ЛОГИКО-МАТЕМАТИЧЕСКИЕ ОСНОВЫ ПРОБЛЕМЫ МОЗ
(Вычислительные системы)
1985 год Выпуск 107

УДК 519.685

Σ-ПРОГРАММИРОВАНИЕ

С.С.Гончаров, Д.И.Свириденко

Цель настоящей статьи - описать новую концепцию программирования, опирающуюся на современные идеи и методы математической логики. В основе данной концепции, носящей название Σ-программирование, лежит тот факт, что вычислимые функции допускают формульное описание. В качестве подобных описаний в Σ-программировании рассматриваются формулы специального вида (так называемые Σ-формулы) языка исчисления предикатов I-го порядка.

§I. О некоторых тенденциях развития программирования

Среди множества проблем, обсуждающихся в программировании, наибольшей популярностью пользуются такие, как:

- качество, в частности, надежность и корректность программного обеспечения;
- производительность труда программистов;
- "интеллектуализация" программных систем и ЭВМ.

Нетрудно понять, что все эти и другие проблемы современного программирования тесно связаны между собой, а потому представляются несостоятельными попытки решать их отдельно, изолированно друг от друга. Другими словами, предпочтителен комплексный, концептуально единый подход. Поиск и разработка такого подхода существенно зависит от того, какой представляется исследователю природа таких понятий, как информация, процесс обработки информации. Не менее важными являются здесь и такие вопросы, как "что такое задача?", "что значит решить задачу?" и др. Весь этот и родственный им комплекс проблем и вопросов возникают практически в любой области программирования. Рассмотрим некоторые из них.

I. К о н с т р у и р о в а н и е п р о г р а м м.Современное представление о процессе разработки программ связано, прежде всего, с выделением в этом процессе определенных этапов ("жизненный цикл программы"): этап спецификации, этап проектирования,этап кодирования исходного текста программы и т.д. Подобная структуризация вызвана, главным образом, сложностью и масштабностью создаваемых программ, что,в свою очередь, приводит к проблеме адекватности создаваемых программ тем задачам, для решения которых они предназначаются.Адекватность программ (иногда говорят "корректность","правильность") необходима потому,что нам нужна у в е р е н н о с т ь в правильности и надежности их функционирования; мы хотим д о в е р я т ь программам - иначе зачем они нам нужны. Следовательно, должны быть средства демонстрации подобной адекватности. Выбор же тех или иных средств существенно зависит от того, что мы понимаем под с п е ц и ф и к а ц и е й, каким мы представляем себе п р о ц е с с п р о е к т и р о в а н и я программ и как он связан с спецификацией, что мы называем п р о г р а м м о й и как происходит ее в ы п о л н е н и е. Если подобные средства допускают естественную формализацию, то, с одной стороны, появляется возможность автоматизировать либо весь процесс построения программ по их спецификациям (автоматический синтез программ), либо отдельные его фрагменты (автоматизация проектирования), а с другой - подобные формализмы для их эффективной реализации могут потребовать совершенно новых принципов организации вычислительного процесса и архитектуры ЭВМ. Последнее может,в свою очередь, привести к изменениям наших представлений о том,что есть спецификация, проектирование и кодирование программ, в том числе и о самой программе.

2. С т р у к т у р ы д а н н ы х. Когда говорят об автоматизации некоторого процесса обработки информации, то подразумевают, что речь идет прежде всего об умении решать такие проблемы, как адекватное представление информации в виде с т р у к т у р д а н н ы х и адекватное представление автоматизируемого процесса в виде а л г о р и т м а обработки соответствующих структур данных. Заметим, что,говоря о структурах данных и алгоритмах, мы как бы невольно подчеркиваем отличие этих понятий друг от друга. Нужно сказать, что в программировании весьма популярна точка зрения, которая данное различие абсолютизирует и согласно которой "программа = алгоритм + структура данных". И этому есть

свои объяснения. Например, благодаря отделению структур данных от алгоритмов их обработки, программу можно начинать проектировать, скажем, без точного представления о требованиях к структурам данных, т.е. до того, как структуры данных будут определены полно и точно. Кроме того, изменение структур данных является весьма распространенным способом улучшения программ. При этом затрагиваются модификацией лишь последние этапы проектирования программы. В то же время в программировании существует подход к созданию программ, отталкиваясь от данных, т.е. ведущим при проектировании является не алгоритм, а структура данных (например, -технология программирования). Но и здесь проводится четкое разделение понятий "структура данных" и "алгоритм". И лишь в последнее время, особенно в связи с появлением концепции "абстрактных типов данных", стали постепенно осознавать, что подобное разделение понятий "структура данных" и "алгоритм" носит не принципиальный, а искусственный характер. Эти понятия скорее отражают те роли, которые играют программистские конструкции (и даже одна и та же конструкция) в процессе вычисления - все зависит от того, как используются эти конструкции. "Абстрактные типы данных" - это фактически тот контекст, который управляет использованием объектов и в котором происходит использование тех или иных из них.

Необходимо отметить, что понятие "абстрактные типы данных" позволяет взглянуть на многие аспекты программирования с более или менее единых позиций. Как следствие повысился уровень языковых конструкций и их разнообразие, языки программирования приобрели более декларативный характер. Другими словами, выше становится уровень автоматизации информационных процессов, упрощается (с точки зрения пользователя) процесс проблемной ориентации языковых средств. Но концепция "абстрактных типов данных" требует создания, адекватных системно-технических средств обработки данных, т.е. новых вычислительных машин, основанных на отличных от традиционной фон-неймановской модели вычислений.

В области "абстрактных типов данных" сейчас наблюдается активная деятельность, ведется масштабный научный эксперимент. В рамках этой концепции в настоящее время сложилось много различных подходов (инициальный, финальный, теоретико-модельный, категорный, аксиоматический и др.). Нужно отметить, что не все они методологически и математически проработаны. На повестке дня стоит вопрос о

систематизации и возможно определенной унификации накопленных знаний в этой области.

3. О п и с а н и е б а з д а н н ы х и р а б о т а с н и м и. Решение многих задач требует зачастую огромных массивов информации. Естественно поэтому привлекать к решению подобных задач ЭВМ. Но здесь возникает проблема машинного представления огромного объема информации, ее организации, структурирования и обеспечения простого и эффективного доступа к ней, возможность постоянного хранения и обновления информации. Подобные проблемы возникают при построении систем управления сложными техническими, военными, экономическими и социальными системами, при создании информационно-поисковых, вопросно-ответных и других "интеллектуальных" систем, в частности, экспертных. Для последнего типа систем характерны также проблемы, связанные с их "интеллектуализацией" - проблемы организации диалога, общения на естественно-подобных языках, проблемы "интеллектуальной" обработки данных (логический вывод, машинное доказательство теорем, генерация гипотез, планирование хода вычислений, распознавание образов и т.п.). Очевидно, что необходимо решать все эти проблемы комплексно. Определенный прогресс в данной области связывается с так называемым р е л я ц и о н н ы м подходом. Большие надежды возлагаются на абстрактные данные. Весьма перспективным (и что главное, синтетическим) представляется л о г и ч е с к о е п р о г р а м м и р о в а н и е. Но и здесь возникает проблема эффективной реализации всех этих подходов, т.е. все упирается в вычислительные средства. Подобная задача приобретает еще более сложный концептуальный и технический характер в работе с так называемыми р а с п р е д е л е н н ы м и базами данных.

