ЛОГИЧЕСКИЕ МЕТОДЫ В ПРОГРАММИРОВАНИИ
(Вычислительные системы)
1990 год Выпуск 133

УДК 519.68:519.685

ПРОЕКТ СИГМА. ЦЕЛИ И ЗАДАЧИ

Д.И.Свириденко

В в е д е н и е

Отличительной особенностью современного этапа развития вычислительной техники является стремительное расширение сферы ее применения, в связи с чем постоянно ощущается потребность в новых компьютерных методах и средствах решения задач. Кроме того, возрастают и требования к качеству решения задач, поскольку ошибки в их решении могут иметь далеко идущие последствия, соизмеримые в отдельных случаях с ущербом, причиняемым природными катастрофами. Поиск методов и средств, гарантирующих правильность решений осложняется тем обстоятельством, что непрерывно увеличивается масштабность и сложность решаемых задач. Зачастую решение таких задач оказывается под силу только целым коллективам специалистов, а это, в свою очередь, также требует создания соответствующих методов и средств. Здесь имеются, по крайней мере, три подхода:

1) использовать для решения новых задач уже имеющиеся методы и средства, не меняя при этом содержания фундаментальных понятий и идей, составляющих их концептуальную основу;

2) для выделенного класса задач разработать методы,средства и даже технологию ЭВМ на принципиально новой концептуальной и понятийной основе, не совместимой с имеющимися;

3) разработать новую концепцию программирования, адекватную рассматриваемому классу задач и совместимую с имеющимися.

Реализации последней возможности применительно к задачам логической природы и посвящен проект СИГМА.

Логические задачи в последнее время приобретают все большее значение. Примерами таких задач могут служить задачи логического моделирования, прототипирования систем, многие задачи искусственного интеллекта, в частности, большинство задач, решаемых экспертными системами, и т.д.. Для нас весьма важным обстоятельством является то, что большая часть этих задач может быть естественно сформулирована как задачи установления истинности логических утверждений. При этом нас будет интересовать не просто факт установления истинности, а указание тех объектов, на которых эта истинность достигается (такая истинность логических утверждений носит название конструктивной).

К настоящему времени известно несколько подходов к решению указанного выше класса задач. Наибольшую популярность приобрел подход, развиваемый в рамках так называемого логического программирования [12,14,34,59,62]. Отличительной особенностью данной концепции программирования является ее теоретико-доказательное происхождение. Другими словами, речь идет о том, что задача установления конструктивной истинности логического утверждения в данной концепции решается в терминах его выводимости. Однако в математической логике для установления истинности широко используется другой, теоретико-модельный подход. Именно этот подход положен в основу концепции семантического программирования, точнее ее конкретного варианта, называемого Σ-программированием [3-6,26-28,49,56,57]. Основные методологические положения Σ-программирования могут быть сформулированы следующим образом:

а) логическая задача в Σ-программировании мыслится как тройка $(\mathbb{A}, q, I)$, где $\mathbb{A}$ - описание проблемной области, q - утверждение, конструктивная выполнимость которого в $\mathbb{A}$ нас интересует, I - дополнительная информация нелогической природы (системные требования, ресурсные ограничения и т.п.);

б) проблемная область $\mathbb{A}$ в Σ-программировании формализуется как многосортная модель $\mathcal{M}$ языка исчисления предикатов 1-го порядка;

в) в качестве логического утверждения, выполнимость которого нас интересует, выступает формула специального вида (так называемая Σ-формула) многосортного исчисления предикатов 1-го порядка сигнатуры модели $\mathcal{M}$; под ее истинностью понимается истинность по Тарскому на модели $\mathcal{M}$, формально представляющей проблемную область $\mathbb{A}$;

г) в основу формального описания модели $\mathcal{M}$ кладется механизм ее элементарной определимости в другой модели $\mathcal{N}$,выступающей в качестве базовой и относительно которой предполагается, что она уже задана неким конструктивным образом;

д) возможность алгоритмического установления конструктивной выполнимости утверждения q в модели $\mathcal{M}$ гарантируется тем, что при ее определимости в базовой модели $\mathcal{N}$ используются Σ-формулы.

Важным обстоятельством является то, что Σ-программирование не отрицает другие концепции программирования, а, наоборот, позволяет разумно с ними сосуществовать и использовать их сильные стороны. Это достигается с помощью базисной модели.Так, например, нужные численные расчеты могут осуществляться, скажем, в рамках традиционного, фон неймановского программирования, а логические преобразования естественно могут быть реализованы в Σ-программировании. Роль сигнатуры базисной модели при этом - осуществлять интерфейс между этими видами вычислений. Именно данное обстоятельство позволяет рассматривать

Σ-программирование как естественное расширение уже имеющихся концепций. Прежде всего это относится к фон неймановскому программированию.

К настоящему времени в основном закончена теоретическая проработка вопросов, связанных с моделью вычислений, составляющей математическую основу Σ-программирования. На повестке дня - осуществление фундаментальных и прикладных исследований, связанных с дальнейшим развитием концепции Σ-программирования применительно к созданию экспериментальной системы Σ-программирования СИГМА, ориентированной на решение задач установления истинности логических утверждений.

Реализации этой цели и посвящен проект СИГМА.

§1. Традиционная схема решения задач

До сих пор наиболее популярными подходами к решению задач в программировании являются два подхода, известные как подходы "сверху-вниз" и "снизу-вверх" (в реальной процедуре решения задачи, как правило, присутствуют элементы того и другого). Подход "сверху-вниз" характерен тем, что при переходе от поставленной задачи к "машинному" представлению алгоритма ее решения промежуточные конструкции и дополнительные понятия, с помощью которых и осуществляется данный переход, создаются по ходу решения (рис.1).

Подход же "снизу-вверх" имеет как бы противоположное "направление", поскольку при переходе от постановки задачи к программе (представлению алгоритма решения в "машинных" терминах) наборы дополнительных конструкций, каждая из которых для своей реализации может потребовать нескольких "машинных" команд, создаются предварительно.

