МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ВС ИЗ МИКРО-ЭВМ
(Вычислительные системы)
1983 год Выпуск 96

УДК 681.142.2

О ПРИРОДЕ ПРОГРАММИРОВАНИЯ

Д.И.Свириденко

I. В в е д е н и е. Программирование как термин означает одновременно и д е я т е л ь н о с т ь и с о в о к у п н о с т ь з н а н и й, обслуживающих эту деятельность. Говоря о природе программирования, мы должны иметь в виду оба этих аспекта.

Наиболее впечатляющие тенденции в программировании - стремительный рост круга задач, увеличение сложности и масштабности реализуемых проектов. В определенной степени это связано со значительным улучшением технико-экономических характеристик ЭВМ. Прогресс технической базы программирования породил до сих пор живущую иллюзию о том, что с помощью ЭВМ можно решать сколь угодно сложные задачи. Однако действительность опровергает многие оптимистические прогнозы 50-х и 60-х годов. Вот уже более десяти лет говорят о кризисе в программировании, связывая его прежде всего с невысокой производительностью труда программиста.

Все многообразие рецептов, предлагаемых для выхода из сложившейся ситуации, сводится к двум основным тезисам. Первый звучит примерно так: "Мы до сих пор не научились и/или не умеем учить пользоваться тем инструментом, каковым является вычислительная машина фон Неймана". В связи с этим предлагается:

- продолжать увеличивать выразительную силу формализмов, ориентированных на машину фон Неймана;

- искать новые формализмы, не затрагивая при этом их принципиальной традиционной ориентации;

- пересмотреть систему обучения навыкам и знаниям программирования "с целью построения всеобщего и интенсивного курса обучения, направленного на активное формирование новой личности программиста". Программирование должно стать "второй грамотностью" [I].

Второй тезис можно сформулировать так: "Традиционное направление в своем основании имеет особенности, которые в настоящее время выступают как принципиальные ограничения, навязывая так называемый фон неймановский (традиционный) стиль программирования, и, следовательно, за основу нужно брать либо уже существующие (но отличные от традиционной) модели вычислений, либо искать новые и на их основе строить вычислительную науку и практику" (см., например, [2]).

Несмотря на принципиальное различе этих двух подходов, их объединяет понимание того, что дальнейшее развитие программирования должно предопределяться не столько его технической базой, как это происходило до сих пор, сколько его второй составляющей - ч е л о в е ч е с к и м ф а к т о р о м, значение которого не может быть устранено никаким техническим прогрессом. Таким образом, если говорить о становлении программирования как науки, то мыслить ее нужно прежде всего как гуманитарную. При этом речь должна идти не о поиске каких-то чудодейственных и сверхобщих принципов и положений, а о создании по-настоящему конструктивного знания, способного действенно помогать справляться с главной трудностью в программировании - с л о ж н о с т ь ю. И вполне естественно, что подобные исследования не могут обойтись без науки, имеющей многовековой опыт борьбы с этой проблемой, - математики.

Настоящая работа посвящена анализу некоторых тенденций в программировании с целью обоснования следующей концепции: программирование как научная дисциплина есть новая отрасль прикладной математики, а как деятельность - новый вид математической практики. И действительно, наиболее серьезные и перспективные результаты в программировании получаются в тех направлениях, которые либо эффективно редуцируются к хорошо разработанным математическим теориям, либо имеют столь отчетливое математическое содержание, что их формализация не представляет особого труда и приводит к интересным математическим моделям. Особо подчеркнем, что поскольку программа - это конструктивный объект алгоритмической природы, то основания программирования должны быть по своей сущности л о г и к о - м а т е м а т и ч е с к и м и.

2. О с н о в н ы е э т а п ы р а з в и т и я п р о г р а м м и р о в а н и я. Большинство существующих в программировании представлений о таких понятиях как "вычисление" и "программа" основывается на модели вычислительной машины, предложен-

ной фон-Нейманом в 1944 году [3]. Можно считать, что с этого момента и начинается история программирования. Но еще в 1936 году английский математик А.Тьюринг и независимо от него американский логик Э.Пост предложили свои варианты очень простых вычислительных машин. Описание этих гипотетических вычислительных устройств преследовало чисто теоретическую цель - формализовать интуитивное понятие вычислимой функции. Модель фон Неймана представляла собой уже и практический интерес. Несомненно, что идеи Тьюринга и Поста, которые, в свою очередь, восходят к идеям Паскаля, Лейбница, Бэббиджа и других мыслителей прошлого, оказали серьезное влияние на представления фон Неймана.

История программирования складывается из истории развития его технической базы и истории развития идей, относящихся собственно к деятельности по написанию программ. Эти направления тесно взаимодействуют, взаимно обогащая и обусловливая ход событий, который закономерно, как мы убедимся ниже, приводит нас к тезису, обсуждению которого посвящена данная статья.

Почти 40-летнюю историю программирования можно разделить на четыре этапа, приблизительно соответствующих 50-м, 60-м, 70-м и 80-м годам.

Первый этап - 50-е годы - годы открытия и описания большого числа языков программирования и методов их реализации (ФОРТРАН, АЛГОРИТМ-58 и т.д.). Появляются зачатки операционных систем - специальные программы, управляющие функционированием периферийных устройств. Преобладает э м п и р и ч е с к и й подход к поиску и реализации идей и концепций программирования. Так, например, языки программирования рассматривались вначале только как инструмент, облегчающий написание программ. Создание программ носит ярко выраженный индивидуальный характер. Идет накопление и осмысление большого фактического материала и наблюдений. Появляются такие понятия, как "открытая" и "замкнутая подпрограмма", "библиотека подпрограмм", "процедура", "компиляция", "интерпретация" и т.п. Выявляются новые области применения ЭВМ, анализируется специфика решаемых задач (вычислительных, коммерческих, обработки символьной информации и т.п.). Становится ясным, что успешное применение ЭВМ существенно зависит от их программного обеспечения и, в частности, от языковых средств. К концу этапа в развитии языков программирования начинают выявляться три концепции. Согласно первой, необходим единый универсальный язык программирования. Вторая основывается на идее базового языка с небольшим набором изобра -

зительных средств, допускающих возможность различных расширений (этими же средствами). И наконец, концепция системы языков специализированного и общего назначения: пользователь выбирает тот язык, который наиболее приемлем. В развитии отдельные части программ могут быть написаны на различных языках. В 50-х годах закладываются основы теории языков программирования, ищутся формализации понятия программы (схемы Янова). Это были романтические годы, полные грез и надежд.

