участок вывода следующим образом:



Здесь $\bar{c}_1$, $\bar{c}_2$, $\bar{c}_3$, ...— наборы вспомогательных констант. Длина вывода при такой перестройке не меняется.

В дальнейшем, говоря «применение построения, связи, признака, опровержения», мы будем иметь в виду их использование как главной посылки правила применения процедуры, а «применение классификации» — как главной посылки правила разбора случаев (возможно, после выделения заключения, используя правило применения процедуры).

Лемма 3. *Нормализованное доказательство факта, связи, признака 1-го или 2-го рода в теории без гиперпостроений, признаков 3-го рода и опровержений не может содержать применений построений.*

Доказательство. После применения построения в любом его заключении имеется хотя бы одна вспомогательная константа. Применение связей, признаков 1-го и 2-го рода, классификаций к посылкам, не содержащим терм t, обязательно содержит терм t. Правило описания процедуры, примененное как последнее правило в выводе нужной теоремы, также не избавляет нас от вспомогательных констант.

Следующая лемма является непосредственным следствием только что доказанной.

Лемма 4. *Факт в теории без опровержений, гиперпостроений и признаков 3-го рода имеет нормализованное доказательство, состоящее лишь из фактов и их дизъюнкций.*

Следствием лемм 3 и 4 является

Лемма 5. *Если в беспризнаковой теории в нормализованном доказательстве факта встретилось применение построения, то это произошло в отбрасываемой из-за противоречия альтернативе разбора случаев.*

Лемма 6. *Проблема выводимости для фактов в теории без опровержений, гиперпостроений и признаков разрешима.*

Доказательство. Достаточно заменить все переменные в связях и классификациях всеми возможными комбинациями констант и рассмотреть проблему выводимости в конечной пропозициональной теории, полученной добавлением для соответствующих подстановок аксиом — фактов.

Рассмотрим теперь проблему выводимости для построений в теории, состоящей лишь из построений и фактов (ПФ-теория).

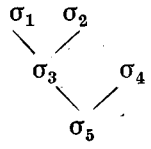
Лемма 7. *В ПФ-теории факт выводим в том и только в том случае, когда он является аксиомой.*

Следствие леммы 4.

Введем в рассмотрение так называемые m-секвенции, конструкция которых аналогична секвенциям, введенным в работе [53]. Пусть задано дерево уровней T. Его элементы будем обозначать σ, σ_1, σ_2, ..., τ, τ_1, ... Для каждого уровня σ через $*_1^{\sigma}, \ldots, *_n^{\sigma}, \ldots$ будем обозначать метапеременные соответствующего уровня.

Определение 9. Совокупность двух конечных списков: списка посылок $\langle \sigma_1 A_1, \sigma_2 A_2, \ldots \rangle$ и списка результатов $\langle \tau_1 B_1, \tau_2 B_2, \ldots \rangle$, где $A_1, \ldots, B_1, \ldots$ — формулы, снабженные индексом уровня, причем входящие в них константы и метапеременные также снабжены уровнями и уровни констант должны быть меньше уровня формул, куда они входят, а уровни метапеременных — меньше уровня формул из списка результатов и строго меньше формул из списка посылок, называется *m-секвенцией.*

Пример. *Пусть дано дерево уровней:*



Тогда $\langle \sigma_1 P\left(*_1^{\sigma_5}, *_2^{\sigma_3}, c^{\sigma_5}\right), \sigma_2 \forall x, y\left(Q\left(x, *_3^{\sigma_3}, c_1^{\sigma_2}\right) \Rightarrow \exists z H\left(c_2^{\sigma_3}, x, y, z\right)\right)\rangle$, $\langle \sigma_4 R\left(*_4^{\sigma_5}, c^{\sigma_5}\right), \sigma_2 H\left(c_2^{\sigma_3}, *_5^{\sigma_2}, *_4^{\sigma_2}, *_6^{\sigma_2}\right)\rangle$ — m-секвенция.