Поскольку попытка реализовать традиционную схему непосредственно, "в один прием", зачастую приводила к ошибкам в программах и делала процесс конструирования программ неуправляе-

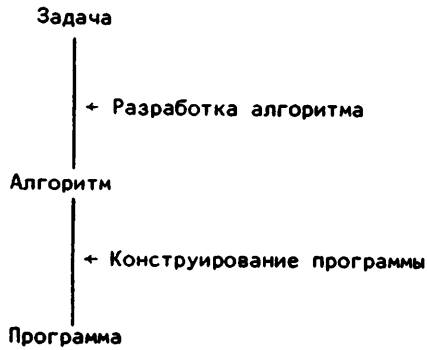

Рис. 1

мым, то более 20-ти лет тому назад была выдвинута идея промышленного подхода к изготовлению программ, оформившаяся впоследствии в так называемую инженерную точку зрения на программирование [1,8,9,17-19,24,73,77]. Согласно данной точке зрения, программа - это вид промышленного изделия и, следовательно, требует для своего изготовления специальной технологии программирования (по аналогии с проектированием сложных механизмов). Как отмечается в [9], наиболее характерной особенностью инженерного подхода является его эмпиричность. Особенно ярко эта особенность проявляется в проблеме правильности программ: заключение о корректности программы выносится только при ее испытании (отладке), поскольку при инженерном подходе вначале предлагается создать программу и лишь затем экспериментально продемонстрировать ее соответствие решаемой задаче. Заметим, что хотя процесс отладки является одним из самых трудоемких и дорогих этапов конструирования программ (кстати, не гарантирующего при этом отсутствия ошибок), он и сейчас остается наиболее распространенным способом подтверждения правильности программ.

Основная организационная идея инженерного подхода заключается в декомпозировании всего процесса построения программ на последовательность более мелких, функционально осмысленных этапов, делающих весь процесс более управляемым и верифицируемым. При этом практически все конкретные варианты подобного разбиения, получившие общее название "жизненный цикл программы", включают как промежуточный этап уточненную постановку за-

дачи (состоящей, в том числе, и из системных требований к будущей программе), что позволяет разбить весь процесс на два больших этапа: формулировка задачи и определение системных требований и собственно сам процесс программирования, состоящий из разработки алгоритма, написания кодов программы и ее тестирования. Однако и эти два этапа являются слишком большими. Поэтому усилия многих исследователей были направлены на выявление возможностей дальнейшей декомпозиции этих этапов (в особенности этапа программирования) на более мелкие шаги (рис.2).

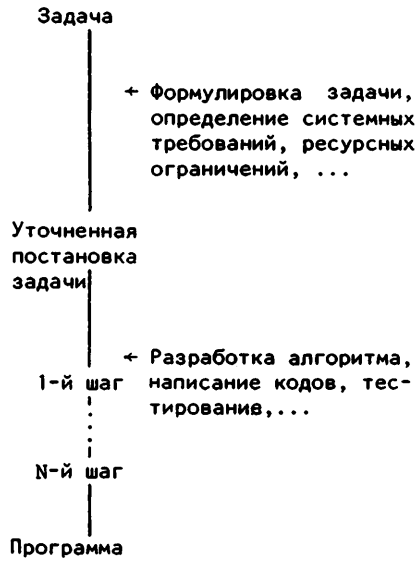

Рис. 2

Одной из наиболее популярных идеологий в данном направлении является идеология "пошагового уточнения" (см., например, [77]). Здесь уже просматривается определенная дисциплина конструирования программ. Однако в предлагаемом подходе переход от одной промежуточной стадии к другой осуществляется интуитивно, что имеет свои существенные недостатки: чтобы быть уверенным в корректности итоговой программы, необходимо осуществлять верификацию каждого шага. Кроме того, предлагаемая дисциплина конструирования программ носит скорее характер искусства, нежели характер производственного процесса.

Другая важная идея инженерного подхода - это идея интегрированных технологических систем и сред (и-систем, и-сред), которая смещает центр внимания с разработки изолированных ин-

струментальных средств поддержки деятельности программиста на создание систем и сред, поддерживающих работу программиста как можно на большем числе этапов конструирования программ: постановки задачи, определении системных требований, кодирования, верификации, отладки, модификации и сопровождении программ. Однако существующие и-системы и и-среды не покрывают все виды программистской деятельности [71]. Слабым местом таких систем и сред до сих пор остается этап получения точной формулировки проблемы - то, что называют спецификацией задачи. Но именно неполнота, неточность, ошибочность спецификаций задач привносят, как нетрудно понять, самые большие трудности в процесс разработки программного обеспечения и существенно увеличивают связанные с этим расходы.

Все эти обстоятельства послужили причиной активного поиска новых подходов (особенно в последнее десятилетие), в том числе и подходов, уточняющих и развивающих инженерную точку зрения, но базирующихся уже на принципиально иной точке зрения на природу программирования и, следовательно, на то, какие нужны программы и как их конструировать. В последнее время все более привлекательной становится идея конструирования программ правильных по построению. Одной из концепций, в которой эта идея реализуется наиболее последовательно, является доказательное программирование [1,6-9,17-19,22,24,25,39,43,48,65,66,71, 73, 75].

§2. Доказательное программирование

Согласно исходным положениям этого подхода, процесс построения программы обязательно должен включать в себя (в явной или неявной форме) доказательство ее правильности в том смысле, как "правильность" и "доказательство" понимаются в математике. Таким образом, деятельность программиста при доказательном программировании приобретает не столько инженерный, сколь-

ко прикладной математический характер. Однако реализация кон - цепции доказательного программирования предъявляет довольно жесткие требования как к используемым средствам, так и к методам конструирования программ. Прежде всего, речь идет о необ - ходимости уметь записывать постановки задач в точных, формальных терминах (именно такие формулировки мы и будем в последующем называть спецификациями задач), что,в свою очередь, требует формального языка. Во-вторых, доказательное программирова - ние предполагает наличие таких методов построения программ,которые реально обеспечивают реализацию принципа "правильность по построению". И наконец, инструментарий, который должен поддерживать процесс доказательного конструирования программ,обязан быть концептуально единым и носить по необходимости интегрированный характер.