На втором этапе в 60-е годы программное обеспечение охватывает уже ряд языков программирования высокого уровня (АЛГОЛ-60, КОБОЛ, PL/1, ЛИСП, СНОБОЛ и т.д.), достаточно развитые операционные системы и библиотеки подпрограмм. Языки программирования становятся объектом пристального внимания не только программистов, но и математиков. Активно ведутся исследования по теории формальных языков, формализации семантики языков программирования, теории схем-программ. Это были годы анализа и совершенствования ранее выработанных концепций и поиска новых. Появляется концепция параллельного программирования. Преобладает т е о р е т и ч е с к и й подход. Делаются попытки далеко идущих обобщений, создания общих концепций и даже общих теорий программирования. Начинают различать два вида программирования: концентрирующееся на программе и уделяющее основное внимание ее разработке (данные рассматриваются как объекты преобразований для программ) и концентрирующееся на данных, делающее основной упор на структуре данных при спецификации решаемой проблемы (программы играют подчиненную, обслуживающую роль). Первый подход характерен для вычислительных задач со сложными преобразованиями данных простой структуры. Второй - обычен при решении экономических задач и задач управления, для которых характерна сложная структура данных и простота действия над ними. В соответствии с этими подходами классифицируются и анализируются языки программирования, а также ищутся новые языковые средства. Стремительный рост сложности и масштабности конструируемых программ стимулировал попытки коллективного конструирования, носящего пока слабоорганизованный и стихийный характер. Хотя, как и ранее, не было недостатка в самых различных оптимистических прогнозах, однако большое число неудач при реализации больших проектов, заставило в конце 60-х годов открыто заговорить о кризисе в программировании.

Третий этап (70-е годы) характеризуется пониманием того, что само по себе имеющееся программное обеспечение не обеспечивает создание качественных, в частности, надежных программ. Начинает преобладать инженерный подход, выражающийся в разработке т е х н о л о г и ч е с к и х о с н о в программирования: в поиске конкретных методов управления разработкой, оценок эффективности, стоимости и надежности программ и т.п., т.е. программа уже рассматривается как промышленное изделие, хотя не теряет популярности и взгляд на программирование как на особый и сложный вид прикладного искусства. Появляются новые инструментальные средства, реализующие технологические открытия и поддерживающие различные технологии программирования. Четко прослеживается тенденция отхода от процедурных языков к непроцедурным и проблемно-ориентированным. Одновременно выдвигается тезис, что поскольку потребность в программах опережает возможности программистов, то необходимо применять универсальные или условно-универсальные программные и технологические средства: в конечном счете это оказывается более выгодным. Развиваются идеи параллельного программирования. Так как смена поколений ЭВМ влечет за собой уменьшение продолжительности жизненного цикла програмы, то возникает проблема сохранения накопленного программного продукта и переноса его на другие машины. Потребность ее решения сказывается на развитии технической базы программирования. Появляется концепция перестраиваемых вычислительных систем, способных сохранять совместимость с прежним программным обеспечением. Однако наиболее популярна идея встроенных эммуляторов и вспомогательных процессоров.

Столь же острой становится проблема сопровождения и модификации програмы. Один из предлагаемых технологией выходов - создание гибких и компактных операционных систем, позволяющих регулярным образом добавлять новые функции. Другой путь, также подсказанный технологией (в частности, практикой модульного и структурного программирования), - это системы программирования очень высокого уровня, способные настраиваться на решение конкретных задач и освобождать программиста от ненужных ему при построении программы технических подробностей: детали их функционирования планируются и реализуются самой системой. Задача предстает как гипотеза, а процедура ее решения приобретает характер машинного эксперимента. В частности, представляют интерес системы, способные формировать программы на основе формальных спецификаций задач или частных

примеров их решений. Однако оказалось, что чисто инженерная, технологическая точка зрения на программирование не способна ответить на вопрос, какие же принципы должны быть заложены в эти системы.

Поскольку в существовавших языках программирования высокого уровня основной акцент делался на облегчение создания программ, а не на доступ пользователя к данным, то возникла потребность такой формы организации информации, как базы данных, с которой программы пользователей взаимодействуют через специальную управляющую программу. Упомянутые системы управления базами данных выступают, таким образом, в роли мостика между программами и данными. Языковые средства таких систем, естественно, подразделяются на языки описания данных и языки манипулирования данными. В 70-х годах появляются базы данных, ориентированные на пользование многими потребителями*). Концепция баз данных и системы управления ими потребовала своей технологии. Однако вскоре выяснилось, что и здесь инженерный подход не может предложить удовлетворительного решения многих проблем.

Поскольку проблемная специфика решаемых задач может быть отражена не только в структурах данных, но и в методах и дисциплине их обработки, в конце 60-х - начале 70-х годов появляется концепция пакетов прикладных программ. Соответственно сразу же был поставлен вопрос о технологии их конструирования. В середине 70-х годов появились попытки сочетать достоинства обеих идей.

Нужно отметить, что основные проблемы в программировании, как правило, возникали из-за быстрого роста возможностей вычислительных устройств. В то же время практически отсутствовали устройства, решающие насущные проблемы программирования. Другими словами, развитие программирования в основном определяла аппаратурная часть, но не наоборот. Технологическое знание не способно было изменить это соотношение, которое противоречило самой сущности программирования, поскольку для последнего отсутствуют принципиальные физические ограничения, столь сильно доминирующие в инженерных науках. Кроме того, становится понятным, что постулируемые и внедряемые в практику технологические принципы должны вытекать из более глубокого и всестороннего проникновения в природу

*) Похоже, что возникшая тенденция централизации данных и управления ими сохранится и в 80-х годах, хотя привлекательность этой идеи заметно уменьшилась.

программ и проблему их конструирования. На повестку дня был поставлен вопрос о получении специальных знаний, относящихся к научному формированию мотиваций в определенной системе понятий и обосновывающих направление как теоретических, так и технологических разработок. Речь зашла о разработке нового раздела программистских знаний - м е т о д о л о г и и п р о г р а м м и р о в а н и я. Конец 70-х - начало 80-х годов - период активных методологических исследований. Методология оказалась тем синтезирующим началом, которого так не хватало программированию. И результаты не заставили себя ждать: программные логики и верификация программ, функциональное программирование, абстрактные типы данных, реляционные базы данных, логический подход к программированию, композиционное и трансформационное программирование, индуктивный синтез программ. Элементы этих подходов существовали в той или иной форме и до появления методологии программирования. Но методология сознательно акцентирует свое внимание на корректном извлечении из мотиваций интуитивно ощущаемых проблем, потребностей и т.п. точных их формулировок с целью формирования теоретического представления о программе и о деятельности по ее созданию, постулируя принцип, что только после методологической проработки проблемы можно переходить к теоретическим исследованиям и технологической апробации найденных решений [4]. И здесь выявляется то обстоятельство, о котором речь шла во введении - все эти уточненные представления оказываются л о г и к о - м а т е м а т и ч е с к и м и по своей природе. Думается, что в 80-е годы методология программирования в содружестве с математической логикой сохранит и еще более укрепит свое лидирующее положение в структуре программстских знаний.