4. А р х и т е к т у р а ЭВМ. Мы уже неоднократно подчеркивали ту мысль, что реализация новой концепции представления и обработки информации требует адекватных этой концепции вычислительных средств. Другими словами, некий подход в программировании только тогда может рассчитывать на положительные результаты, когда он затрагивает не только чисто программистские, но и технические аспекты, т.е. аспекты, связанные с организацией вычислительных процессов, архитектурой вычислительных систем.

Большая часть современных идей в программировании так или иначе реализуется на фон-неймановской модели вычислений. Но все чаще обнаруживается, что узкое место этой модели (по меткому вы-

ражению Дж.Бэкуса - "узкое горлышко фон-Неймана (von Neumann bottleneck)") становится существенным препятствием для эффективной реализации этих идей. Нужны новые машины. И они появляются. Так, например, последовательная реализация идей функционального (или, как иногда говорят, аппликативного) программирования привела к появлению проектов, в которых разрабатываются не только функциональные языки, но и "функциональные" машины - "потоковые" машины, различного вида LISP-машины, "редукционные" машины и т.п. Примером такого единого концептуального проекта может служить проект под управлением Д.Тернера, в рамках которого были разработаны язык SASL (Saint Andrews Static Language) и машина KRC (Kent Recursive Calculattor). Среди функциональных машин можно упомянуть такие "редукционные" машины, как SKIM, ZAPP, ALICE (Англия), SKY (Швеция), GMD (ФРГ), "функциональные" машины, реализующие идеи Дж.Бэкуса - ARVIND, FFP-машина (США) и др. В японском проекте по созданию ЭВМ 5-го поколения также предусмотрена разработка "функциональной" машины. В этом же проекте предусмотрено создание ЭВМ, поддерживающих концепцию логического программирования (PROLOG-машины). Недавно было объявлено о создании экспериментальной PROLOG-машины PAROLOG. Активно ведутся разработки "реляционных" ЭВМ и ЭВМ, поддерживающих концепцию "абстрактных типов данных". При разработке "функциональных", "логических" и других вычислительных машин активно используются различные формы параллельной обработки данных (например, ИЛИ- и И-параллизм). И хотя подобные разработки стали возможными во многом благодаря успехам в современной технологии, нужно заметить, что мы является свидетелями совершенно новой тенденции в программировании - появление новых программистских концепций становится определяющим при создании технической базы программирования.

Заканчивая наш краткий анализ состояния дел в некоторых областях программирования, нельзя не упомянуть об одном важном обстоятельстве - имеется явное концептуальное сходство упоминавшихся выше подходов. Подтверждением этому является, например, возможность успешной реализации идей логического программирования на "редукционных" машинах, В то же время функциональное программирование может рассматриваться как частный (эквациональный) случай логического программирования. Тоже можно сказать и о реляционном подходе. Весьма отчетливо видна связь концепции "абстрактных типов данных" и реляционного подхода. В рамках первой концепции очень ин-

тересно может быть поставлена задача синтеза программ по спецификациям. Причина такой концептуальной близости ясна – все эти подходы по своей природе являются логико-математическими. Таким образом, мы наблюдаем активное проникновение идей и методов математической логики в программирование. Более того, можно предсказать, что математическая логика станет тем синтезирующим началом, которого так не хватает современной методологии, теории и технологии программирования. Данное замечание позволяет надеяться, что возможны такие логико-математические концепции машинной обработки информации, которые позволят с единых позиций взглянуть на все те проблемы, о которых речь шла выше. Нужен научный поиск, своего рода эксперимент в данном направлении. Настоящая статья относится именно к таким исследованиям.

§2. Методология Σ-программирования

Основным понятием Σ-программирования является понятие "задача". Задача в Σ-программировании представляется своей с п е ц и ф и к а ц и е й. Среди спецификаций задач выделяется специальный класс так называемых Σ-с п е ц и ф и к а ц и й, которые могут быть "исполнены". Предполагается, что "исполнение" Σ-спецификаций осуществляется неким устройством – Σ-м а ш и н о й[*]. Среди "исполняемых" спецификаций особое место отводится Σ-спецификациям, "исполнение" которых Σ-машиной осуществляется достаточно эффективно. Такие Σ-спецификации мы будем называть Σ – п р о г р а м м а м и. Синтаксически Σ-спецификации и Σ-программы неразличимы. Выделение Σ-программ осуществлено из чисто прагматических соображений, единственным критерием здесь является "эффективность исполнения". Поскольку в статье последнее понятие никак не уточняется, мы в дальнейшем не будем различать Σ-спецификации и Σ-программы. Язык, в котором записываются Σ-спецификации, мы назовем Σ-я з ы к о м. Таким образом, в концепции Σ-программирования мы выделяем Σ-язык, на котором пишутся спецификации задач и, в частности, Σ-спецификации, и Σ-машину – некое абстрактное устройство, "исполняющее" Σ-спецификации. В настоящей работе речь в основном будет идти о Σ-языке и Σ-спецификациях. Представления же о Σ-машине будут носить весьма общий и схематичный характер, поскольку мы считаем, что "Σ-машина" как концептуаль –

[*] Естественно, что в данном случае речь идет не о реальном вычислительном устройстве, а неком а б с т р а к т н о м понятии.

ное понятие должно формироваться вслед за уточнением и развитием таких понятий, как "Σ-спецификация" и "Σ-программа". Однако сразу же отметим, что как исходный текст, так и "исполняемый код" Σ-программы будут конструкциями одного и того же языка - Σ-языка. Отличать их будет только уровень абстракции.

Поскольку речь зашла об уровнях абстракции, то уместно заметить, что процесс Σ-программирования начинается с написания весьма абстрактной (по отношению к уровню Σ-машины) спецификации решаемой задачи и заканчивается созданием вполне конкретной (т.е. воспринимаемой Σ-машиной) Σ-программой. Поскольку исходная спецификация может существенно отличаться по своему виду от Σ-спецификации, то возникает проблема построения по спецификации задачи соответствующей Σ-программы. Процесс построения по исходной спецификации (включая и само построение исходной спецификации) Σ-программы мы назовем Σ-п р о е к т и р о в а н и е м. О методологических принципах, положенных в основу Σ-проектирования, речь будет идти в последующих публикациях. А здесь мы более подробно остановимся на методологических положениях, составляющих основу Σ-языка.

Выше мы уже говорили о том, что и спецификации задач, и Σ-спецификации (в том числе Σ-программы) являются синтаксическими конструкциями Σ-языка. Поскольку в реальной математической и программистской деятельности по решению задач динамические аспекты тесно переплетены со статическими, то в Σ-языке должны существовать средства, адекватно представляющие оба этих аспекта. Таким образом, Σ-язык, в терминах которого будут формулироваться задачи и записываться их решения в виде Σ-программ, должен позволять полно и удобно отражать специфические черты исследуемых проблемных областей, их динамические и статические аспекты. С этой целью Σ-язык будет содержать развитую с и с т е м у и м п е р а т и в о в (исходные операции и средства комбинирования новых) и с и с т е м у д е к л а р а т и в о в. Роль последних будут играть формулы специального языка логики предикатов I-го порядка. Из того, что Σ-спецификации должны быть "исполняемыми" описаниями, вытекает требование о существовании "естественной" императивной их интерпретации. Этим обстоятельством и объясняется то, что в качестве Σ-спецификаций выбраны формулы специального вида - так называемые Σ-ф о р м у л ы (см.§3). "Исполнение" Σ-формул будет заключаться в проверке их и с т и н н о с т и в модели. Подобная

проверка может осуществляться самыми различными способами, в том числе и синтаксическими (выводимость). Главное здесь - эффективность используемых методов (т.е. эффективность "вычислений"). Если пользователя Σ-языка интересует только возможность спецификации задачи в виде Σ-спецификации, то выбор стратегии ее исполнения ложится на Σ-транслятор. Таким образом, речь идет об универсальных "встроенных" стратегиях проверки Σ-формул на истинность. Если же пользователя интересует эффективность Σ-спецификации, то Σ-язык должен позволить ему описать стратегию "вычисления", считая подобное описание одной из компонент Σ-программ. Таким образом, в общем случае Σ-программа представляется как конструкция, состоящая из 2-х компонент: первая компонента указывает на то, *что* должна делать Σ-программа (логическая компонента), вторая определяет *как* должно это "что" делаться. При этом обе компоненты имеют сугубо декларативный вид - вид Σ-формул. Подобный взгляд на Σ-программы придает процессу Σ-проектирования существенно математический, исследовательский характер - нужно не только построить Σ-спецификацию, но и превратить ее в Σ-программу, т.е. сделать процесс проверки истинности эффективным.