Проблема спецификации является сейчас одной из централь - ных как в теоретическом, так и в практическом программировании. В ходе ее решения постепенно сформировались основные требования к "хорошим" методам спецификаций, в которых нашел свое отражение негативный опыт применения неформальных средств при формулировках и решении задач. Так, например, "хорошие" методы должны:

- требовать точной формальной постановки решаемой проблемы;

- обеспечивать возможность точной формулировки и убедительного доказательства свойств спецификаций;

- допускать возможность машинной поддержки этапов конст - руирования и верификации спецификаций;

- требовать максимальной независимости спецификационных понятий от реализационных понятий;

- поддерживать такое иерархическое и модульное построение спецификаций, которое было бы адекватно целям модульного и иерархического построения программ;

- поддерживать концепцию повторной применимости спецификаций;

- иметь средства для исполнения спецификаций задач (прототипирование систем);

- быть понятными и удобными для широкого круга пользователей.

Нетрудно понять, что большая часть данных требований может быть удовлетворена, если в качестве прообразов языка спецификаций рассматривать языки математической логики, которые, как правило, являются в высшей степени декларативными языками. Сразу же заметим, что помимо выбора логического языка спецификаций желательно одновременно указать также и ту логику,на которой будут основываться рассуждения о спецификациях. Однако при таком выборе наибольшую сложность представляет требование исполнимости спецификаций.

Формальные методы спецификаций наиболее активно стали развиваться в 70-е годы. Самый известный из них базируется на идеях денотационной семантики и известен под названием VDM (Vienna Development Method [62,63,75]). Другой метод, так называемый метод алгебраических спецификаций, основывается на аксиоматической теории многосортных алгебр [45-47,53,60,69].Третий метод, метод функциональных спецификаций [13,22,23,31, 35, 54], известен в программировании уже давно, начиная с создания языка ЛИСП [13]. Его можно рассматривать как определенную разновидность алгебраического метода. Наконец, в 70-е годы получил широкое распространение метод, описанный на хорновой логике [12,14,34,59,62] (широко известен язык ПРОЛОГ [11], базирующийся на этой логике).

Указанные методы представляют собой два принципиально разных подхода к формализации спецификаций. Так, например,VDM естественно отнести к семантически-ориентированному подходу, основанному на описании типов данных в терминах многосортных мо-

делей и алгебр (к этому же подходу относится и метод спецификаций настоящего проекта). Большинство же остальных методов представляют собой вариации аксиоматически-ориентированного подхода, основанного на характеристике свойств типов данных в какой-нибудь аксиоматической теории, или, что эквивалентно, в классе моделей. Модель типа данных в рамках семантически-ориентированного подхода описывается при помощи предопределенных (встроенных, базовых) типов (множества, списки, n-кортежи, последовательности и т.п.). Свойства требуемых операций и отношений характеризуются либо условиями на вход-выход, либо условиями, которые должны выполняться между объектами. При аксиоматически-ориентированном подходе объекты типа данных определяются неявно - с помощью операций, их создающих, или отношений между ними. Свойства операций, конструирующих объекты или применяемых к ним, а также отношений описываются в виде аксиом.

Оба подхода были успешно использованы при спецификации нескольких нетривиальных проблем. Так, например, VDM был применен для спецификации языка АДА (известны и другие его применения). При оценке этих применений были отмечены многие положительные стороны VDM. Но известны и его недостатки:

- язык спецификаций VDM не имеет средств для модуляризации спецификаций и средств проектирования, поддерживающих конструирование иерархических и структурных спецификаций;

- отсутствуют компьютерные инструментальные средства для поддержки процесса специфицирования;

- язык спецификаций VDM не имеет удовлетворительной математической семантики, что сказывается на понимании VDM-спецификаций и затрудняет использование теоретико-доказательных методов при рассуждениях о них;

- метод не поддерживает более мощную, чем уравнения для областей, и полезную во многих отношениях методологию абстрактных типов данных.

Кроме того, VDM не имеет средств для спецификации парал - лелизма. Известны также трудности при модифицируемости и пов - торной применимости VDM-спецификаций. В целом можно сделать вывод, что VDM в его изначальном виде был более ориентирован на поддержку получения обычных, последовательных программ из спецификаций, нежели на поддержку собственно этапа конструирова - ния спецификаций. Однако в последнее время появились исследо - вания, в которых большая часть этих недостатков VDM уже устранена [67,68].

Известны успешные применения и аксиоматически-ориенти - рованного подхода. Так, например, метод алгебраических специ - фикаций был использован для спецификации иерархической системы файлов типа UNIX, для формального описания компилятора языка ПАСКАЛЬ и т.д. Необходимо отметить удовлетворительную разви - тость теоретического базиса аксиоматически-ориентированных методов и большую активность исследований в данном направлении (особенно в последние годы).

Методы доказательного программирования можно (хотя и весьма условно) разделить на два вида: содержательные и формальные. Их отличие друг от друга заключается в том, что формальные методы ориентируются на формальные доказательства. Первые же методы, как правило, являются хотя и математизированной, но содержательной версией концепции "пошагового уточнения". К ним можно отнести метод слабейшего предусловия, предложенный Дейкстрой [7,8], метод инференциального программирования Скотта [73]. К содержательным методам необходимо отнести VDM и его различные модификации, а также методы, по своей идеологии близко примыкающие к VDM (например, методы проектов METASOFT [40] и PROGRESS [71]). К таким методам относится и метод, развиваемый в проекте СИГМА.

Формальные методы доказательного программирования подразделяются на трансформационные, дедуктивные и индуктивные. Ос-

новная идея трансформационного программирования заключается в формализации метода "пошагового уточнения" посредством использования специальных, семантически корректных формальных правил (трансформаций) перехода от одной конструкции к последующей, начиная с формальных спецификаций задачи [10,17,21,37,41,42, 48, 50,66,70]. Наиболее развитым трансформационный метод представляется в Мюнхенском проекте CIP (Computer-aided, Intuition-guided Programming) [1,37,66,70]. Работы по трансформационному программированию ведутся также в СССР (Новосибирск, ВЦ СО АН СССР) [10,21,48].