Ниже дается краткий анализ некоторых направлений в методологии, теории и технологии программирования. Из-за ограниченности объема не будут затронуты такие области, как верификация программ и программные логики, методы распараллеливания, теория синтаксического анализа, теория сложности алгоритмов, теория иерархических, сетевых и реляционных баз данных, методы оптимизации и преобразования программ, операционные системы, теория трансляции.

3. М о д е л и п р о г р а м м. К настоящему времени в теории алгоритмов известно достаточно много вариантов формализации понятия алгоритма: уже упоминавшиеся машины Тьюринга и Поста, рекурсивные функции, λ-определимые функции, нормальные алгоритмы Маркова, схемы определений Эрбрана-Геделя, алгоритмы Колмогорова-Успенского и т.п. Поскольку программа - это алгоритм, предназна -

ченный для исполнения на ЭВМ, то не удивительно, что эти определения в той или иной мере нашли свое применение в программировании. Однако все эти формализации оказались непригодными для представления структурных свойств программ и исследования их преобразований. В связи с этим обстоятельством в 50-х годах в теоретическом программировании возникает направление, известное под названием теории схем программ, основы которого были заложены советскими математиками А.А.Ляпуновым, Ю.И.Яновым, Б.М.Глушковым, А.П.Ершовым, С.С.Лавровым.

Схема программы - это математическая модель алголоподобной программы, в которой представлена структура программы при условии абстрагирования от конкретного содержания функций и отношений, входящих в эту программу (заменяя их функциональными и предикатными символами соответствующей арности). Для теории схем программ основными проблемами являются исследования различных классификаций (синтаксических и семантических), функциональной и других видов эквивалентности схем-программ, проблемы тотальности, пустоты и трансляции одних классов схем программ в другие. Основными объектами исследований являются различные классы схем, отражающие структурные особенности реальных программ: структурированные схемы, стандартные схемы, схемы с процедурами, рекурсивные схемы программ, обогащенные схемы (со счетчиками, массивами, магазинами и другими типами данных и интерпретированных операций над ними), параллельные схемы программ, сети Петри, потоковые схемы и т.д.. Имеется обширная литература (см., например, [5-7].

Результаты теории схем-программ нашли свое применение в задачах обоснования алгоритмов трансляции, в частности, при построении оптимизирующих трансляторов и в других областях [5]. Несомненное влияние теория схем программ оказала на такие направления, как верификация и семантика программы, на исследования в области алгоритмических логик. Теория рекурсивных схем сыграла значительную роль в становлении функционального программирования. Наконец, структурированные схемы и результаты, касающиеся трансляции этого класса схем в другие, помогли уточнить представления о возможностях структурного программирования [8], базирующегося на известной теории Глушкова-Бема-Якопини [9,10]. С другой стороны, вряд ли можно считать схемы программ удовлетворительным во всех отношениях вариантом уточнения понятия программы. Как и классические сис-

темы, они уточняют лишь определенный аспект понятия алгоритма (см. ниже).

4. С е м а н т и к а п р о г р а м м и а б с т р а к т н ы е т и п ы д а н н ы х. Любая программа пишется для чего-то, кем-то и предназначается определенному адресату. Таким образом, программа обладает, по крайней мере, тремя видами свойств:

- с и н т а к с и ч е с к и м и, отражающими материал и структуру программы;

- с е м а н т и ч е с к и м и, представляющими особенности значения или, как еще говорят, денотата программы, ее "смысла", для выражения которого программе придают ту или иную форму;

- п р а г м а т и ч е с к и м и, отражающими тот контекст, в котором пишется и/или используется программа.

Естественно, что в процессе написания семантический уровень восприятия программы подчинен прагматическому, а синтаксический аспект - семантическому. Для процесса же понимания уже готовой программы исходными являются ее синтаксические свойства. Именно поэтому столь важной проблемой является задача превращения семантического и даже прагматического аспектов в синтаксический. До определенного момента развития программирования синтаксические свойства программ и их неформальные толкования вполне удовлетворяли нужды практики. Однако неточности, двусмысленности, противоречия, которые сопровождают процесс создания и внедрения практически любого языка программирования все острее ставили перед программированием вопрос о семантических исследованиях. Кроме того, стало очевидным, что дальнейшее развитие самих синтаксических аспектов существенно зависит от успехов в области уточнения семантики языков программирования. Понятно, что семантические исследования могут вестись в различных направлениях, на разных уровнях абстракции и формализации, преследовать различные прагматические цели. Например, можно мыслить себе семантику языков в терминах м о д е л е й р е а л и з а ц и и. Так, модель языка АЛГОЛ-60 основывается на конструкции стека записей активации; два стека и одна куча - модель реализации языка АЛГОЛ-68. Подробно с этим подходом к определению семантики языков программирования можно ознакомиться, например, в [II,I2]. Техника моделей реализации нашла свое применение при написании трансляторов.

Смысл программ можно также определять либо в терминах результатов выполнения каждой ее составляющей конструкции некоторой

(возможно,гипотетической) вычислительной машиной (такое определение обычно предназначено для интерпретации), либо в терминах целей объектных программ, получаемых при ее компиляции. Таким образом, "семантика программы" - понятие относительное,и выбор того или иного определения зависит от прагматики. Если речь идет о вычислениях, то естественна ориентация семантики на компиляцию,если о процессе выполнения - на интерпретацию. Однако каждая программа с точки зрения любой ее семантики ставит в соответствие исходным данным результаты , т.е. является функцией. Следовательно, для строгого изучения семантики программ, нужно прежде всего уточнить способ описания функций, задаваемых программами. Например,при интерпретационном подходе семантика описывается в терминах функций переходов состояний некоторой абстрактной машины-интерпретатора, где под состоянием понимается функция типа $\underline{Var} \to D$, отображающая множество $\underline{Var}$ всех переменных языка (отождествляемых с именами ячеек памяти) в множество D всех возможных значений переменных (содержимое ячеек). Процесс выполнения программы π, начиная с состояния s_0: $\underline{Var} \to D$, формализуется (как и в теории схем программ) в виде последовательности конфигураций (возможно, и бесконечной): $k_0 = (s_0, \pi), k_1, k_2, \ldots$. Здесь конфигурация k_i - это пара (s_i, π_i), где s_i - состояние машины на i-м шаге исполнения программы π ,а π_i - список операторов, которые предстоит выполнить (возможна и более сложная формализация). Таким образом, семантику программ можно представить себе в виде частичной функции $f_\pi: K \to K$,где K - множество всех возможных конфигураций. Такая семантика, названная о п е р а ц и о н н о й, или в ы ч и с л и т е л ь н о й, обычно применяется при использовании информации или учете ситуаций,возникающих по ходу выполнения программы. В этом смысле операционная семантика - наиболее полная и является фундаментом, на котором строятся и ориентируются все другие формальные способы описания семантики языков программирования.Наиболее продвинутое воплощение концепция операционной семантики получила в методе VDL (Венский язык определений), разработанном в 60-х годах в Венской лаборатории фирмы IBM под руководством Лукаса [I3]. Идеи этого метода восходят к работе Маккарти [I4], первым описавшего формальную семантику подмножества АЛГОЛ-60. VDL был использован в некоторых проектах, в частности для описания семантики АЛГОЛ-60 и PL/1 . Оказалось, что метод труден для использования и плохо воспринимаем. Впоследствии на основе алгебраического подхода и концепции

абстрактных типов данных он был трансформирован там же в метод VDM (Vienna Development Method) [I5].