Если в результатах m-секвенции есть построение

$$\sigma \forall \bar{x}(A_1 \& \ldots \& A_n \& \mathscr{V}_1 \& \ldots \& \mathscr{V}_k \Rightarrow \exists \bar{y}(B_1 \& \ldots \& B_l)), \qquad (*)$$

то можно

а) ввести новый уровень σ_1, непосредственно следующий за σ;

б) добавить в список посылок $\sigma_1 A_1, \ldots, \sigma_1 A_n, \sigma_1 \mathscr{V}_1, \ldots, \sigma_1 \mathscr{V}_k$, а в список результатов $\sigma_1 B_1\left(\bar{*}^{\sigma_1}\right), \ldots, \sigma_1 B_l\left(\bar{*}^{\sigma_1}\right)$, где $\bar{*}^{\sigma_1}$ — набор новых метапеременных, заменяющий $\bar{y}$;

в) пометить элементы набора $\bar{x}$ уровнем σ_1 и объявить их константами этого уровня;

г) изъять формулу $(*)$ из списка результатов.

Такое преобразование m-секвенции назовем *подъемом на уровень.* Другое преобразование, называемое *унификацией*, определяется следующим образом:

если в посылках m-секвенции есть построение $(*)$, то

а) выбираем среди заключений факты $\sigma_1 B'_{i_1}, \ldots, \sigma_k B'_{i_k}$ с предикатами, входящими в заключение построения $(*)$, где $\sigma_1, \ldots, \sigma_k \geqslant \sigma$;

б) находим такие подстановки констант и метапеременных вместо $\bar{x}$ и различных метапеременных для $\bar{y}$, отличных от используемых для постановки вместо $\bar{x}$, что после их осуществления B_{i_1} совпадает с $B'_{i_1}, \ldots, B_{i_k}$ — с B'_{i_k};

в) проверяем, чтобы метапеременные, использованные в $B'_{i_1}, \ldots, B'_{i_k}$ вместо $\bar{y}$, не входили ни в какие другие формулы заключения;

г) вычеркиваем из списка результатов $B'_{i_1}, \ldots, B'_{i_k}$, а из списка посылок все формулы, включающие метапеременные, использованные для постановки вместо $\bar{y}$;

д) добавляем в список результатов формулы $\tau_1 A_1, \ldots, \tau_1 A_n, \tau_1 \mathscr{V}_1, \ldots, \tau_1 \mathscr{V}_k$, где τ_1 — максимум из σ и уровней объектов, использованных для подстановки вместо $\bar{x}$.

Кроме этих двух операций над m-секвенциями определим еще одну — *подстановку*:

если в посылках m-секвенции есть факты (возможно, и с метапеременными), то можно проделать следующие преобразования:

а) найти для факта σA из списка посылок в списке результатов факты $\sigma_1 A_1, \ldots, \sigma_n A_n$ с тем же предикатом;

б) найти такую подстановку констант или метапеременных вместо метапеременных из $A, A_1, \ldots, A_n$, чтобы уровень подставляемого был не больше уровня метапеременных и чтобы A совпала с $A_1, \ldots, A_n$.

в) вычеркнуть $A_1, \ldots, A_n$ из списка результатов, а найденную подстановку осуществить повсюду в m-секвенции.

Лемма 8. *Построение $\mathfrak{A}$ выводимо в ПФ-теории T тогда и только тогда, когда m-секвенция $(\langle T^0\rangle, \langle\mathfrak{A}^0\rangle)$, где T^0, $\mathfrak{A}^0$ — результаты пометки всех констант и формул уровнем 0, сводима конечным числом применений операций подъема на уровень, унификации и подстановки к m-секвенции вида $(\langle T'\rangle, \varnothing)$.*

Доказательство данной леммы состоит в рутинной проверке согласованности правил преобразования m-секвенций с правилами вывода.