Дедуктивные методы доказательного программирования базируются на основной парадигме конструктивной математики [38], согласно которой объект, о котором идет речь в теореме существования, должен быть эффективно извлечен (хотя бы в принципе) из конструктивного доказательства этой теоремы. В качестве таких объектов в дедуктивном подходе рассматриваются термы-программы [2,15-20,36,39,43,44,55,58,63,72]. Известны конкретные реализации данного подхода (см.,например, обзор в [17,28,30]). Интересными и перспективными разновидностями дедуктивного подхода являются метод, предложенный Гоадом [52], и методы, основу которых составляет подход "формулы-как-типы" [39,61],в частности, интуиционистская теория типов [64,65]. Из практических методов, реализуемых в СССР, необходимо, прежде всего,отметить проект ПРИЗ (Таллинн,ИК АН ЭССР) [2,16,74]. Заметим, что системы хорнова программирования также можно рассматривать как практические реализации дедуктивного синтеза (термов).

Логико-математическую основу индуктивных методов доказательного программирования составляют индуктивные логики [32, 33]. Например, метод, развиваемый в Рижской школе (ВЦ Латвийского университета), позволяет синтезировать программы по конечным наборам историй их вычислений [29,51].

§3. Цели и задачи проекта СИГМА

Прежде чем формулировать цели и задачи проекта,уточним наше понимание некоторых терминов и понятий.

Алгоритмическое решение задачи предполагает обязательное указание либо модели вычислений (такая модель носит,естественно, абстрактный характер, поскольку в данном случае решение мыслится как абстрактный алгоритм или ищется с помощью подобного алгоритма), либо языка программирования, т.е. языка проектирования классов вычислений, порождаемых конкретным вычислительным устройством. Если решение задачи осуществляется систематическим образом (методом), то будем говорить о стиле программирования. Саму же модель вычислений вместе с адекватными этой модели стилями программирования и будем называть концепцией программирования. Примером модели вычислений может служить "машина фон Неймана" [76], которая до сих пор остается основной моделью в программировании. Метод программирования "сверху-вниз" без привлечения оператора go to - это уже определенный стиль программирования. Вместе они составляют один из вариантов концепции структурного программирования.

Итак, разработка новой концепции программирования предполагает, прежде всего, разработку модели вычислений и адекватного ей стиля (или целого семейства стилей) программирования. В свою очередь, выбор или поиск модели вычислений и соответствующего стиля, как следует из ранее сказанного, во-первых определяется точкой зрения на природу программирования, а во-вторых, тем классом задач, на решение которых ориентирована создаваемая концепция.

Как уже было объявлено ранее, проект СИГМА придерживается точки зрения на природу программирования, принятой в доказательном программировании (см.выше), поскольку теоретической и методологической основой научно-исследовательского проекта СИГМА является логико-математическая концепция семантического

программирования, уже несколько лет разрабатываемая в отделе математической логики Института математики СО АН СССР [3-6,26-28,49,56,57]. Данная концепция ориентирована, как отмечалось во введении, на компьютерное решение таких задач логической природы, характерной чертой которых является то, что они естественно могут быть сформулированы как задачи установления конструктивной истинности логических утверждений. Таким образом, основная цель проекта СИГМА может быть окончательно сформулирована следующим образом: в рамках концепции семантического программирования разработать интегрированную среду, поддерживающую все этапы компьютерного решения логических задач, включая процесс формализации их спецификаций, верификацию логических и семантических свойств этих спецификаций, исполнение спецификаций, и если необходимо, то и доказательное кодирование соответствующих спецификациям программ (в терминах некоторого фиксированного языка программирования).

Формальные средства спецификации задач настоящего проекта примыкают к семантически-ориентированному подходу, о котором шла речь выше. Однако в отличие от Венского метода (VDM), который отталкивается от идеологии денотационной семантики, основным логико-математическим механизмом средств спецификации проекта СИГМА является механизм элементарной определимости (т.е. определения одной многосортной модели в другой, используя для этих целей язык исчисления предикатов 1-го порядка). Заметим, что наряду с основным механизмом спецификации в проекте СИГМА предполагается в качестве вспомогательных использовать и другие логико-математические механизмы, в том числе и примыкающие к аксиоматически-ориентированному подходу. Естественно, что все эти механизмы должны быть согласованы между собой. Таким образом, предполагаемая система семантического программирования СИГМА видится интегрированной как бы в двух измерениях:

а) она является интегрированной с точки зрения поддержки, по возможности, всех этапов компьютерного решения задач, начиная с этапа спецификаций, и в этом смысле поддерживает как "программирование в большом", так и "программирование в малом";

б) она является интегрированной с точки зрения привлечения различных логико-математических средств спецификации задач (в рамках, естественно, концепции семантического программирования) и средств рассуждения о спецификациях.

Интегрированность системы СИГМА в "первом измерении" достигается посредством, во-первых, концептуально единой языковой среды, в которой протекает вся деятельность по компьютерному решению задач, во-вторых, применения доказательного метода поиска решений задач, представляющего собой один из вариантов математизации метода "пошагового уточнения", в-третьих,использования адекватного языку и методу инструментария поддержки процесса компьютерного решения задач и, наконец,наличия специальной документации, включающей различные руководства и обучающий материал по языку, методу и инструментам.

Что касается "второй координаты" интегральности системы, то при выборе средств спецификации задач решающими оказались следующие соображения - данные средства должны:

- позволять использовать не только семантически-ориентированные, но и аксиоматически-ориентированные средства спецификации;

- позволять использовать принципы декомпозиции и модульности уже на этапе спецификации;

- иметь развитые средства конструирования модифицируемых и повторно используемых спецификаций;

- позволять формулировать семантические и аксиоматические свойства конструируемых сущностей (непротиворечивость,полнота, эквивалентность и т.д.) и рассуждать о них;

- иметь модель вычислений, с помощью которой возможно ис - полнять спецификации задач определенного вида и доказывать их свойства.