Другой подход, при котором отвлекаются от деталей описания всего процесса выполнения программы, демонстрирует *денотационная*, или *математическая* семантика. Методология этого подхода основана на заимствованном у математической логики принципе: семантика языка программирования определяется функцией, которая соотносит программе ее математический смысл по математическому смыслу ее копонент. Денотатом программы служит функция, преобразующая состояние памяти некоторой абстрактной машины. Основные трудности при построении денотационной семантики оказались связанными с описанием семантики рекурсивных программ. И если для денотационного описания простой рекурсивной программы достаточно было воспользоваться теоремами Тарского и Клини о существовании наименьших неподвижных точек у монотонных и непрерывных функций полных решеток в себя (для теоремы Клини достаточно ограничиться ω-полными частично упорядоченными множествами), то описание "эффекта самоприменимости" оказалось трудным "орешком". Эта проблема, давно известная математическим логикам в связи с попытками построить математическую интерпретацию λ-исчисления без типов, была независимо решена Д.Скоттом и Ю.Л.Ершовым, в начале 70-х годов [I6-I8] (см.также [I9]). С конструктивной точки зрения предпочтительней оказалась конструкция f-пространств и их ретрактов - A-пространств, предложенных Ю.Л.Ершовым. Адекватность денотационной семантики в рамках λ-исчисления продемонстрирована в [20]. Есть основания считать, что денотационная семантика найдет применение при конструировании доказуемо правильных трансляторов. Однако в основном денотационная семантика применяется при формальных исследованиях свойств программ [26]*). В частности, она оказалась полезной для обоснования аксиом и правил вывода *алгоритмических* или *программных* логик, специально создаваемых для формулировок и доказательства утверждений о семантических свойствах программ (функциональная эквивалентность, корректная завершаемость и т.п.). Поскольку денотационная семантика конкретного языка программирования является математическим объектом, то она, в свою очередь, также может быть описана в

*) Достаточно полные обзоры по денотационной семантике языков программирования содержатся в [7,2I-25]. К настоящему времени описаны денотационные семантики нескольких реальных языков программирования, тем самым продемонстрирована непротиворечивость.

рамках некоторого формального языка (см., например [27]). Если мы согласны с тем, что описание смысла программы в терминах математических объектов - функций может сделать программу более понятной, то естественно использовать этот формальный язык в качестве языка программирования. Так мы приходим к концепции функционального программирования. Это же рассуждение также позволяет рассматривать денотационную семантику программ как новый вид математической модели программ.

Поскольку аксиоматические утверждения о программах можно считать (наряду с операционной и денотационной семантиками) самостоятельным видом семантики, то появляется аксиоматический подход к ее формальному описанию. Большое значение для такого подхода имеет выбор языка и исчисления утверждений о программах, системы аксиом и правил вывода. В настоящее время аксиоматический подход рассматривается в рамках теории программных логик [29-31].

В [28] содержатся описания других, отличных от вышеприведенных подходов к построению формальных семантик программ. В последнее время все большее внимание по причинам, указанным в заключении, уделяется теоретико-категорному подходу [32-36]. Более детальная точка зрения на семантику программ связана с понятиями "структура данных" и "тип данных", которые играют важную роль в современном программировании. В последние годы эта точка зрения оформилась в направление, известное под названием "абстрактные типы данных". По нему имеется большое число публикаций, отражающих различные взгляды на природу понятий "структура данных" и "тип данных" *). Пример последовательного и конструктивного подхода к абстрактным типам данных на основе теории нумераций и конструктивных моделей продемонстрирован в [39].

Для наших целей важно то, что понятие "абстрактный тип данных" и создаваемая на наших глазах математическая теория этого понятия явились закономерным этапом развития понятия "модуль". Модульное программирование в лице теории абстрактных типов данных приобретает прочный методологический и теоретический фундамент, который имеет ярко выраженную логико-математическую природу. Наполняется новым содержанием технология модульного программирования. Идеи и техника абстрактных типов данных также нашли свое примене-

*) Можно порекомендовать сборник [37] (и особенно обзорную статью В.А.Агафонова [38] с достаточно полной библиографией).

ние в новых языках программирования [40,41], в автоматизированном синтезе программ.

5. Функциональное программирование. Формирование и значимость различных направлений в программировании, как и в других областях науки, связано с появлением и развитием новых понятий. Например, понятие "открытой подпрограммы" дало началу такому направлению, как микропроцессоры [42], а "замкнутая программа" стало одним из фундаментальнейших понятий всего программирования и послужило отправной точкой многих подходов и концепций. Об одном из них - модульном программировании - уже шла речь выше. То, что оба этих понятия могут быть положены в основу универсальных алгоритмических систем, хорошо известно математическим логикам: примерами таких систем в первом случае могут служить системы определений Эрбрана-Геделя или алгоритмы Маркова, во втором - рекурсивные функции или λ-определимые функции. В программировании на основе этих моделей были созданы такие системы программирования, как РЕФАЛ [43] и ЛИСП [44]. ЛИСП представляет собой пример функционального языка и связан с обработкой списковых структур данных. Простота языка ЛИСП позволила разработать его математическую модель, которая, в свою очередь, явилась отправной точкой для развития современной математической теории вычислений: денотационной и аксиоматической семантик, верификации и синтеза программ и т.д., а интерпретация языка ЛИСП в терминах функции APPLY привела к появлению операционной семантики языков программирования.

В функциональных языках, противопоставляемых традиционным, или "операторным", языкам, каждая программа - это функция, аргументами которой являются входные данные, а значением - искомый результат вычислений. Если в "операторных" языках требуется явное указание последовательности выполнения инструкций, составляющих программу, то в функциональных языках последовательность действий при применении функции - программы определяется вычислением ее аргументов. Именно в этом смысле функциональное программирование носит более математический характер, чем традиционное, в котором организация вычислений мыслится в терминах смены состояний памяти вычислительной машины, т.е. в организации использования оператора присваивания и его аналогов. Центральную роль в функциональном программировании играет рекурсия, точнее, рекурсивные определения,

в то время как в традиционном программировании рекурсия встречается, как правило, либо в виде рекурсивных вызовов, либо в виде итераций команд, или более крупных фрагментов.