Другим способом повышения эффективности Σ-программ является их *преобразование*. Естественно потребовать, чтобы подобные преобразования осуществлялись в рамках и средствами Σ-языка. Более того, мы потребуем, чтобы в рамках Σ-языка могла быть осуществлена возможность логического обоснования корректности преобразований Σ-программ. Здесь мы коснулись одного обстоятельства, на котором хотелось бы остановиться подробнее. Речь идет о *проблеме самоприменимости*. В данном случае эта проблема интересует нас не только в том смысле, как она понимается в математической теории вычислений (например, когда изучается осмысленность функциональных выражений вида $f(f)$), а в том, что исследование языка, описание конструкций над ним может быть осуществлено в рамках и средствами этого же языка. При этом в отличие, скажем, от арифметики Пеано, где иерархию описаний конструктивных объектов и самих конструкций можно погрузить в исходный формализм с помощью подходящих нумераций, которые сами являются конструкциями арифметики, речь должна идти о *явном* описании и представлении всех нужных конструкций, так для пользователя существенными являются требования простоты, удобства и понятности. Подобные "самоприменимые" языки (таковым является, например, ес -

тественный язык) обладая, с одной стороны, указанными выше достоинствами, весьма трудны при их семантическом изучении. Поэтому, требуя от Σ-языка "явной самоприменимости", мы вправе ожидать нетривиальность устройства данного языка и его теории, а также нетрадиционности архитектурных решений при создании Σ-машины.

Создавая язык программирования, прежде всего нужно хорошо представлять себе его семантику и, в частности, решать проблему семантического представления понятия "конструктивный объект". Под конструктивным объектом мы будем понимать конечный объект, снабженный "внутренней системой координат" (вместе с указанием некоторого начального элемента - "начало координат"), которая позволяет однозначно выделить в объекте любую его компоненту. Примерами конструктивных объектов могут служить конечные слова, графы, деревья, автоматы и т.п. Для нас же наиболее адекватным семантическим представлением понятия "конструктивный объект" будет к о н е ч н ы й с п и с о к: с одной стороны, конечные списки обладают определенной кодифицирующей универсальностью - все примеры конструктивных объектов, упомянутые выше, допускают естественное их представление в виде списков, а с другой - структура данных "список" допускает удовлетворительную машинную реализацию.

Конструктивный объект может рассматриваться как элемент некоторого "реального мира", заданного формально, скажем, в виде многоосновной модели (возможно, с некоторой "надстройкой" над этой моделью). Так как речь идет о конструктивности представлений, то наиболее естественно такую модель мыслить и н т е н с и о н а л ь н о: либо как некую процедуру, генерирующую элементы модели (и элементы "надстройки"), либо как конструктивную (позитивную или сильно конструктивную) модель. Такие модели естественно, даже если они имеют весьма абстрактную природу, называть с т р у к т у р а м и д а н н ы х, или просто д а н н ы м и, а их элементы - э л е м е н т а м и д а н н ы х. Кроме того, конструктивный объект можно рассматривать и как объект "мира", представленного в виде с и с т е м ы з н а н и й об этом мире, например, в виде ф о р м а л ь н о й т е о р и и. Такая теория выделяет среди всех данных такие, которые являются ее моделями. В определенном смысле теория выступает как некое единое описание всего класса выделяемых ею моделей, т.е. как т и п класса структур данных. Именно по этой причине мы в дальнейшем будем рассматривать формальные теории как наиболее адекватную формализацию понятия "абстрактный тип данных".

Решая некоторую задачу, мы так или иначе будем вынуждены помимо использования знаний о некоторой модели, представляющей тот "мир",к которому относится задача, строить некие дополнительные, вспомогательные конструкции над моделью,манипулировать ими и рассуждать о них. Поскольку нас интересует решение задачи с помощью ЭВМ, т.е. конструктивное, алгоритмическое решение,то естественно, что подобные вспомогательные конструкции по необходимости должны быть конструктивными объектами и носить, как уже отмечалось,явный характер.Итак, нам нужна модель,для которой был бы просто и удобно реализован весь набор требующихся нам конструкций конструктивных объектов для решения задач. Именно поэтому, говоря выше о модели, как о формальном представлении проблемной области, мы упоминали и некоторую "надстройку" над ней. Такой надстройкой будут списочные структуры над многоосновными моделями. Модель и ее надстройка и будут той формальной конструкцией, в которой представлена как интересующая нас предметная область, так и все средства манипулирования с нужными нам построениями. Работая в такой "двухъярусной" модели, мы будем использовать синтаксически описанные в терминах Σ-языка конструкции. Поэтому вполне естественно требовать, чтобы для "разумных" описаний в нашей модели существовали соответствующие им конструкции (реализации описаний).При этом структурированность списочной надстройки должна найти свое естественное отражение в языке работы с моделью.

Списочная надстройка как надстройка всех конечных списков над исходной многоосновной моделью будет устроена достаточно единообразным способом.Поэтому на нее можно смотреть как на модель (структуру данных) некоторой формальной теории. Таким образом, работая в рамках Σ-языка, мы будем иметь дело с формальными теориями, где одна часть - теория предметной области, а другая - формальная теория списков над данной областью. Изучение подобных формальных теорий будет осуществляться в рамках так называемой теории списочных расширений GES (см. §3). Содержательная математическая теория списочных расширений и будет нашим дальнейшим инструментом при построении как теории Σ-языка, так и теории Σ-машины. В рамках этой теории будут также изучаться и логико-математические модели систем проектирования Σ-программ. Другими словами, теория GES, излагаемая в следующем параграфе, является теоретической основой Σ-программирования.

Как мы убедимся впоследствии, в теории GES имеются очень мощные средства для представления и изучения широкого класса эффективных конструкций, используемых в Σ-программировании. Совокупность этих конструкций оказывается замкнутой относительно различных рекурсивных определений, что и дает возможность в Σ-языке строить и изучать различные формальные теории.

Ядро Σ-языка будет составлять набор исходных императивов, которые могут рассматриваться как простейшие операции языка, и набор исходных предикатов. В терминах этого базиса можно будет строить уже достаточно сложные Σ-программы. Однако, как будет показано в теории GES, в Σ-языке имеется возможность естественного консервативного расширения базисных понятий. Необходимость подобного расширения нужна по многим причинам. Главная из них - удобство пользования Σ-языком.

Основным механизмом консервативного расширения сигнатуры Σ-языка будет построение инициальных объектов для систем квазитождеств, Σ-определимость и представимость. Представляется также весьма перспективным (однако это требует дополнительных исследований) для консервативного расширения базиса Σ-языка механизм, подобный механизму "извлечения программ из конструктивных доказательств". Этот же механизм предполагается использовать также при проектировании Σ-программ. Подобные механизмы конструктивного расширения будут играть главную роль в задаче проблемной ориентации Σ-языка, решение которой будет заключаться в генерации абстрактных типов данных, формально представляющих понятие рассматриваемой предметной области. И вновь, в силу свойства "самоприменимости" Σ-языка, данные механизмы и конструкции могут использоваться и совершенствоваться в рамках языка и его средствами.