Охарактеризуем теперь кратко главные составляющие проек - та: язык, метод и инструментарий.

Языковая среда системы СИГМА (СИГМА-язык), в которой, как предполагается, будет протекать деятельность пользователя системы по решению задач, состоит из двух главных частей (более подробно языковые средства описываются в статье "СИГМА-язык" [78]).

Первая часть - языки спецификаций - представляет собой языковые средства, ориентированные непосредственно на спецификацию логических задач. О том, какими свойствами должна обла - дать эта часть, уже шла речь выше. Заметим, что главную роль в СИГМА-языке играют исполнимые спецификации, имеющие конструк - тивную семантику. Эти конструкции можно рассматривать как конкретный вариант реализации понятия Σ^+-программы [5].

Вторая часть языка системы СИГМА, называемая языком заго - ловков, ориентирована на описание отношений между спецификациями и позволяет тем самым, конструировать как иерархические спецификации, характерные для сложных и масштабных задач, так и описывать историю создания объекта языка. Эти отношения делятся на отношения использования и проектировочные отношения. Все отношения фиксируются в так называемых заголовках спецификаций. Основными механизмами, определяющими отношения использо - вания, являются механизмы элементарной определимости, параметризации и импорта. Не менее важным средством построения спецификаций являются механизмы, позволяющие создавать новые спецификации из уже существующих.

Проектировочные отношения - это отношения, возникающие между объектами в процессе проектирования спецификаций. Эта часть языка заголовков тесно связана с выбранным методом формализа -

ции исходных требований, доказательным методом конструирования спецификаций, их конструктивизацией и возможным превращением в коды программ инструментального языка. Важное место при описании проектировочных отношений занимают объекты, являющиеся обоснованиями "правильности" проектировочных решений.

Существенную часть системы СИГМА составляют те инструментальные языки, в терминах которых мыслится запись программных кодов, соответствующих сконструированным ранее спецификациям. Средствами этих же языков планируется и сама реализация системы СИГМА. Кроме того, некоторые конструкции инструментальных языков будут доступны и могут быть использованы уже на этапе спецификации задач. Основными такими инструментальными языками являются языки программирования "Си" и МОДУЛА-2 (базовые инструментальные языки).

Таким образом, СИГМА-язык представляет собой развитые языковые средства поддержки процесса:

- постановки задач;
- превращения неформальных постановок задач в формальные спецификации;
- превращения формальных спецификаций в исполняемые спецификации задач, т.е. средства поддержки процесса прототипирования;
- превращения исполнимых спецификаций в эффективно исполняемый программный код;
- оптимизации конструируемых исполнимых спецификаций и программ;
- модификации и повторной применимости имеющихся спецификаций;
- рассуждений о спецификациях и их свойствах, убедительного доказательства правильности принимаемых проектировочных решений.

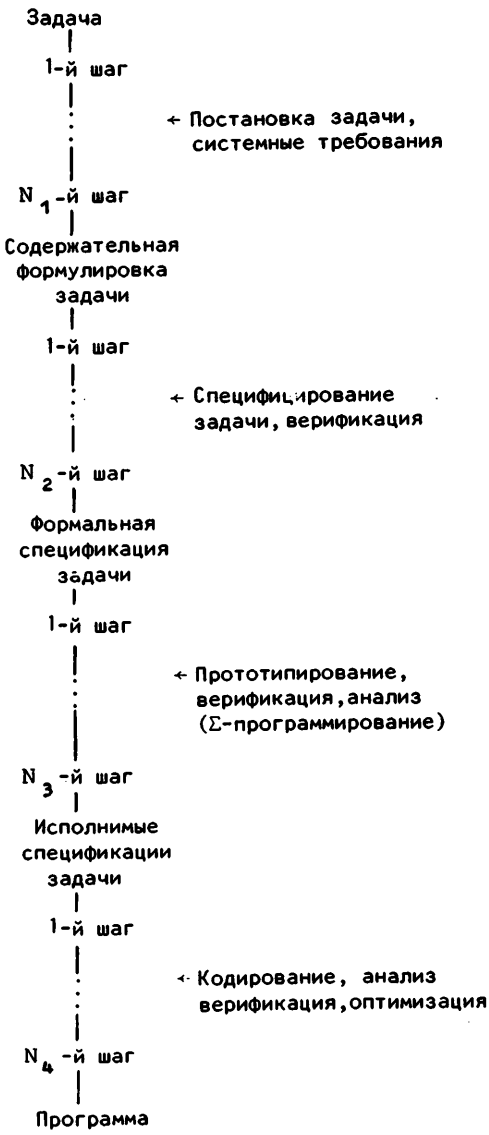

Рис. 3

Как уже говорилось выше, метод решения задач, принятый в проекте СИГМА (СИГМА-метод), представляет собой математизированную версию метода "пошагового уточнения" (рис.3). Весь процесс решения задачи разбивается на четыре крупных этапа.

Первый этап - этап содержательной постановки задачи (с учетом системных требований на решение). В определенном смысле этот этап соответствует этапу подготовки Технического Задания в существующих технологиях программирования. Данный этап мыслится как диалог между заказчиком (постановщиком задачи) и исполнителем (спецификатором задачи). Их обязанность в соответствии с принятым в

системе сценарием - отвечать на определенные вопросы (как правило, это должен делать заказчик) и анализировать ответы (обязанность исполнителя). Итогом их деятельности должна стать совокупность объектов языка, связанных между собой отношениями как использования, так и проектировочными. Эта совокупность, с одной стороны,позволит автоматически создать специального вида документы (объединяемые под общим названием "техническое задание"), а с другой - послужит основой последующего этапа.

Второй этап - это этап формализации содержательной постановки задачи (специфицирования),анализа полученной спецификации и ее декомпозиции на подзадачи (если это необходимо). На этом этапе используется СИГМА-язык спецификаций. Деятельность по специфицированию, как и на этапе постановки задачи, поддержи - вается соответствующими инструментами - редактором системы и подсистемой управления базой данных. Кроме того, на этом этапе используется также и анализатор языка спецификаций.