По Дж.Бэкусу [2], оператор присваивания - это "узкое горлышко" традиционных языков программирования, ориентированных на машину фон Неймана, который повинен и в неэффективности вычислений, порождаемых традиционными программами (поскольку каждое выполнение этого оператора связано с большим объемом обмена информацией между различными компонентами вычислительной системы), и в сложности понимания самих программ. К главным достоинствам функционального стиля программирования, помимо "математичности" смысла программы, Дж.Бэкус относит возможность порождать в соответствии с природой решаемых задач новые структуры управления (их роль в формализме Бэкуса играют функциональные формы), а также самоприменимость языка функционального программирования для исследования семантических свойств программ этого же языка (традиционное программирование для этих целей вынуждено привлекать другие языки, например, языки исчисления предикатов). Более того, денотационная семантика языка функционального программирования достаточно очевидна, легко строится и описывается терминами самого же языка. Однако никакого чуда здесь нет - за всеми прелестями функционального программирования скрывается сложная и громоздкая процедура декодирования объектов, поступающих на вход машины Бэкуса. Поэтому чисто функциональный смысл программирования хотя и устраняет все трудности, связанные с оператором присваивания, тем не менее создает новые [45]. Существуют области, где применение языков функционального программирования вполне оправдано, что стимулирует их дальнейшее развитие. Но вряд ли имеет смысл полностью отождествлять природу программирования с функциональным подходом.

6. Технология программирования. История науки подсказывает, что наиболее успешной стратегией борьбы со сложностью является ее "декомпозиция". Многие инженерные дисциплины накопили богатый опыт успешного использования этой стратегии (особенно для преодоления организационных трудностей). Поэтому когда в 60-х годах программирование столкнулось с неудачами при реализации больших и сложных проектов, то вполне естественно, что программисты обратились к данному опыту. На программу стали смотреть как на промышленное изделие, а проблемы конструирования рассматривались как инженерные. Появился и быстро внедрился в

лексикон программистов термин "технология программирования". То обстоятельство, что с инженерной точки зрения программы - это вид промышленных изделий, заставило многих обратить серьезное внимание на качество этих изделий и на оптимальность процессов их конструирования [46-50]. Несомненной заслугой технологии программирования является включение в область интересов программистов таких аспектов, как диагностика потребностей в программах, оценивание и выбор программистских решений и т.д.

Неудачные попытки простого копирования технологических решений других инженерных наук заставили сделать вывод, что созданию и внедрению любой технологии программирования должно предшествовать внимательное и серьезное изучение самой программистской деятельности и ее продуктов - программ. А это уже задачи методологии [4]. И здесь инженерная точка зрения подсказывает нам следующий вывод - то, что действительно роднит программы с промышленными изделиями, так это быть искусственным объектом. Поэтому если мы хотим действительно понять природу программ и программистской деятельности, то нужно попытаться прежде всего понять своеобразие искусственных объектов.

Существует несколько способов охарактеризовать объект. Первый и самый простой - демонстрация данного или похожего объекта. Тем самым демонстрируем материал и структуру, т.е. морфологические характеристики, и получаем М-описание объекта. Второй путь - указание на функциональные характеристики (F-описание) объекта. Однако одни и те же F-характеристики могут присутствовать у объектов разной морфологии; F-описание объекта может меняться в зависимости от ситуации, в которой оказывается объект. Поэтому наибольший интерес представляют характеристики объекта, описывающие соответствие между его М- и F-характеристиками. Такие характеристики называются атрибутивными, а соответствующее описание - А-описанием. Наличие А-характеристик объекта разделяет F-характеристики на два класса: первый включает чисто функциональные, которые безразличны к морфологии, второй класс состоит из тех F-характеристик, которые являются проявлением возможностей объекта, генетически заложенных в его морфологии. Последние характеристики назовем FA-характеристиками.

Все вышеприведенные способы описания характерны для любого объекта. Искусственность объекта дает еще один чрезвычайно важный

для нас способ - через его происхождение (будем называть такой способ г е н е т и ч е с к и м). Обычно генетическое описание представляет собой описание процесса конструирования объекта в виде совокупности описаний промежуточно-устойчивых форм существования данного процесса, включая возможные формы дальнейшего существования конструируемого объекта. Эта совокупность форм называется еще ж и з н е н н ы м ц и к л о м о б ъ е к т а.

При конструировании объекта, чтобы достичь определенной цели, мы соединяем между собой или видоизменяем другие объекты. Целесообразность объекта отражена в его А-характеристиках и проявляется в FA-характеристиках. Чем точнее мы представляем себе процесс конструирования объекта, тем полнее наше знание о его А-характеристиках, т.е. последние являются л о г и ч е с к и м с л е д с т в и е м генетического описания. Поскольку нам не всегда известно описание действительно имевшего место процесса конструирования объекта, то мы можем лишь предполагать о том, как он протекал. Однако при этом мы должны потребовать, чтобы такое гипотетическое генетическое описание объекта было в е р и ф и ц и р у е м ы м, т.е. принципиально осуществимым. Итак, можно дать следующее методологическое определение: искусственный объект - это объект, атрибутивное описание которого л о г и ч е с к и с л е д у е т из некоторого верифицируемого генетического описания. Из этого определения вытекает очень важное методологическое положение - если мы знаем генетическое описание объекта и логику его конструирования, то мы можем извлечь из этого знания А-характеристики объекта. Таким образом, большое значение приобретает знание л о г и к и п о с т р о е н и я объекта, поскольку из конструкции этой логики, описывающей процесс конструирования объекта, может быть извлечено логическими средствами его А-описание. Применительно к программированию, где в качестве М-описания программы выступает ее текст на языке программирования, FA-описание - это спецификация программы, А-описание - сама аннотированная программа или ее итоговая пользовательская документация, и, наконец, генетическое описание - это описание технологии ее конструирования, этот вывод становится основным положением логического подхода к программированию [4, 57, 60].

7. С и н т е з п р о г р а м м. В последнее время в литературе по программированию все чаще стал встречаться термин "синтез программ". Попытки придать точный смысл задаче синтеза про-

грамм привели к появлению многочисленных формулировок и подходов к ее решению. Анализ подобных определений позволяет выделить четыре основных момента, так или иначе в них присутствующих:

1) исходная задача, для решения которой предназначается синтезируемая программа (описание задачи носит название спецификации задачи, а язык, в терминах которого это описание фиксируется, - языком спецификаций);

2) набор "элементарных" операций, из которых собирается программа;

3) набор правил композиции, по которым происходит сборка программы;

4) набор правил декомпозиции исходной информации.