Есть основание считать (но опять-таки это требует дополнительных исследований), что в Σ-языке можно будет конструировать (определять) императивы высокого уровня абстракции (функционалы конечных типов), что позволило бы строить функциональные программы. Это означает, что Σ-программирование действительно является концепцией, синтезирующей в себе большую часть логико-математических подходов в программировании: логического, функционального и трансформационного программирования, концепции "абстрактных типов данных", синтез программ и др. Каждый из этих подходов находит свое отражение в Σ-программировании.

§3. Теория списочных расширений GES

Пусть $\mathcal{M}$ - многосортная модель сигнатуры σ_0, I - множество сортов этой модели и $\{M_i\}_{i\in I}$ - семейство ее основных множеств. В дальнейшем мы будем рассматривать модели со всюду определенными операциями и предикатами, хотя представляется целесообразным изучение и более общего случая - частичных моделей, для чего среди сортов модели нужно выделить сорт ω, для которого $M_\omega = \{\perp\}$, где $\perp$ интерпретируется как "неопределенность". Элементы модели $\mathcal{M}$ мы будем рассматривать как а т о м ы, из которых строятся списки, списки списков и т.д. Данную списочную надстройку над моделью обозначим через $S(\mathcal{M})$. В ней особое место занимает пустой список nil, который, кстати, может одновременно рассматриваться и как атом. На $S(\mathcal{M})$ можно смотреть как на новый сорт в объектов, предназначенный для определения над $\mathcal{M}$ различных конструкций и изучения их алгоритмической природы. Для этого помимо самих элементов множества $S(\mathcal{M})$ необходимо ввести дополнительные операции и отношения. 0-местную функцию nil мы уже ввели. Кроме этого, нам потребуются следующие операции и отношения*):

head: $S(\mathcal{M}) \to S(\mathcal{M}) \times (\bigcup_{i\in I} M_i)$ - операция взятия последнего элемента в списке, если список не пустой, и равная nil в остальных случаях;

tail: $S(\mathcal{M}) \to S(\mathcal{M})$ - операция взятия остатка списка, из которого удален последний элемент, если список не пуст, и равная nil в противном случае;

cons: $S(\mathcal{M}) \times (S(\mathcal{M}) \cup (\bigcup_{i\in I} M_i)) \to S(\mathcal{M})$ - операция присоединения к списку нового элемента в качестве последнего;

conc: $S(\mathcal{M}) \times S(\mathcal{M}) \to S(\mathcal{M})$ - операция конкатенации двух списков;

$\in \subseteq (S(\mathcal{M}) \cup (\bigcup_{i\in I} M_i)) \times S(\mathcal{M})$ - отношение "быть элементом списка";

$\sqsubseteq \subseteq S(\mathcal{M}) \times S(\mathcal{M})$ - отношение "быть началом списка";

$S^1 \subseteq S(\mathcal{M}) \cup (\bigcup_{i\in I} M_i)$ - унарное отношение "быть списком".

Рассмотрим стандартную списочную надстройку $S^{fin}(\mathcal{M})$, представляющую для Σ-программирования главный интерес и состоящую из конечных списков. Положим

---

*) Операции head и tail несколько отличаются от общепринятых. Выбор принятой в данной статье семантики этих операций определялся только соображениями удобства.

$S^0(\mathcal{M}) \leftrightarrows$ { конечные слова (линейные списки) из элементов $M = \bigcup_{i \in I} M_i$};

$S^1(\mathcal{M}) \leftrightarrows$ { линейные списки из элементов $S^0(\mathcal{M}) \cup M$ };

. . .

$S^{n+1}(\mathcal{M}) \leftrightarrows$ {линейные списки из элементов $S^n(\mathcal{M}) \cup M$}; и т.д.

Пусть $S^{fin}(\mathcal{M}) = \bigcup_{n \geq 0} S^n(\mathcal{M})$.

Если список α состоит из элементов $\alpha_1, \ldots, \alpha_n$ в данной последовательности, то будем писать $\alpha = \langle \alpha_1, \ldots, \alpha_n \rangle$. Определим на $S^{fin}(\mathcal{M})$ операции head, tail, cons, conc, nil, полагая:

head $(\langle \alpha_1, \ldots, \alpha_{n+1} \rangle) = \alpha_{n+1}$ и head $(nil) = nil$;

tail $(\langle \alpha_1, \ldots, \alpha_n, \alpha_{n+1} \rangle) = \langle \alpha_1, \ldots, \alpha_n \rangle$, tail$(\langle \alpha_1 \rangle) =$ = tail $(nil) = nil$;

cons $(\langle \alpha_1, \ldots, \alpha_n \rangle, \beta) = \langle \alpha_1, \ldots, \alpha_n, \beta \rangle$;

conc $(\langle \alpha_1, \ldots, \alpha_n \rangle, \langle \beta_1, \ldots, \beta_m \rangle) = \langle \alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m \rangle$;

nil $= \langle\ \rangle$;

$\beta \in \langle \alpha_1, \ldots, \alpha_n \rangle \leftrightarrow [\beta = \alpha_i$ для некоторого $1 \leq i \leq n]$;

$\langle \alpha, \ldots, \alpha \rangle \subseteq \langle \beta_1, \ldots, \beta_m \rangle \leftrightarrow [n \leq m$ и для каждого $1 \leq i \leq n$ $\alpha_i = \beta_i]$;

$\alpha \in S^1 \leftrightarrow \alpha \in S^{fin}(\mathcal{M})$.

Обозначим через $HFS(\mathcal{M})$ модель, полученную из $\mathcal{M}$ добавлением нового сорта s элементов $S(\mathcal{M})$ с указанными выше операциями. Будем называть ее наследственно конечной списочной надстройкой над $\mathcal{M}$.

Определим теперь формальный язык исчисления предикатов I-го порядка для работы с этой моделью. Обозначим через σ сигнатуру $\sigma_0 \cup$ {head, tail, cons, conc, nil, $\in, \subseteq, S^1$} и положим $\sigma^* \leftrightarrows \sigma \cup \{Q_1^{m_1}, \ldots, Q_n^{m_n}, \ldots\}$, где $Q_i^{m_i}$ $(i \geq 1)$ – m_i-арные предикатные символы, не входящие в σ_0. Заметим, что если I – множество сортов сигнатуры σ_0, то $I \cup \{s\}$ – множество сортов сигнатуры σ и σ^*. Кроме сортов, нам потребуется еще одна конструкция – т и п, под которым мы будем понимать конечный упорядоченный набор $\langle K_1, \ldots, K_n \rangle$ подмножеств I. В дальнейшем мы будем считать, что каждый сигнатурный символ обладает определенным типом.

Т е р м ы (сигнатур σ_0, σ и σ^*) определяются обычным образом:

- переменные и константы любых типов вида $\langle K\rangle, K \subseteq I \cup \{s\}$ будут термами этих же типов;

- если F - символ операции типа $\langle K_1,\dots,K_n,K\rangle$ (таким образом, F интерпретируется в модели $\mathfrak{M}$ как операция

$$\bar{F}\colon \Big(\bigcup_{i\in K_1} M_i\Big)\times\dots\times\Big(\bigcup_{i\in K_n} M_i\Big) \to \bigcup_{i\in K} M_i),$$

а $t_1,\dots,t_n$ - допустимые для F (т.е. нужного типа) термы, то $F(t_1,\dots,t_n)$ - терм типа $\langle K\rangle$;

- других термов нет.