Лемма 9. *Если T — беспризнаковая ПФ-теория и $\mathfrak{A}$ — построение вида $(*)$ с несокращающимися посылками, то в любой m-секвенции, полученной преобразованиями m-секвенции $S_1 = (\langle T^0, 1A_1, \ldots, 1A_n, 1\mathscr{V}_1, \ldots, 1\mathscr{V}_k\rangle, \langle 1B_1(\overline{*^1}), \ldots, 1B_l(\overline{*^1})\rangle)$, которая в свою очередь получена из m-секвенции $S_0 = (\langle T^0\rangle, \langle\mathfrak{A}_0\rangle)$ применением операции подъема на уровень (здесь 1 — уровень, непосредственно следующий за 0), содержится не больше метапеременных, чем в S_1.*

Доказательство. При унификации количество метапеременных может только понизиться из-за того, что построения в посылках S_1 — несокращающиеся и выделяются из них при унификации в посылки опять-таки несокращающиеся построения. При операции подъема на уровень число метапеременных может вновь увеличиваться по сравнению с предыдущей m-секвенцией, но не более чем до количества, которое предшествовало унификации, выделявшей построение — результат. Правило подстановки новых метапеременных не производит.

Следующая лемма очевидна.

Лемма 10. *Если в m-секвенции S посылки уровня σ имеют вид $A_1, \ldots, A_n$, а посылки уровня $\sigma' > \sigma$ — $A_1', \ldots, A_n'$ отличаются от $A_1, \ldots, A_n$ лишь взаимно однозначной заменой констант уровня σ на константы уровня σ' и других констант уровней σ, σ' в S нет, то без изменения выводимости можно из S удалить $A_1', \ldots, A_n'$, а в списке результатов заменить константы уровня σ' на соответствующие константы уровня σ.*

Итак, что же можно сказать об операциях унификации и подъема на уровень? При унификации не увеличивается ни число метапеременных, ни количество констант. При подъеме на уровень количество метапеременных не превышает определенной границы (устанавливаемой по той предшествующей m-секвенции без построений в списке результатов, из которой получилась данная), но число констант может увеличиваться. Единственный случай, когда число констант увеличивается одновременно с сохранением того же числа метапеременных — это применение построения вида

$$\forall\bar{x}_1(\mathfrak{A}_1 \Rightarrow \exists\bar{y}_1\mathfrak{B}_1)\ \&\ \ldots\ \&\ \forall\bar{x}_k(\mathfrak{A}_k \Rightarrow \exists\bar{y}_k\mathfrak{B}_k) \Rightarrow \exists\bar{y}\mathfrak{C}).$$

Однако при повторном применении такого построения (это может оказаться необходимым) новые константы уже могут быть устранены как избыточные (лемма 10). Таким образом, верна

Лемма 11. *Для любой m-секвенции S, все посылки которой — факты или несокращающиеся построения, а результаты — только факты,*

число применений операций, при которых количество метапеременных не изменяется, равномерно ограничено.

Из этой леммы легко следует справедливость утверждения теоремы для ПФ-теорий. Добавление связей к ПФ-теории фактически не меняет доказательства теоремы, поскольку операция унификации позволяет свести его к предыдущему случаю.

Появление классификаций делает необходимым добавить еще одну операцию — *унификацию классификации вида*

$$\sigma\forall x(A \Rightarrow B_1 \vee \ldots \vee B_k).$$

Операция заключается в следующем:

находим такие различные $\sigma_1 \geqslant \sigma$, $\tau_1, \ldots, \tau_k$, что $\tau_1, \ldots, \tau_k$ непосредственно следуют за σ_1 и что единственными результатами уровней $\tau_1, \ldots, \tau_k$ являются $\tau_1 B_1', \ldots, \tau_k B_k'$, унифицируемые с $B_1, \ldots, B_k$.

Уровень A после унификации будет σ_1. Доказательство конечности процесса сведения остается без существенного изменения (добавляются только шаги вида: раздробления σB на $\sigma_1 B, \ldots, \sigma_k B$, где $\sigma_1, \ldots, \sigma_k$ — новые уровни, непосредственно следующие за σ).