К настоящему времени сложились следующие основные подходы к решению задачи синтеза программ: композиционный, трансформационный, дедуктивный и индуктивный.

При к о м п о з и ц и о н н о м подходе [45,51,52] основным объектом изучения являются пп. 1 и 2, т.е. задача синтеза программ рассматривается как задача композиции программ относительно разных наборов первичных конструкций и правил их композиции. Более того, предлагается уточнение природы программирования свести к уточнению универсального класса композиций, поскольку считается, что механизмы, лежащие в основе процесса программирования, по необходимости устроены проще, чем множество возможных программ, т.е. делается упор на п.2. Спецификация правил композиции в первых работах носили ярко выраженный императивный характер. В последних работах по композиционному программированию основные композиции (коннетативные и денотативные) предлагается уже рассматривать как правила вывода некоторой программной логики. Считается, что множество выводов этой логики адекватно уточняет всю совокупность процессов конструирования программ. В композиционном программировании п.3, как правило, не рассматривается, следовательно, и не исследуются правила декомпозиции.

Т р а н с ф о р м а ц и о н н ы й подход [53-55] требует с самого начала существование алгоритма, который в дальнейшем должен быть "трансформирован" в программу. Запись этого алгоритма может внешне выглядеть вполне декларативной и рассматривается как своеобразная форма записи исходной задачи. Набор элементарных операций соответствует операторам и подпрограммам языка программирования, в терминах которого будет записана синтезируемая про-

грамма. Набор правил декомпозиции задается как набор "трансформаций". Каждое правило "трансформации" одновременно определяет (неявно) набор правил композиции. Одной из серьезных и центральных математических проблем в трансформационном синтезе является переход от рекурсивных определений к итерациям, поскольку итеративная запись алгоритма более эффективно реализуется на машинах фон Неймана.

Среди правил "трансформации" выделяются фундаментальные, которые можно рассматривать в определенном смысле как правила вывода "трансформационного исчисления". Другие правила носят вспомогательный характер, и зачастую их появление оправдывается программистской практикой (см., например, [53]). Таким образом, налицо две тенденции в трансформационном программировании: с одной стороны, поиски строгих логико-математических оснований, а с другой - попытки обогатить стратегии трансформации эвристическими приемами.

Дедуктивный [4,56-60] и индуктивный [6I-64] подходы с самого начала опираются на формальные логические системы. Если в основу синтеза программ положена дедуктивная логика, то мы имеем дело с дедуктивным подходом, если индуктивная логика - то с индуктивным. В действительности это различие условное. Задача индуктивного синтеза программы формулируется так: в фиксированном языке (спецификаций) формулируется исходная задача и приводится конечный набор описаний параметров ее решения с помощью какого-то алгоритма; требуется по этой информации синтезировать (восстановить) программу, реализующую упомянутый алгоритм. Заметим, что исходный конечный набор примеров историй вычислений в формулировке задачи индуктивного синтеза программ есть не что иное, как набор правил декомпозиции.

В случае дедуктивного синтеза процесс построения программ можно представить как поиск вывода в некоторой дедуктивной системе. Здесь мы можем либо формулировать дедуктивные правила композиции программ, непосредственно генерирующие и преобразующие аннотированные программные тексты, либо формулировать чисто логическое исчисление, дающее нам возможность после завершения доказательства получить программу. Во втором случае можно столкнуться либо с неэффективностью, когда доказательство может дать неалгоритмическое решение (например, классическая логика), либо с очень большой сложностью алгоритма преобразования доказательства в программу, либо, наконец, со сложностью, самой извлекаемой из доказательства программы. При дедуктивном подходе естественно интерпре-

тировать правила вывода соответствующей формальной системы как правила декомпозиции; язык формальной системы выступает как язык спецификаций.

Таким образом, во всех вышеупомянутых подходах к синтезу программ отчетливо прослеживается их ориентация на использование идей и методов математической логики.

8. З а к л ю ч е н и е. Тезис о том, что программирование имеет математическую природу, у некоторых специалистов вызывает возражение. Основу их позиции составляет концепция, согласно которой программирование требует особого "программистского стиля мышления". Однако формулировка этих требований (см., например, [65]) не выдерживает сколько-нибудь серьезной критики, поскольку эти требования с полным правом могут быть отнесены и к характеристике деятельности прикладного математика. Исключение составляет требование, касающееся умения пользоваться ЭВМ. Но и это требование характерно для любой деятельности, в структуру которой встраивается новый инструмент. Справедливости ради стоит отметить, что вычислительная машина не совсем обычный инструмент и, возможно, с этим обстоятельством связано желание выделить программирование в особый вид деятельности. Действительно, появление и дальнейшее развитие вычислительной техники оказало серьезное влияние на существовавшие до того времени представления о методах обработки информации. На наших глазах меняется статус понятия алгоритма, его роль и значение в интеллектуальной деятельности человека, в том числе и в математической. Становится все более очевидным, что имеющиеся формализации этого понятия не вполне удовлетворительны с точки зрения практики и теории вычислений. Так, например, до сих пор нет общепринятых теоретических основ оперирования с объектами, которые могут быть представлены в памяти ЭВМ (реальные математические объекты [66]). Другими словами, речь идет о необходимости создания такой теории алгоритмов, в которой одним из основных понятий было бы понятие "ресурс". Однако наиболее неудовлетворительным с разных точек зрения в современной теории алгоритмов, по мнению специалистов, является существующая теоретико-множественная точка зрения на алгоритм. Как отмечал в своей лекции на II Школе молодых ученых Сибири и Дальнего Востока по современным проблемам математики Ю.Л.Ершов "... в математике создалась уникальная ситуация: мы не располагаем общим понятием, что такое процесс вычисления или вычислительная машина, относительно которого все согласились

бы, что оно адекватно, но мы располагаем точным определением того, что такое вычислимая функция. То есть мы знаем, что такое алгоритм как функция, но не знаем, что такое алгоритм как процесс...". Пожалуй, за такую ситуацию должна нести ответственность современная теория множеств, которая отождествляет функцию с ее графиком*), хотя первоначальное интуитивное представление о функции имеет несомненно динамический характер. Альтернатива языку множеств - язык категорий или язык λ-исчисления - позволяет более адекватно отразить этот аспект понятия функции. Возможно, что на этом пути будет найдена приемлемая формализация понятия алгоритма как процесса. Итак, с одной стороны, практика и теория программирования становятся серьезным заказчиком новых математических конструкций и идей, а с другой - использование ЭВМ в математической деятельности может серьезно изменить (и уже меняет) ее структуру и содержание. Применение ЭВМ дает возможность осуществлять эксперименты, строить математические модели, которые могут одновременно выступать и как источники новых гипотез, и как средство их проверки. Прикладная математика приобретает все более экспериментальный и эмпирический характер. В связи с этим поднимается вопрос о создании методологии и технологии прикладной математической деятельности [67] и процесса обучения математике, который часто сводится только к изучению уже полученных знаний, а не к тому, как эти знания применять и как они были получены. Смещение акцентов в сторону этих аспектов, в свою очередь, потребует определенной "конструктивизации" математических знаний.