Спецификационная деятельность осуществляется спецификатором задачи (теперь он выступает в роли заказчика) и анализатором спецификаций (роль исполнителя). В функции спецификатора входит не только построение спецификаций, но и "доказательст - во" соответствия полученных формальных описаний исходной по - становке задачи. Если задача носит масштабный характер, то дополнительно на спецификатора возлагается обязанность декомпо - зиции формальной спецификации всей задачи на спецификации ее подзадач, решение которых - прерогатива исполнителей. Поскольку последние должны, по крайней мере, понимать предложенные им спецификации, то рекомендуется и им принимать участие в процессе специфицирования, взяв на себя роль анализаторов получаемых конструкций, в том числе и обоснований правильности проектировочных решений. Отметим, что термины "заказчик", "спецификатор", "исполнитель" носят чисто ролевой характер; все эти роли могут "исполняться" и одним лицом.

На третьем этапе формальные спецификации должны быть подвергнуты конструктивизации, т.е. превращены в исполнимые. Другими словами, должен быть создан прототип будущего решения задачи, мыслимого в виде программы или системы. Именно на этом этапе, по существу, начинает работать методология доказательного программирования - каждый шаг перехода от спецификации к ее исполнимому варианту должен быть настолько убедительно обоснован, насколько убедительно обоснованными для нас являются математические рассуждения. Роль языка спецификаций инструментального окружения системы СИГМА - в максимальной степени способ - ствовать этому. Так, скажем, целесообразно хранить историю получения как самих формальных спецификаций, так и исполнимого варианта; также должны храниться проектировочные решения и их обоснования.

После того как получены исполнимые спецификации задачи, они должны быть проверены на предмет их соответствия поставленной задаче. Такой анализ по своему содержанию напоминает этап тестирования и отладки обычных программ, только носит более математический характер. Заметим, что, вообще говоря, не исключ - чена возможность того, что полученные исполнимые спецификации могут быть признаны вполне удовлетворительными с точки зрения эффективности их исполнения. В этом случае они могут рассмат - риваться как готовые программы (Σ-программы). Если же требуется более эффективное исполнение, то необходимо осуществить следующий этап - получение программных кодов в базовом инструментальном языке. И вновь этот этап должен носить доказательный характер - каждый шаг в направлении программы должен быть убедительно обоснован.

Несколько слов об инструментарии системы СИГМА. В его состав входят редактор текстов (с некоторыми дополнительными возможностями), СУБД, системы программирования на инструменталь-

ных языках. Но главной компонентой конечно же является транс - лятор языка, позволяющий исполнять Σ-спецификации задач.

§4. Этапы реализации проекта

В заключение несколько слов о последовательности работ по осуществлению проекта. Целью первого этапа разработки проекта является,главным образом, создание методологических и теоретических средств, описывающих процесс специфицирования, конструктивизации спецификаций, т.е. превращения их в исполнимые кодирования программ, а также верификации и анализа возникающих при этом конструкций. На этом же этапе должен быть спроектирован язык СИГМА. Планируется в его терминах осуществить специ - фикацию некоторых реальных задач.

Круг решаемых задач на втором этапе должен быть сосредо - точен на главной цели проекта: разработке экспериментальной системы семантического программирования СИГМА.Сюда входят разработка функциональной и системной архитектуры системы,ее программная и методическая реализация и, наконец, апробация на реальных задачах.

Поскольку одним из основных классов задач, на решение которых ориентирована система СИГМА,видятся задачи искусственного интеллекта и, в частности, задачи,решаемые экспертными си - стемами, то на систему СИГМА можно смотреть и как на инструментальную систему искусственного интеллекта. С этой целью в состав системы СИГМА предусматривается ввести специальные функ - циональные компоненты - инструментальную подсистему конструи - рования лингвистических препроцессоров, инструментальную подсистему анализа логических данных (поиск закономерностей, обучение, индуктивный вывод и синтез, распознавание образов и т.п.), подсистему управления базами знаний. Разработка этих компонент - цель третьего этапа проекта.

Литература

1. БАУЭР Ф., БРЕЙ М. и др. На пути к языку широкого спектра для поддержки спецификации в разработке программ //Требования и спецификации в разработке программ. - М. - 1984. -С. 28-46.

2. ВОЛОЖ Б.Б. и др. Система ПРИЗ и исчисление высказываний //Кибернетика. - 1982. - № 6. -С. 65-70.

3. ГОНЧАРОВ С.С., СВИРИДЕНКО Д.И.Σ-программирование //Логико-математические проблемы МОЗ. - Новосибирск, 1985.-Вып.107: Вычислительные системы. - С.3-29.

4. Их же. Математические основы семантического программирования //Докл. АН СССР. - 1986. -Т. 289, № 6. -С.1324-1328.

5. Их же. Σ-программы и их семантики //Логические методы в программировании. - Новосибирск, 1987. - Вып. 120: Вычислительные системы. -С. 24-51.

6. ГОНЧАРОВ С.С., ЕРШОВ Ю.Л., СВИРИДЕНКО Д.И. Методологические аспекты семантического программирования //Научное знание: логика, понятия, структура. - Новосибирск: Наука. - 1987. - С. 154-184.

7. ГРИС Д. Наука программирования. - М.: Мир, 1984.-416 с.

8. ДЕЙКСТРА Э. Дисциплина программирования. - М.: Мир, 1978. - 280 с.

9. ЕРШОВ А.П. Научные основы доказательного программирования //Вестн. АН СССР. - 1984. - № 10. - С. 7-18.

10. КАСЬЯНОВ В.Н. Редуцированные преобразования программ //Трансляция и оптимизация программ. - Новосибирск. - 1983. - С. 86-98 (ВЦ СО АН СССР).

11. КЛОКСИН В., МЕЛЛИШ К. Программирование на языке ПРОЛОГ. - М.: Мир, 1987. - 336 с.

12. КОВАЛЬСКИЙ Р. Логическое программирование //Логическое программирование. -М.: Мир, 1988. - С. 134-166.