Добавление опровержений в совокупности с классификациями приводит к появлению опровергнутых вариантов, т. е. с точки зрения извлечения программы — бездействующих частей вывода. Остановить процесс разрастания количества метапеременных удается лишь при помощи приема, использованного в лемме 10. Если некоторое опровержение дважды применено на уровнях $\sigma < \sigma_1$, то его следы на уровне σ можно вычеркнуть. Итак, теорема доказана.[1)]

Заметим, что теорема может быть обобщена на случай теорем с признаками 1-го и 2-го рода, а также для теорий с гиперпостроениями, если соответствующим образом сформулировать понятие несокращаемости. Кроме того, в теореме можно заменить локальное условие несокращаемости, которое является очень сильным, на глобальное условие следующего вида.

Предикат P назовем *непосредственно зависящим от Q в теории T*, если P является членом заключения некоторой аксиомы, куда Q входит позитивно в посылку. Транзитивное замыкание этого отношения назовем *отношением зависимости* в теории T.

Требование. Для любого предиката P из заключения построения число переменных, входящих в него, должно быть не меньше суммы числа переменных, входящих в зависящие от него предикаты посылки.

Есть основания считать, что данное требование является необходимым и достаточным.

§ 4. ЗАКЛЮЧЕНИЕ

В данной работе мы не касались многих важных методологических и теоретических вопросов. Среди них можно отметить задачу соотношения проблемы синтеза с проблемами верификации, оптимизации и модификации программ. Заметим, что чисто алгоритмический подход к этим проблемам страдает определенными недостатками. Наиболее естественным, на наш взгляд, является сочетание алгоритмического и логического подходов. При этом возникает проблема освоения логическим подходом полезных приемов, разработанных, например, в трансформационном программировании. Интерес представляет и проблема установления правильности исходных подпрограмм. Здесь естественно возникает взаимо-

[1)] Формулировки теорем 1, 2 были приведены в [45].

связь логического и индуктивного подходов. Осталась в стороне и задача практической реализации поиска вывода и связанные с ней вопросы оптимизации извлекаемых программ, вложения в приведенной формализм понятия абстрактного типа данных и постановка проблемы синтеза соответствующих программ. И наконец, не обсуждалась проблема обобщения логического подхода на другие классы задач, отличных от вычислительных. Как эти, так и многие другие актуальные проблемы теории синтеза программ ждут своего решения.

Авторы признательны Ю. Л. Ершову за внимание и заинтересованность, проявленные к настоящей работе.