Столь же традиционно определяются ф о р м у л ы (сигнатур σ_0, σ и σ^*)*):

- если t_1 и t_2 - термы одного и того же типа, то $t_1 = t_2$ - формула;

- если P - n-местный предикатный символ типа $\langle K_1,\dots,K_n\rangle$, а $t_1,\dots,t_n$ - допустимые для P термы, то $P(t_1,\dots,t_n)$ - формула;

- если Ψ и Φ - формулы, то $(\Psi\vee\Phi)$, $(\Psi\,\&\,\Phi)$, $(\Psi\to\Phi)$, $\neg\Psi$, $\forall x\,\Psi$, $\exists x\,\Psi$ - формулы, где x - переменная какого-либо типа $\langle K\rangle$;

- если Ψ - формула, t - терм типа $\langle\{s\}\rangle$, а x - переменная типа $\langle I\cup\{s\}\rangle$, то выражения $(\forall x\in t)\Psi$, $(\exists x\in t)\Psi$, $(\forall x\subseteq t)\Psi$, $(\exists x\subseteq t)\Psi$ также будут формулами; кванторы в этих записях назовем о г р а н и ч е н н ы м и.

ОПРЕДЕЛЕНИЕ I. а) Формула, в записи которой встречаются только ограниченные кванторы, называется ф о р м у л о й с о г р а н и ч е н н ы м и к в а н т о р а м и, или Δ_0-ф о р м у л о й;

б) Σ-формулой называется любая формула из наименьшего класса формул, содержащего все Δ_0-формулы и замкнутого относительного дизъюнкции и конъюнкции формул, а также навешивания кванторов существования. Для Σ-формул сигнатуры σ^* требуется также, чтобы предикатные символы $Q_1^{m_1},\dots,Q_n^{m_n},\dots$ входили бы в такие формулы только положительно.

Назовем список α из двух элементов α_1, α_2, входящих в α именно в такой последовательности, у п о р я д о ч е н н о й п а р о й и будем обозначать его $\langle\alpha_1,\alpha_2\rangle$. Заметим, что свойст-

*) Исключение составляет последний пункт определения формулы.

во "быть упорядоченной парой" выразимо в виде Δ_0-формулы:

$$\mathrm{ORDPARE}(\alpha) = (\exists \alpha_1 \in \alpha)(\exists \alpha_2 \in \alpha)(\alpha = \mathrm{cons}(\mathrm{cons}(\mathrm{nil}, \alpha_1), \alpha_2)).$$

Таким образом, запись $\langle \alpha_1, \alpha_2 \rangle$ является фактически сокращением для $\mathrm{cons}(\mathrm{cons}(\mathrm{nil}, \alpha_1), \alpha_2))$.

Список α назовем **списочной функцией списка** β, если элементами α являются упорядоченные пары, у которых в качестве первых элементов выступают начальные сегменты списка β. При этом элементы списка α упорядочены в порядке возрастания первых элементов пар из α (относительно предиката $\subseteq$). Таким образом, предикат $\underline{\mathrm{LF}}(\alpha, \beta)$ ("α есть списочная функция β") может быть представлен следующей Δ_0-формулой:

$$\underline{\mathrm{LF}}(\alpha,\beta) \Leftrightarrow (\forall \gamma \in \alpha)(\underline{\mathrm{ORDPARE}}(\gamma) \;\&\; (\forall \alpha_1 \in \gamma)(\forall \alpha_2 \in \gamma) \times$$
$$\times (\langle \alpha_1, \alpha_2 \rangle = \gamma \to \alpha_1 \subseteq \beta)) \;\&\; (\forall \delta \subseteq \beta)(\exists \gamma \in \alpha)(\underline{\mathrm{cons}}(\mathrm{nil}, \delta) =$$
$$= \underline{\mathrm{tail}}(\gamma) \;\&\; (\forall \alpha' \subseteq \alpha)(\forall \alpha'' \subseteq \alpha')(\exists \delta' \in \underline{\mathrm{tail}}(\underline{\mathrm{head}}(\alpha'))) \times$$
$$\times (\exists \delta'' \in \underline{\mathrm{tail}}(\underline{\mathrm{head}}(\alpha'')))(\delta'' \sqsubset \varepsilon').$$

Список α, являющийся списочной функцией списка β, можно интерпретировать как табличную функцию, аргументами которой являются начальные отрезки списка β. Здесь $\sqsubset \Leftrightarrow \subseteq \setminus =$. Список α назовем **наследственной списочной функцией** β (соответствующий предикат обозначим $\mathrm{HLF}(\alpha,\beta)$), если α является списочной функцией β и для любого $\gamma \in \alpha$ выполнено:

1) если $\underline{\mathrm{head}}(\underline{\mathrm{tail}}(\gamma)) = \underline{\mathrm{nil}}$, то $\underline{\mathrm{head}}(\gamma) = \underline{\mathrm{nil}}$;

2) если $\gamma = \langle \alpha_1, \alpha_2 \rangle$ и $\alpha_1 = \underline{\mathrm{cons}}(\alpha_1', a)$, то $\alpha_2 = \underline{\mathrm{cons}}(\alpha_2', b)$ и $\langle \alpha_1', \alpha_2' \rangle \in \alpha$.

Заметим, что пара $\langle \alpha_1', \alpha_2' \rangle$ в списке α должна находиться непосредственно перед парой $\langle \alpha_1, \alpha_2 \rangle$.

Наследственная списочная функция α списка β может рассматриваться как "правильно устроенная" табличная функция, наследующая свои предыдущие значения.

Формальная теория GES списочных надстроек над моделью $\mathcal{M}$ в качестве своих аксиом имеет универсальные замыкания следующих формул.

АКСИОМА ПУСТОГО СПИСКА. $\neg(\exists \delta)(\delta \in \underline{\mathrm{nil}})$.

АКСИОМА ЕДИНСТВЕННОСТИ. $(\underline{\mathrm{cons}}(\alpha,\varepsilon) = \underline{\mathrm{cons}}(\delta',\delta') \to (\alpha = \alpha' \;\&\; \delta = \delta'))$, где α, α' – переменные типа $\langle\{s\}\rangle$ (т.е. типа **список**), а δ,δ' – типа $\langle I \cup \{s\}\rangle$.

В дальнейшем, когда из контекста однозначно видно, какого типа рассматриваются переменные, мы будем опускать описание типов этих переменных.

АКСИОМЫ СПИСОЧНЫХ ОПЕРАЦИЙ.

$$\underline{tail}\ (\underline{cons}(\alpha,\delta)) = \alpha;$$

$$\underline{head}\ (\underline{cons}(\alpha,\delta)) = \delta;$$

$$(\neg(\alpha = \underline{nil}) \to \underline{cons}(\underline{tail}(\alpha),\underline{head}(\alpha)) = \alpha);$$

$$\underline{tail}\ (\underline{nil}) = \underline{nil};$$

$$\underline{head}\ (\underline{nil}) = \underline{nil};$$

$$\underline{cons}\ (\underline{conc}(\alpha,\beta),\gamma) = \underline{conc}(\alpha,\underline{cons}(\beta,\gamma));$$

$$\underline{conc}\ (\underline{conc}(\alpha,\beta),\gamma) = \underline{conc}(\alpha,\underline{conc}(\beta,\gamma));$$

$$\underline{conc}\ (\underline{nil},\alpha) = \underline{conc}(\alpha,\underline{nil}) = \alpha.$$

АКСИОМЫ ИНДУКЦИИ. $([\Phi]^x_{\underline{nil}} \,\&\, (\forall\alpha)(\forall\delta)([\Phi]^x_\alpha \to [\Phi]^x_{\underline{cons}(\alpha,\delta)}) \to \alpha\Phi)$, где Φ – произвольная формула и запись $[\Phi]^x_t$ означает подстановку вместо всех свободных вхождений переменной x в Φ терма t так, чтобы не было коллизии переменных.