13. ЛАВРОВ С.С., СИЛОГАДЗЕ Г.Г. Автоматическая обработка данных. Язык ЛИСП и его реализация. - М.: Наука, 1978.

14. Логическое программирование /Отв. ред. В.Н.Агафонов. - М.: Мир, 1988. - 366 с.

15. МИНЦ Г.Б. Логические основы синтеза программ. - Таллинн,1982. - 42 с. -(Препринт/АН ЭССР, Ин-т кибернетики).

16. МИНЦ Г.Е., ТЫУГУ Э.Х. Обоснование структурного синтеза программ //Автоматический синтез программ. - Таллинн, 1983. - С. 52-60.

17. НЕПЕЙВОДА Н.Н. Анализ методов доказательного программирования в конструктивных логиках: Автореф. дис...д-ра физ.-мат. наук: 05.13.11-01.01.06. - Новосибирск, 1988.

18. НЕПЕЙВОДА Н.Н., СВИРИДЕНКО Д.И. Программирование с логической точки зрения. - Новосибирск. - 1981. - 50 с. - (Пре - принт/АН СССР. Сиб.отделение. Ин-т математики, Т-1).

19. Их же. Логическая точка зрения на программирование. - Новосибирск. - 1981. - 50 с. -(Препринт/АН СССР. Сиб.отделение. Ин-т математики. Т-2).

20. Их же. К теории синтеза программ //Труды Ин-та мате - матики/ АН СССР. Сиб.отделение. - 1982. -Т.2 /Математическая логика и теория алгоритмов. - С. 159-175.

21. НЕПОМНЯЩИЙ В.А., САБЕЛЬФЕЛЬД В.К. Трансформационный синтез корректных программ //Прикладная информатика. - 1986. - Вып. 2(11). -С. 19-38.

22. РЕДЬКО В.Н. Основания композиционного программирова - ния //Программирование. - 1979. - № 3. -С. 3-13.

23. РЕДЬКО В.Н., НИКИТЧЕНКО Н.С. Композиционные аспекты программологии. 1 //Кибернетика. - 1987. -№ 5. -С. 49-56.

24. СВИРИДЕНКО Д.И. О природе программирования //Матема - тическое обеспечение вычислительных систем из микроЭВМ. - Новосибирск, 1983. - Вып.96: Вычислительные системы. -С.51-74.

25. Его же. Предпосылки логического подхода к программи - рованию //Философские основания научной теории. - Новосибирск, Наука, 1985. - С. 91-107.

26. Его же. Проектирование Σ-программ. Постановка проблемы //Методы анализа данных. - Новосибирск, 1985. - Вып. 111: Вычислительные системы. - С. 108-127.

27. Его же. Проектирование Σ-программ. Σ-оцениваемость // Логические вопросы теории типов данных. - Новосибирск, 1986. - Вып. 114: Вычислительные системы. -С. 59-83.

28. Его же. Теория семантического программирования: Автореф. дис... д-ра физ.-мат.наук: 05.13.11. - Новосибирск, 1989. - 17 с.

29. Теория алгоритмов и программ, том 1,2,3. /Под ред. Я.М.Барздиня. - Рига, 1974,1975,1977 (Латв.госуниверситет).

30. ТЫУГУ Э.Х. Синтез программ (обзор) //Всесоюз.конф. по методам математической логики в проблемах искусственного ин - теллекта и семантическое программирование. Вильнюс, 1980 г. : Тез. докл. - Ч.2. - Вильнюс. - 1980. - С. 70-89.

31. ХЕНДЕРСОН П. Функциональное программирование. - М.: Мир, 1983. - 349 с.

32. Analogical and Inductive Inferece /K.P.Jantke (Ed.) //Lect. Notes in Al. - 1989. - N 397. - 338 p.

33. ANGLUIN D., SMITH C.H. Inductive inference: theory and methods //Computing Serveys. - 1983. -Vol. 15. -P. 237-269.

34. APT K.P., van EMDEN M.H. Contributions to the theory of logic programming //J.ASM. - 1982. -Vol. 29. -P. 841-863.

35. BACKUS J. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs //Comm.ACM. - 1978. -Vol. 21, N 8. -P. 613-641.

36. BATES J.L., CONSTABLE R.L. Proofs as programs // ACM Trans. Programming Languages and Systems. - 1985. -Vol. 7, N1. - P . 113-136.

37. BAUER F.L., MOLLER B., PARTSCH H., PEPPER P. Formal program construction by transformations - computer-aided, in - tuition-guided programming //IEEE Transactions on Software Engneering. - 1989. -Vol. 15, N 2. -P. 165-180.

38. BEESON M. Foundation of constructive mathematics // Mathematical studies. - Berlin: Springer Verlag. - 1985.

39. BEESON M. Proving programs and programming proofs // Logic, Methoodology and Philosophy of Science.VII. Studies in Logic avd Foundations of Mathematics. - Vol. 114. - Amsterdam: North-Holland, 1986. - P. 51-82.

40. BLIKLE A. MetaSoft Primer, Towards a Metalanguage for Applied Denotational Semantics //Lect. Notes in Comp. Sci. - 1987. N 288.

41. BURSTALL R.M., DARLNGTON J. Transformation for developing recursive programs //J.ACM. - 1977. - Vol. 24, N 1.-P.44-67.

42. CLARK K.L., DARLINGTON J. Algorithm classification trough synthsis //The Computer Journal. - 1980. - Vol. 23, N1. - P. 61-65.

43. CONSTABLE R.L. Constructive mathematics and automatic program writes //Proc.of IFIP'71. - Amsterdam: North-Holland, 1971. -P. 229-233.

44. CONSTABLE R.L., ZLATIN D.R. The type theory of PL/CV3 //Lect. Notes in Comp. Sci. - 1982. -Vol. 131. -P. 72-93.

45. DERSHOWITZ N., PLAISTED D.A. Equational programming. //Machine Intelligence. - 1987. -Vol. 11. -P. 21-56.

46. DERSHOWITZ N., SIVAKUMAR G. Solving goals in equational languages //Lect. Notes in Comp. Sci. - 1987.-N.308.-P.45-55.