ЛИТЕРАТУРА

1. **Непейвода Н. Н., Свириденко Д. И.** Программирование с логической точки зрения. Новосибирск: Б. и., 1981.— (Препринт ИМ СО АН СССР; Т — 1).
2. **Непейвода Н. Н., Свириденко Д. И.** Логическая точка зрения на программирование. Новосибирск: Б. и., 1981.— (Препринт ИМ СО АН СССР; Т — 2).
3. **Ichbian J. D., Heliard J. C., Roubine O. e. a.** Rationale for design of the ADA programming language.— SIGPLAN Notices, 1979, v. 14, N 6, part A, B, p. 400.
4. **Тыугу Э. Х.** Синтез программ (обзор).— В кн.: Методы математической логики в проблемах искусственного интеллекта и систематическое программирование: Тезисы докладов Всесоюз. конференции. Ч. 2. Вильнюс, 1980, с. 70—89.
5. **Тамм Б. Г., Тыугу Э. Х.** Пакеты программ.— Изв. АН СССР. Сер. «Техническая кибернетика», 1977, № 5, с. 111—124.
6. **Manna Z., Waldinger R. J.** Towards automatic program synthesis.— Communications of the ACM, 1971, v. 14, N 3, p. 151—165.
7. **Constable R. J.** Constructive mathematics and automatic program writers.— In: Proceeding of IFIP congress, Ljubljana. Amsterdam, North-Holland, 1971, p. 229—233.
8. **Manna Z., Waldinger R. J.** Synthesis: Dreams → Programs.— IEEE Trans. on Software Eng., 1979, v. SE — 5, N 4, p. 294—328.
9. **Degli Antoni G., Miglioli P., Ornaghi M.** The synthesis of programs in an intuituinistic frame. Milan: S. n., 1974.
10. **Miglioli P., Ornaghi M.** A calculus to build up correct program.— In: Mathematical Foundations of Computer Science, 1977. Proceeding 6th symposium. Lect. Notes in Comp. Sci., v. 53, Berlin — N. Y.: Springer-Verlag, 1977, p. 398—409.
11. **Miglioli P., Ornaghi M.** A purely logical computing model: the open proofs as programs. Milan: S. n., 1979.
12. **Balzer R. M., Goldman N., Wile D.** Informality in program specifications.— In: Proceeding 5th International Joint Conference on Artifical Intelligence. Cambridge, MIA, 1977, p. 389—397.
13. **Barstow D.** A knowledge-based system for automatic program construction.— In. Proceeding 5th International Joint Conference on Artificial Intelligence. Cambridge, MIA, 1977, p. 382—388.
14. **Плакс Т. П.** Синтез параллельных программ на вычислительных моделях.— Программирование, 1977, № 4, с. 55—63.
15. **Buchanan J. R., Luckham D. C.** On automating the constracting of Programs.— Artificial Intelligence Lab., Stanford Univ., Stanford, CA, Tech. Rep., May, 1974.
16. **Manna Z., Waldinger R. J.** The synthesis of structure — changing programs.— In: Proceeding 3rd International Conference on Software Eng., May 10—12, 1978, Atlanta, Georgia, USA. IEEE Computer Society — ACM, p. 175—187.
17. **Manna J., Waldinger R. J.** Knowledge and reasoning in programs synthesis.— Artificial Intelligence J., 1975, v. 6, N 2, p. 175—208.
18. **Ulrich J. W., Moll R.** Program synthesis by analogy.— SIGPLAN Notices, 1977, v. 12, N 8, p. 22—28.
19. **Tyugu E. H.** A programming system with automatic program synthesis.— In: Methods of Algorithmic Language Implementation, Lect. Notes in Comp. Sci., v. 47, Berlin — N. Y.: Springer-Verlag, 1977, p. 251—267.
20. **Darlington J.** Application of program synthesis.— In: Proceeding IRIA symposium on Proving and Improving Programs. Are-et-Senans (Franc), S. n., 1975, p. 133—144.
21. **Darlington J., Burstall R. M.** A system which automatically improves programs.— Acta Informatica, 1976, v. 6, N 1, p. 41—60.
22. **Darlington J.** A synthesis of several sorting algorithms.— Acta Informatict, 1978, v. 11, N 1, p. 1—30.
23. **Ершов А. П.** Трансформационный метод в технологии программирования.— В кн.: Тезисы докладов I Всесоюз. конференции «Технология программирования». Киев: Б. и., 1979, с. 12—26.