Возвратимся к обсуждению тезиса об особом "программистском стиле мышления". Во-первых, умение создавать объекты с нужными свойствами, столь же необходимо математику, как и программисту. Во-вторых, программу можно мыслить себе двояко: как некую абстракцию (рекурсивная функция, абстрактный тип данных, теорема, доказательство, система определений, набор примеров), существующую независимо от того, есть или нет реальная вычислительная машина, способная исполнить ее (т.е. программа трактуется как м а т е м а т и ч е с к и й о б ъ е к т), и как реалию, поскольку этот математический объект предназначен для исполнения на ЭВМ и должен обладать определенными свойствами, прежде всего эффективностью т.е. рациональным (в идеале-оптимальным) использованием предостав-

*) Как это часто бывает, подобная абстракция вскоре стала восприниматься как абсолютное определение функции.

ляемых ресурсов (как при конструировании программы, так и при ее исполнении на ЭВМ). На наш взгляд, роль программирования как научной дисциплины должна заключаться в наполнении математическим содержанием как технологических, так и творческих аспектов программирования, т.е. в превращении программирования как деятельности в разновидность математической практики. Образно говоря, если эвристический, чисто прагматический подход к решению проблем программирования напоминает процесс решения задач человеком, обладающим способностью интуитивно угадывать решение, то строгий математический подход - это способность не только решить задачу, но и продемонстрировать ход решения и объяснить, почему он протекал так, а не иначе. Как подсказывает история науки, только при таком подходе появляется реальная возможность с и с т е м а т и ч е с к и решать действительно сложные задачи [68].

Литература

I. ЕРШОВ А.П. Программирование - вторая грамотность. - Новосибирск: Б.и., 1981. - 18 с. - (Препринт/ВЦ СО АН СССР).

2. BACKUS I. Can programming be liberated from the Von Neumann style? A functional style and its algebra of programms. - Comm.ACM,1978,v.21,N 8,p.613-641.

3. Von NEUMANN I. The EDVAC report. - In: Computer from Pascal to von Neumann/ Ed. H.Goldstein.- Prinston:Prinston Univ.Press 1972.

4. НЕПЕЙВОДА Н.Н., СВИРИДЕНКО Д.И. Программирование с логической точки зрения. - Новосибирск: Б.и., 1981. - 49 с.-(Препринт/ИМ СО АН СССР).

5. КОТОВ В.Е. Введение в теорию схем программ. -Новосибирск: Наука, 1978. - 257 с.

6. GREIBACH S.A. Theory of Program Structures: Shemes, Semantics,Verification.- Lec.Notes in Comp.Sci.- Berlin-Heidelberg-New York: Springer-Verlag,1975,v.35.-364 p.

7. НЕПОМНЯЩИЙ В.А., ДЕХТЯРЬ М.И. Математическая теория программирования. -Новосибирск: Б.и., 1982. - 52 с. -(Препринт/ВЦ СО АН СССР; 341).

8. ДАЛ У., ДЕЙКСТРА Э., ХООР К. Структурное программирование. - М.: Мир, 1975. - 275 с.

9. ГЛУШКОВ В.М. Теория автоматов и формальные преобразования микропрограмм. - Кибернетика, 1965, №5, с. 1-10.

10. BÖHM C.,IACOPINI G. Flow diagrams, Turing machines and languages with only two formations rules.- Comm.ACM,1969,N 9, p. 366-371.

11. WAGNER P. Programming languages, information structures and machine organization.- New York-London: McGraw-Hill Book Company,1968.- 401 p.

I2. ПРАТТ Т. Языки программирования. Разработка и реализация. -М.: Мир, 1979. - 574 с.

I3. ОЛЛОНГРЕН А. Определение языков программирования интерпретирующими автоматами. -М.: Мир, 1977. - 288 с.

I4. McCARTHY I. A formal definition of a subset of ALGOL.-In: Formal description lenguages, Proc.IFIP Conf.- Amsterdam: North Helland Publ.Comp.,1966,p.1-12.

I5. The Vienna Development Method: the Meta-Languages.- Lect. Notes in Comp.Sci./ Ed. D.Bjorner and C.B.Iones.- Berlin-Heidel - berg-New York: Springer-Verlag,1978,v.61.- 382 p.

I6. SCOTT D. Continuous lattices.- In: Toposes, Algebraic Geometry and Logic, Lecture Notes in Math.- Berlin: Springer-Verlag, 1972,v.274,p.97-136.

I7. ЕРШОВ Ю.Л. Вычислимые функционалы конечных типов. -Алгебра и логика, 1972, т.II, № 4, с. 367-437.

I8. ЕРШОВ Ю.Л. Теория -пространств. - Алгебра и логика,1973, т. I2, № 4, с. 369-416.

I9. SCOTT D. Some ordered sets in computer science.- In: Ordered Sets/ Ed.I.Rivel,D.Reidel. Publ.Comp.,1982,p.677-718.

20. САЗОНОВ В.Ю. Последовательно и параллельно вычислимые функционалы. - Сиб.мат. журнал, 1976, т.XVII, № 3, с. 648-672.

2I. ЛАВРОВ С.С. Методы задания семантики языков программирования. - Программирование, 1978, № 6, с. 3-I0.

22. STOY I.E. Denotational semantics: the Scott-Strachey approach to programming languages theory.- Cambridge: M.I.T.Pres , Mass.,1977.- 414 p.

23. STOY I.E. Foundations of denotational semantics.- In: Abstract Software Specification, Lect.Notes in Comp.Sci.- Berlin - Heidelberg-New York: Springer)Verlag,1980,v.86,p.43-99.

24. GORDON M.I.C. The denotational description of Programming languages. An Introduction.- Berlin-Heidelberg-New York: Springer -Verlag,1979.- 160 p.

25. SEMANTICS of concurrent computation: [The mat.coll.] - Lecture Notes in Comp.Sci. - Berlin-Heidelber-New York: Springer-Verlag,1970,v.70. - 368 p.

26. МАННА З. Теория неподвижной точки программ. -В кн.: Кибернетический сборник, № I5, 1978, с. 38-I00.

27. MOSSES P.R. The mathematical semantics of Algol-60.- Oxford,1974.- 120 p.- (Technical Monograph PRG-12).

28. Семантика языков программирования: Темат.сб. /Пер. с англ. - М.: Мир, 1980. - 396 с.