47. EHRIG H., MAHR B. Fundamentals of algebraic specifi - cation. -Vol. 1: Equations and initial semantics //EATCS. - Springer Verlag. - 1985.

48. ERSHOV A.P. The transformational mashine, theme and variations //Lect. Notes in Comp. Sci. - 1982. -Vol. 118. - P. 16-32.

49. ERSHOV Y.L., GONCHAROV S.S., SVIRIDENKO D.I. Semantic foundations of programming //Lect. Notes in Comp.Sci. - 1987. - Vol. 278. -P. 116-122.

50. FEATHER M.S. A survey and classification of some program transformation approaches and techniques //Program Specification and Transformation. - Amsterdam: North-Holland, 1987. - P. 165-198.

51. FREIVALDS R., KINBER E.B., WIEHAGEN R. Inductive inference from good examples //Lect. Notes in AI.-1989.-N 397. - P. 1-17.

52. GOAD C.A. Computational uses of the manipulation of formal proofs //Stanford Univ.Report N STAND-CS-80-819. -1980. - 122 p.

53. GOGUEN J.A., THATCHER J.W., WAGNER E.G. An initial algebra approach to the specification, correctness, and imple - mentation of abstract data types //Current trends in program - ming methodology. Vol.4: Data structures. - Englewood Cliffs, NY; Prentice-Hall, 1978. -P. 80-114.

54. GORDON M.J., MILNER A.J., WADSWORTH C.P.Edinburgh LCE //Lect. Notes in Comp.Sci. - 1979. - Vol. 78. - 160 p.

55. GOTO S. Program synthesis from natural deduction proofs //Proc. of the 6th Internat.Joint Conf. on Articial Intelligence. - Amsterdam, 1979. - Vol. 2. -P. 339-341.

56. GONCHAROV S.S., SVIRIDENKO D.I. Theoretical aspects of Σ-programming //Lect.Notes in Comp.Sci. - 1986. - Vol.215. - P. 169-179.

57. GONCHAROV S.S., ERSHOV Y.L., SVIRIDENKO D.I. Semantic programming //Proc. 10th World Congress Information Processing 86. - Amsterdam. - 1986. -P. 1093-1100.

58. HAYASHI S. Extracting LISP-programs from proofs // PRIMS, Kyoto University. - 1983.-Vol.19, N 1. -P. 169-191.

59. HOGGER C. Introduction to logic programming. -London: Academic Press, 1984. - 278 p.

60. HOFFMANN C.M., O'DONNELL M.J. Programming with equa - tions //ACM Trans.Programming Languages and Systems. - 1987. - Vol. 4, N 1. -P. 83-112.

61. HOWARD W.A. The formulae-as-types notion of construc - tion //Festschrift on rhe occasion of Curry's 80th birthday. - N.Y., 1980. - P. 83-112.

62. LLOYD J.W. Foundation of logic programming. - Berlin: Springer-Verlag, 1984. - 124 p.

63. MANNA Z., WALDINGER R. Synthesis: dreams → programs// IEEE Trans. on Software Eng. - 1979. -Vol. SE-5, N4.-P.294-328.

64. MARTIN-LOF P. An intuitionistic theory of types: pre - dicative part //Logic Coll.'73.- Amsterdam: North-Holland,1975. -P. 73-119.

65. MARTIN-LOF P. Constructive mathematics and computer programming //Logic, Methodology and Philosophy of Science. VI. -Amsterdam: North-Holland. - 1982. -P. 153-179.

66. The Munich project CIP, vol.1: The wide spectrum lan - guage CIP-L//Lect.Notes in Comp.Sci. - 1985. -Vol. 183.-276 p.

67. MIDDELBURG C.A. VVSI:A language for structured VDM Specifications //Formal Aspects of Computing. - 1989. -Vol. 1,N 1. - P. 115-136.

68. NIELSEN M., HAVELUND K., WAGNER K.R., GEORGE Ch. The RAISE language, method and tools //Formal Aspects of Computing. - 1989. -Vol. 1, N 1. -P. 85-114.

69. O'DONNELL M.J.Equational logis as a programming language.- Cambridge, Mass: MIT Press, 1985. - 212 p.

70. PARTSCH H. The CIP transformation system //Program transformation and program environment. -Rept.Workshop NATO Adv.Res., Munich, 1983. - Berlin, a.o. - 1984. -P. 305-322.

71. PRIVARA I. PROGRESS - system podporujuci systematicku konstrukciu programu z formalnych specifikacii //Programovacie a databazove systemy, projekt G a projeckt Progress, Vyskumna praca 9, VUSEI-AR, Bratislava. - 1984. -P. 83-119.

72. SATO M.Towards a mathematical theory of program synthesis //Proc. Internat. Joint Conf. on Artificial Intelligence. - Tokyo. - 1979. -P. 757-762.

73. SCHERLIS W.L., SCOTT D.S. First stepts towards inferential programming //IFIP'83.- Amsterdam, North-Holland. - 1983. -P. 199-212.

74. TYUGU E.H., MATSKIN M.B., PENJAM J.E., FOMOIS P.U.NUT - an object-oriented language //Computer and Atificial Intelligence. - 1986. -Vol. 5, N 6. -P. 521-542.

75. VDM - A formal method at work/Bjoerner D.,Jones C.B., MacAirchinnigh M., Neuhold E.J. (eds) //Lect. Notes in Comp. Sci. - 1987. -N 252.

76. VON NEUMANN J. The EDVAS report //Computer from Pas - cal to von Neumann /Ed. H.Goldstein- -Prinston: Prinston Univ. Press. - 1979.

77. WIRTH N. Program development by step-wise refinement //Commun.ACM. - 1974. -Vol. 14. -P. 221-227.

78. СВИРИДЕНКО Д.И., КОТОВ С.В. СИГМА-язык // Настоящий сб. -С. 95-134.

Поступила в ред.-изд.отд.
15 апреля 1990 года