24. **Барздинь Я. М.** Замечания о синтезе программ по историям их работы.— Уч. зап. Латвийск. ун-та, 1974, т. 210, с. 145—151.
25. **Барздинь Я. М., Бичевский Я. Я., Калниньш А. А.** Построение полной системы примеров для проверки корректности программ.— Уч. зап. Латвийск. ун-та, 1974, т. 210, с. 152—187.
26. **Калниньш А. А., Бичевский Я. Я., Барздинь Р. М.** Разрешимые и неразрешимые случаи проблемы построения полной системы примеров.— Уч. зап. Латвийск. ун-та, 1974, т. 210, с. 188—205.
27. **Барздинь Я. М.** Индуктивный вывод автоматов. функций и программ.— In: Proceeding, International Congress Math., Vancouver, v. 2, 1974, S. 1, 1975, с. 455—460.
28. **Biermann A. W.** Approaches to automatic programming.— In: Advances in computers, v. 15, N — 4. N. Y. Academic-Press, 1976, p. 1—63.
29. **Biermann A. W., Krishnaswamy R.** Constructing programs from example computations.— IEEE Trans. Seftware Eng., 1976, v. SE — 2, N 5, p. 141—153.
30. **Hardy S.** Synthesis of LISP functions from examples.— Advance Papers of the 4th International Joint Conf. on Artificial Intelligence. Cambridge, MIT, 1975, p. 240—245.
31. **Заславский И. Д.** Симметрическая конструктивная логика. Ереван: Б. и., 1978.
32. **Ершов Ю. Л.** Теория нумераций. М.: Наука. 1977.
33. **Kleene S. C.** Realizability: retrospective sur vey.— In: Cambridge Summer school in mathematical logic. Lecture Notes in Math., v. 337. Berlin — N. Y.: Springer — Verlag, 1973, p. 95—112.
34. **Клини С., Весли Р.** Основания интуиционистской математики. М.: Наука, 1978.
35. **Минц Г. Е.** О Е-теоремах.— Зап. науч. сем. Ленинград. отд. МИ АН СССР, т. 40, Л.: Наука, 1974, с. 110—118.
36. **Troelstra A. S.** Notes on intuitionistic second order aritmetic.— In: Cambridge Summer school in mathematical Logic. Lecture Notes in Math., v. 337. Berlin — N. Y.: Springer — Verlag, 1973, p. 171—184.
37. **Myhill J.** Constructive set theory.— J. Symbolic Logic, 1975, v. 40, N 3, p. 347—382.
38. **Pratt V. R.** Semantical consideration on Floyd — Hoare Logic.— Cambridge, MIT, 1976. 40 p.— (Project MAC, TR — 168).
39. **Непейвода Н. Н.** Применение теории доказательств к задаче построения правильных программ.— Кибернетика, 1979, № 2, с. 43—48.
40. **Гильберт Д., Бернайс П.** Основания математики. М.: Наука, 1979.
41. **Curry H. B., Feys R., Graig W.** Combinatory Logic, v. 1, 2. Amsterdam: North-Holland Company, 1958.
42. **Непейвода Н. Н.** Соотношение между правилами естественного вывода и правилами алгоритмических языков высокого уровня.— Докл. АН СССР, 1978, т. 239, № 3, с. 526—529.
43. **Непейвода Н. Н.** О построении правильных программ.— Вопросы кибернетики, 1978, № 46, с. 81—113.
44. **Непейвода Н. Н.** Об одном методе построения правильной программы из правильных подпрограмм.— Программирование, 1979, № 1, с. 11—21.
45. **Nepeivoda N. N.** The logical approach to programming.— In: Lecture notes in Computer Science, 122. Berlin a. o., 1981, p. 261—289.
46. **Nepeivoda N. N.** A proof-tneoretical comparison of program Synthesis and program vertification.— In: 6th International congress of Logic, methodology and philosophy of science (Abstracts), sections 1—4. Hannover. S. n., 1979, p. 7—11.
47. **Смирнов В. А.** Формальный вывод и логические исчисления. М.: Наука, 1972.
48. **Ершов Ю. Л.** Вычислимые функционалы конечных типов.— Алгебра и логика, 1972, т. 11, № 4, с. 367—437.
49. **Ершов Ю. Л.** Всюду определенные непрерывные функционалы.— Алгебра и логика, 1972, т. 11, № 4, с. 656—665.
50. **Ершов Ю. Л.** Теория А-пространств.— Алгебра и логика, 1973, т. 12, № 4, с. 369—416.
51. **Свириденко Д. И.** Некоторые вопросы математической семантики языков программирования.— В кн.: Вычислительные системы, № 67. Новосибирск: Б. и., 1976, с. 93—105.
52. **Свириденко Д. И.** Об одном классе моделей λ-исчисления без типов.— В кн.: Тезисы докладов IV Всесоюз. конференции по математической логике. Кишинев: Штиинца, 1976, с. 131.
53. **Маслов С. Ю.** Исчисления с монотонными выводами.— Зап. науч. сем. Ленинград. отд. МИ АН СССР, т. 88, Л.: Наука, 1979, с. 90—105.