29. HAREL D. First-order dynamic logic.- Lect.Notes in Comp. Sci. - Berlin-Heidelberg-New York: Springer-Verlag,1979,v.68. - 133 p.

30. ПЛЮШКАВИЧУС Р.А., ПЛЮШКАВИЧЕНЕ А.Ю., САКАЛАУСКАЙТЕ Ю.В., ЮКНА С.П. О программных логиках. - Кибернетика, 1979, № 2, с. I2-I9.

31. ART K.P. Ten years of Hoare's logic: a survey (part 1). - ASM Transactions on Prog.Languages and System,1981, v.3, N 4,p.431-483.

32. LEHMAN D.I., SMYTH M.B. Algebraic specification of data types: a synthetic approach.- Math.Systems Theory,1981,v.14,p.97-139.

33. WAND M. On the recursive specification of data types. - Lec.Notes in Comp.Sci.- Berlin-Heidelberg-New York: Springer-Verlag,1974,v.25.- 240 p.

34. SMYTH M.B., PLOTKIN G.D. The Category-Theoretic Solution of recursive domain equations.- SIAM J.Computation,1982,v.11,N 4, p.761-783.

35. SMYTH M.B. Computability in categories.- Lec.Notes in Comp.Sci.- Berlin-Heidelberg-New York: Springer-Verlag,1980,v.85, p.609-620.

36. WAND M. Fixed-Point constructions in order enriched categories.-Theoretical Comp.Science,1979,N 8,p.13-30.

37. Данные в языках программирования: Темат.сб./Пер. с англ.. - М.: Мир, 1982. - 328 с.

38. АГАФОНОВ В.Н. Типы и абстракция данных в языках программирования (обзор). -В кн.: Данные в языках программирования. М., 1982, с. 265-327.

39. ГОНЧАРОВ С.С. Матрица как абстрактный тип данных. - Настоящий сборник, с. 75-86.

40. ЛИСКОВ Б., ЗИЛЛЕС С. Методы модификации, используемые для абстракции данных. - В кн.: Данные в языках программирования.Мир. 1982, с. 91-122.

41. ВЕГНЕР П. Программирование на языке АДА. - М.: Мир, 1983, - 238 с.

42. БРАУН П. Микропроцессоры и мобильность программного обеспечения. -М.: Мир, 1977. - 254 с.

43. Базисный РЕФАЛ и его реализация на вычислительных машинах (методические рекомендации): М.: ЦНИПИАСС Госстроя СССР, 1977. - 300 с.

44. МАУРЕР У. Введение в программирование на языке ЛИСП.-М.: Мир, 1976. - 104 с.

45. РЕДЬКО В.Н. Композиционная технология программирования. - Киев: Знание, 1981. - 24 с.

46. ЙОДАН Э. Структурное проектирование и конструирование программ. - М.: Мир, 1979. - 416 с.

47. ВЕЛЬБИЦКИЙ И.В., ХОДАКОВСКИЙ В.Н., ШОЛМОВ Л.И. Технологический комплекс производства на машинах ЕС-ЭВМ и БЭСМ М БЭСМ-6. - М.: Статистика, 1980. - 262 с.

48. МАЙЕРС Г. Надежность программного обеспечения. -М.: Мир, 1980. - 360 с.

49. БОЭМ Б. и др. Характеристики качества программного обеспечения. - М.: Мир, 1981. - 206 с.

50. ЗЕЛКОВЕЦ М., А.ШОУ, Дж.ГЭННОН. Принципы разработки программного обеспечения. -М.: Мир, 1982. - 368 с.

51. РЕДЬКО В.Н. Композиция программ и композиционное программирование. - Программирование, 1978, № 5, с. 3-24.

52. РЕДЬКО В.Н. Основание композиционного программирования. - Программирование. 1979, № 3, с. 3-13.

53. MANNA Z., WALDINGER R.I. Synthesis: Dreams-Programs.-IEEE Trans.on Software Eng.,1979,v.SE-5,N 4,p.294-328.

54. ДАРЛИНГТОН Дж. Синтез нескольких алгоритмов сортировки.-В кн.: Кибернетический сборник, 1981, № 18, с. 141-176.

55. ERSHOV A.P. The transformational machine, theme and variations.- Lect.Notes in Comp.Science,1981,v.118,p.16-32.

56. CONSTABLE P. Constructive mathematics and automatic prog - ram writes.- In: Proc.IFIP Congress (Lynbljana).- Amsterdam,1971, p.229-233.

57. НЕПЕЙВОДА Н.Н. О построении правильных программ. -Вопросы кибернетики, 1978, № 46, с. 81-113.

58. ТЫУГУ Э.Х. Синтез программ (обзор). -Всесоюзн. конф. по методам мат. логики в проблемах искусственного интеллекта и систематическое программирование. Тез. докл. 4.2. Вильнюс, 1980, с. 70-89.

59. GOAD Chr.A. Computational uses of the manipulation of formal proofs.- Stanford: 1980. - 122 p. (Report No.STAN-CS-80-819/ Stanford University).

60. НЕПЕЙВОДА Н.Н., СВИРИДЕНКО Д.И. К теории синтеза программ. -Тр. Института математики СО АН СССР, 1982, т.2, с.159-175.

61. БАРЗДИНЬ Я.М. Замечания о синтезе программ по историям их работы. -Уч зап.Латвиск. ун-та, 1974, т. 210, с. 145-151.

62. БАРЗДИНЬ Я.М. Индуктивный вывод автоматов, функций, примеров. - In: Proc.International Cong.Math. (Vancouver,1974), 1975, v.2, p.445-460.

63. BIERMANN A.W., KRISHNASWAMY R. Constructing programs from examples computation.- IEEE Trans.Software Eng.,1976,v.SE-2, N 5, p.141-153.

64. HARDY S. Synthesis of USP-functions from examples.- In: Advance Papers of the 4th International Joint Conf.on Artifical Intelligence.- Cambridge: M.I.T.Press,1975,p.240-245.

65. ЕРШОВ А.П., ЗВЕНИГОРОДСКИЙ Г.А., ПЕРВИН Ю.Н. Школьная информатика (концепции, состояние, перспективы).-Новосибирск: Б.и., 1979. - 51 с. (Препринт/ВЦ СО АН СССР; 152).

66. ЯНЕНКО Н.Н. Тенденция развития современной математики. -В кн.: Математические проблемы научного познания.Новосибирск, 1977, с. 64-72.

67.ЯНЕНКО Н.Н., КАРНАУЧУК В.И., КОНОВАЛОВ А.И. Проблема математической технологии. - В кн.: Численные методы механики сплошной среды, т.8, № 3, Новосибирск, 1977, с. 129-157.

68. ДЕЙКСТРА Э. Дисциплина программирования. -М.: Мир, 1978. - 274 с.

Поступила в ред.-изд.отд.
13 апреля 1983 года