АКСИОМА РАВНООБЪЕМНОСТИ.

$$\alpha = \beta \leftrightarrow (\forall\gamma \subseteq \alpha)(\gamma \subseteq \beta \,\&\, (\neg(\gamma = \beta) \to$$

$$\to (\exists\delta\in\alpha)(\underline{cons}(\gamma,\delta) \subseteq \alpha \,\&\, \underline{cons}(\gamma,\delta) \subseteq \beta))).$$

АКСИОМА ФУНДИРУЕМОСТИ.

$$(\forall\delta)((\forall\gamma \in \delta)\Phi(\gamma) \to \Phi(\delta)) \to \forall\alpha\Phi(\alpha).$$

АКСИОМА Δ_0-ВЫДЕЛЕНИЯ. Если $\Phi(\alpha)$ – Δ_0-формула со свободной переменной α, то имеет место

$$(\forall\beta)(\forall\alpha)(\underline{HLP}(\alpha,\beta) \,\&\, (\forall\beta' \subseteq \beta)(\forall\delta \in \beta)(\underline{cons}(\beta',\delta) \subseteq \beta \to$$

$$\to (\forall\gamma_1\in\alpha)(\forall\gamma_2\in\alpha)(\underline{head}(\underline{tail}(\gamma_1)) = \beta' \,\&\, \underline{head}(\underline{tail}(\gamma_2)) =$$

$$= \underline{cons}(\beta',\delta) \to (\Phi(\delta) \to \underline{cons}(\underline{head}(\gamma_1),\delta) = \underline{head}(\gamma_2)) \,\&$$

$$\&(\neg\Phi(\delta) \to \underline{head}(\gamma_1) = \underline{head}(\gamma_2)))) \,\&\, (\forall\gamma \in \alpha)(\underline{head}(\underline{tail}(\gamma)) =$$

$$= \underline{nil} \to \underline{head}(\gamma) = \underline{nil})).$$

Данная аксиома утверждает существование для каждого списка β табличной функции, выделяющей из него только те элементы, на которых выполняется свойство Φ.

АКСИОМА Δ_0-ВЫБОРКИ. Если $\Phi(x,y)$ – Δ_0-формула, то справедливо следующее:

$$(\forall\beta)[(\forall\alpha \in \beta)(\exists\delta)\Phi(\alpha,\delta) \to \exists\alpha\,(\underline{\mathrm{HLP}}(\alpha,\beta)\,\&$$
$$\&\ (\forall\gamma \in \alpha)\ x\ (((\underline{\mathrm{head}}(\underline{\mathrm{tail}}(\gamma)) = \underline{\mathrm{nil}}) \to \underline{\mathrm{head}}(\gamma) = \underline{\mathrm{nil}})\,\&$$
$$\&\ (\neg(\underline{\mathrm{head}}(\underline{\mathrm{tail}}(\gamma)) = \underline{\mathrm{nil}}) \to \Phi(\underline{\mathrm{head}}(\underline{\mathrm{head}}(\underline{\mathrm{tail}}(\gamma))),$$
$$\underline{\mathrm{head}}(\underline{\mathrm{head}}(\gamma))))))].$$

Эта аксиома говорит о возможности в каждом конкретном случае (т.е. для каждого списка β) реализовать "функциональное соответствие", определяемое формулой Φ, ограничившись только некоторым списком α, а не всем универсом.

На этом список аксиом, описывающих списочную надстройку над моделью $\mathcal{M}$, заканчивается. Хотелось бы обратить внимание читателя на одно обстоятельство. Вид аксиом несколько громоздок, но если внимательно их проанализировать, то станет ясна причина этой громоздкости – аксиомы конструктивны по своей природе, так как они говорят не только о существовании нужных объектов, но и о том, как их построить. Подобная конструктивность является следствием детерминированности списочных конструкций в отличие, скажем, от теоретико-множественных построений.

Рассмотрим некоторые свойства теории GES, иллюстрирующие ее адекватность целям, о которых шла речь в § 2.

Прежде всего заметим, что в GES оказываются определимыми традиционные списочные конструкции, например, конструкция "список списков":

$$(\forall\beta)(\exists\alpha)(\underline{\mathrm{HLP}}(\alpha,\beta)\ \&\ (\forall\gamma \subseteq \beta)(\forall\delta \in \beta)(\mathrm{cons}(\gamma,\delta) \subseteq$$
$$\subseteq \beta \to (\exists\gamma_1 \in \alpha)(\exists\gamma_2 \in \alpha)(\underline{\mathrm{head}}(\underline{\mathrm{tail}}(\gamma_1)) = \gamma\ \&\ \underline{\mathrm{head}}$$
$$(\underline{\mathrm{tail}}(\gamma_2)) = \mathrm{cons}(\gamma,\delta)\ \&\ (S^1(\delta) \to \underline{\mathrm{conc}}(\mathrm{head}(\gamma_1),\delta) =$$
$$= \mathrm{head}(\gamma_2))\ \&\ (\neg S^1(\delta) \to \underline{\mathrm{cons}}(\underline{\mathrm{head}}(\gamma_1),\delta) = \underline{\mathrm{head}}(\gamma_2))\ \&$$
$$\&(\forall\gamma \in \alpha)(\underline{\mathrm{head}}(\underline{\mathrm{tail}}(\gamma)) = \underline{\mathrm{nil}} \to \underline{\mathrm{head}}(\gamma) = \underline{\mathrm{nil}})).$$

Пусть теперь Φ – некоторая формула и v – переменная, не встречающаяся в ее записи. Мы будем писать $\Phi^{(v)}_{\in}$, обозначая этим результат замены всех входящих в Φ неограниченных кванторов на ограниченные: $\exists\alpha$ на $\exists\alpha \in v$ и $\forall\alpha$ на $\forall\alpha \in v$, и писать $\Phi^{(v)}_{\subseteq}$ при осуществлении замен вида: $\exists\alpha$ на $\exists\alpha \subseteq v$, $\forall\alpha$ на $\forall\alpha \subseteq v$. Если

же в формуле Φ произошли замены всех неограниченных кванторов на ограниченные без указания типа замены ($\in$ или $\subseteq$), то будем обозначать результат замены через $\Phi^{(v)}$. Заметим, что $\Phi^{(v)}_{\in}$, $\Phi^{(v)}_{\subseteq}$ и $\Phi^{(v)}$ являются Δ_0-формулами. Если же Φ уже была Δ_0-формулой, то $\Phi = \Phi^{(v)}_{\in} = \Phi^{(v)}_{\subseteq} = \Phi^{(v)}$.

ТЕОРЕМА I (принцип Σ-рефлексии). Для любой Σ-формулы Φ сигнатуры σ имеет место следующее:

а) $GES \vdash \Phi \leftrightarrow \exists v\, \Phi^{(v)}_{\in}$,

б) $GES \vdash \Phi \leftrightarrow \exists v\, \Phi^{(v)}_{\subseteq}$,

в) $GES \vdash \Phi \leftrightarrow \exists v\, \Phi^{(v)}$.

В дальнейшем формулу вида $\exists\alpha\Phi(\alpha)$, где Φ – Δ_0-формула, будем называть Σ_1-формулой. Таким образом, теорема I говорит о том, что каждая Σ-формула эквивалентна в GES Σ_1-формуле.

Выше мы выделили класс Σ-формул. С другой стороны, можно выделить класс Π-формул – наименьший класс формул, содержащий Δ_0-формулы и замкнутый относительно дизъюнкции, конъюнкции, навешивания ограниченных кванторов и неограниченного квантора всеобщности. Очевидно, что отрицание Σ-формулы логически эквивалентно Π-формуле, и обратно. Формула, логически эквивалентная одновременно Σ-формуле и Π-формуле, называется Δ-формулой.

ТЕОРЕМА 2 (принцип Δ-выделения). Для любых Σ-формул Φ и Ψ справедливо:

$$GES \vdash (\forall\alpha \in \beta)(\Phi(\alpha) \leftrightarrow \neg\Psi(\alpha)) \to \exists\alpha(\underline{HLP}(\alpha,\beta) \,\&$$
$$\&(\forall\gamma \in \alpha)[(\underline{head}(\underline{tail}(\gamma)) = \underline{nil} \to \underline{head}(\gamma) = \underline{nil})\&$$
$$\&(\neg(\underline{head}(\underline{tail}(\gamma)) = \underline{nil}) \to (\Psi(\underline{head}(\underline{head}(\underline{tail}(\gamma)))) \to$$
$$\to \underline{cons}(\underline{cons}\,(\underline{nil},\underline{tail}(\underline{head}(\underline{tail}(\gamma)))),\underline{tail}(\underline{head}(\gamma))) \in \alpha))]).$$

СЛЕДСТВИЕ I.

$$GES \vdash (\forall\alpha\in\beta)(\Phi(\alpha) \leftrightarrow \neg\Psi(\alpha)) \to (\exists\alpha)((\forall\delta \in \beta)(\Phi(\delta) \to$$
$$\to \delta \in \alpha)) \,\&\, (\forall\gamma \in \beta)(\Psi(\gamma) \to \neg(\gamma \in \alpha)).$$

ТЕОРЕМА 3 (принцип Σ-выборки). Для любой Σ-формулы Φ имеет место

$$GES \vdash (\forall \beta \subseteq a)(\exists \gamma)\ \Phi(\beta,\gamma) \to (\exists \alpha)(\exists b)(\underline{STRLF}(\alpha,a,b)\ \&\ \&(\forall \delta \subseteq a)(\neg(\delta=\underline{nil}) \to \Phi(\underline{head}(\delta),\alpha(\delta)))))),$$

где предикат $\underline{STRLF}(\alpha,a,b)$ ("список α структурно отображает список a на b") определяется формулой

$$\underline{STRLF}(\alpha,a,b) \Leftrightarrow [\underline{HLF}(\alpha,a)\ \&\ (\forall \gamma \in \alpha)(\forall \gamma' \in \alpha)(((\underline{head}(\underline{tail}(\gamma)) = \underline{nil}) \to (\underline{head}(\gamma) = \underline{nil}))\ \&\ (\neg(\underline{head}(\underline{tail}(\gamma)) = \underline{nil})\ \&\ (\underline{head}(\underline{tail}(\gamma)) = \underline{tail}(\underline{head}(\underline{tail}(\gamma'))) \to \underline{tail}(\underline{head}(\underline{tail}(\gamma'))) = \underline{head}(\underline{tail}(\gamma)))\ \&\ \underline{head}(\underline{head}(\alpha)) = b].$$

ЗАМЕЧАНИЕ. В определении предиката $\underline{STRLF}(\alpha,a,b)$ каждому начальному отрезку из a список α "соотносит" единственный начальный отрезок из b, причем "отображение" α монотонно по включению списков "$\subseteq$". Поэтому можно ввести функциональное обозначение $\alpha(\gamma)$ для каждого $\gamma \subseteq a$, где $\alpha(\gamma)$ обозначает соответствующий γ при структурном отображении начальный отрезок списка b. Через $\underline{\alpha(\gamma)}$ будем обозначать последний элемент списка $\alpha(\gamma)$. Выражение $\alpha(\gamma) = \delta$ достаточно просто определяется формулой:

$$\alpha(\gamma) = \delta \Leftrightarrow (\exists s \in \gamma)(s = \underline{cons}(\underline{cons}(\underline{nil},\gamma),\delta)).$$

Аналогично можно поступать и в случае $\underline{LF}(\alpha,a)$, вводя и здесь обозначение $\alpha(\gamma)$ для значения соответствия α на отрезке $\gamma \subseteq a$ и обозначая $\underline{\alpha(\gamma)}$ последний элемент в списке $\alpha(\gamma)$. Таким образом, все вхождения выражения $\alpha(\gamma) = \delta$ в формулы можно заменить Δ_0-подформулой, выражающей это равенство.

ОПРЕДЕЛЕНИЕ 2. Пусть $\Phi(\bar{x},y)$ - Σ-формула сигнатуры σ, для которой справедливо $GES \vdash (\forall \bar{x})(\exists!y)\Phi(\bar{x},y)$, где $\bar{x} = \langle x_1,\ldots,x_n \rangle$, и F - n-местный функциональный символ, не принадлежащий сигнатуре σ. Положим

$$(F):\ F(\bar{x}) = y \leftrightarrow \Phi(\bar{x},y)$$

и будем говорить, что F Σ-определим в GES.

Аналогичным образом можно ввести понятие Σ-определимого предиката.

ТЕОРЕМА 4. Для любой формулы $\Phi(\bar{z},F)$ сигнатуры $\delta \cup \{F\}$ существует формула Φ_0

сигнатуры σ такая, что $GES+(F)\vdash\Phi(\bar z,F)\leftrightarrow\Phi_0(\bar z)$. Если Φ - Σ-формула, то Φ_0 также Σ-формула. Если Φ - Δ_0-формула, то найдутся такие Σ- и Π-формулы Φ_0 и Φ_1 соответственно (сигнатуры σ), что

$$GES+(F)\vdash\Phi\leftrightarrow\Phi_0,\quad GES+(F)\vdash\Phi\vdash\Phi_1,$$

и $GES+(F)$ является консервативным расширением GES.

ЗАМЕЧАНИЕ. Аналогичную теорему можно доказать и для предикатов. Эти результаты, в частности, последний пункт, имеют для Σ-программирования чрезвычайно важное значение в связи с замечаниями, сделанными в §2 относительно определимых конструкций в Σ-языке, консервативно его расширяющих.

Также исключительно важную роль в теории GES и в Σ-программировании в целом играет конструкция $\underline{TC}(\alpha)$ - транзитивное замыкание списка α. Ее определение весьма громоздко и поэтому здесь не приводится. Укажем только, что по своему смыслу она аналогична понятию транзитивного замыкания множества. $TC(\alpha)$ как Σ-определимая функция имеет важное значение в теории Σ-рекурсии, позволяя строить по любым Σ-определимым функциям рекурсивно новую Σ-определимую функцию. Возможность и корректность такого построения устанавливается теоремой 5, которую можно рассматривать как аналог теоремы Ганди в теории допустимых множеств [I,2]. Прежде чем перейти к ее формулировке, определим некоторые конструкции.

Пусть $\mathcal{M}$ - модель сигнатуры σ_0. Многоосновную модель $\mathcal{A}(\mathcal{M})$ сигнатуры σ, получаемую из $\mathcal{M}$ присоединением к ней объектов нового сорта s, будем называть списочным расширением модели $\mathcal{M}$, если $\mathcal{A}(\mathcal{M})\models GES$. Если $\mathcal{A}(\mathcal{M})$ - списочное расширение $\mathcal{M}$, то совокупность всех объектов сорта s будем обозначать через $S(\mathcal{M})$ и называть их списками над $\mathcal{M}$, а само множество $S(\mathcal{M})$ - списочной надстройкой над $\mathcal{M}$. В дальнейшем списочное расширение $\mathcal{A}(\mathcal{M})$ мы будем часто обозначать как пару $\langle\mathcal{M};S(\mathcal{M})\rangle$. Примером списочного расширения может служить модель $HFS(\mathcal{M})\leftrightarrows\langle\mathcal{M};S^{fin}(\mathcal{M})\rangle$.

Пусть $\Phi(\bar x,\bar y,Q)$ (где $\bar x$ и $\bar y$ - наборы свободных переменных) есть Σ-формула сигнатуры σ^*, имеющая позитивные вхождения только одной предикатной переменной $Q\notin\sigma$, и $\mathcal{A}(\mathcal{M})$ - списочное расширение модели $\mathcal{M}$. Определим оператор

$$\Gamma_{\Phi,\bar y}(\tilde Q)\leftrightarrows\{\bar a\mid\mathcal{A}(\mathcal{M})\models\Phi(\bar a,\bar y,Q)\}.$$

В силу позитивности вхождения переменной Q в Φ оператор $\Gamma_{\Phi,\bar{y}}$ является монотонным, и, следовательно, имеет наименьшую неподвижную точку.

ТЕОРЕМА 5. Для любой Σ-формулы $\Phi(\bar{x},\bar{y},Q)$ сигнатуры σ^*, имеющей позитивное вхождение предикатной переменной Q, найдется Σ-формула $\Psi(\bar{x},\bar{y})$, определяющая наименьшую неподвижную точку оператора $\Gamma_{\Phi,\bar{y}}$ при любом фиксированном значении параметров.

Теорию Σ-рекурсии можно развивать в различных направлениях. Например, аналогично работе Ю.Л.Ершова [3] можно расширить логику первого порядка над списками, определив динамическую логику над списковыми расширениями, которая обладает теми же хорошими свойствами, что и динамическая логика над допустимыми моделями. Полезным может оказаться бесконечный вариант Σ-формул. Столь же естественно в GES, как и в [3], решается вопрос о понятии абстрактных типов данных и проблеме представимости одного типа данных в другом. Другими словами, предстоит серьезная работа по дальнейшему развитию теории GES.

§4. Σ-программы

В данном параграфе основное внимание будет уделено понятию Σ-программы. Ранее мы говорили о том, что в общем случае Σ-программа состоит из 2-х частей - логической и управленческой. Здесь же речь будет идти только о логической компоненте Σ-программ. Само понятие Σ-программы будет сформулировано не как синтаксическая категория, а на содержательном математическом уровне. Таким образом, синтаксические аспекты Σ-языка останутся за пределами данной статьи. Им будет посвящена в дальнейшем отдельная статья.

Пусть σ_0 и σ - сигнатуры, определенные в §3. Через σ^* обозначим обогащение сигнатуры σ предикатными и функциональными переменными. Будем говорить, что функциональная переменная F входит в формулу Φ позитивно, если позитивно входят все те элементарные формулы, в которых встречается переменная F.

Рассмотрим Σ-формулу $\Phi(\bar{x}; Q, F)$, где $\bar{x}$ - набор всех ее свободных переменных длины n; Q и F - наборы позитивно входящих в Φ предикатных и функциональных переменных. Пусть Q - n-местная предикатная переменная, а F - (n-1)-местный функциональный символ из $\sigma^* \setminus \sigma$.

ОПРЕДЕЛЕНИЕ 3. Выражения вида

$$\underline{\text{def}}\ Q(\bar{x}):\ \Phi(\bar{x}, Q, F), \tag{1}$$

а также вида

$$\underline{\text{def}}\ F(x_1,\ldots,x_{n-1}) = x_n:\ \Phi(\bar{x}, Q, F); \tag{2}$$

(при условии, что в последнем случае формула Φ определяет функциональное равенство) называются Σ - о п р е д е л е н и я м и (сигнатуры σ^*). Если набор $\bar{x}$ пустой, то допускаются выражения только вида (I) и соответствующие Σ-определения называются ф а к т а м и.

ПРИМЕР I[*].

а) $\underline{\text{def}}\ \underline{\text{FACT}}\ (x,y):\ [\exists z((x=0 \to y=1) \vee (\neg(x=0) \to$
$\to [(y=x \cdot z)\ \&\ \underline{\text{FACT}}(x-1,z)])];$

б) $\underline{\text{def}}\ x! = y:\ ((x=0 \to y=1) \vee (\neg(x=0) \to$
$\to y = x \cdot (x-1)!)).$

В этом примере оба Σ-определения представляют один и тот же предикат. Но если в случае "а" определением служила Σ-формула, то в случае "б" определением является уже Δ_0-формула.

В дальнейшем, как и в примере I, нам часто будет встречаться конструкция типа $(\Phi\ \&\ \Psi_1) \vee (\neg\Phi\ \&\ \Psi_2)$, где Φ, Ψ_1 и Ψ_2 - Δ-формулы. Мы будем ее обозначать в более привычных программистских терминах $\underline{\text{if}}\ \Phi\ \underline{\text{then}}\ \Psi_1\ \underline{\text{else}}\ \Psi_2$.

В Σ-определении мы будем различать левую часть определения, стоящую от ":" и называемую в дальнейшем з а г о л о в к о м, и правую, называемую т е л о м Σ-определения. Если Σ-определение является фактом, то его заголовок часто будет называться также у к а з а т е л е м.

ПРИМЕР 2. $\underline{\text{def}}\ \underline{\text{TRUE}}$: 0=0.

Рассмотрим теперь конечное множество $\mathcal{O}$ Σ-определений. Будем его называть п р а в и л ь н ы м, если

а) среди заголовков его Σ-определений не встречаются выражения с одинаковыми предикатными и/или функциональными символами;

б) наборы свободных переменных у Σ-определений попарно не пересекаются.

---

[*] В дальнейшем мы в качестве сигнатуры σ_0 рассматриваем сигнатуру элементарной арифметики.

Пусть $S_1(\bar{x}_1),\ldots,S_n(\bar{x}_n)$ - соответствующая Σ-определениям из множества $\mathcal{O}$ последовательность заголовков, а Q и F -наборы (воз - можно, пустые) предикатных и функциональных переменных, входящие в тела Σ-определений из $\mathcal{O}$, но не встречающиеся в левых частях Σ-определений.

ОПРЕДЕЛЕНИЕ 4. Выражение вида

$$\langle\text{идентификатор}\rangle\ (S_1(\bar{x}_1),\ldots,S_n(\bar{x}_n);\ Q;\ F):\mathcal{O}\ \underline{\text{end}}, \qquad (3)$$

где $\mathcal{O}$ - правильное множество Σ-определений, а $S_i(\bar{x}_i), i=\overline{1,n}$, Q и F - определены выше, называется Σ-с х е м о й (сигнатуры σ^*).

В выражении (3) ⟨ идентификатор ⟩ называется и м е н е м Σ-схемы, $S_i(\bar{x}_i)$, $i = 1,\ldots,N$, называются в х о д а м и схемы, $\bar{x}_j$, $j = 1,\ldots,n$, - ф о р м а л ь н ы м и п р е д м е т н ы м и п а р а м е т р а м и j-го входа, элементы Q (F) - п р е д и к а т н ы м и (функциональными) ф о р м а л ь н ы м и п а р а м е т р а м и.

Σ-схема называется з а м к н у т о й, если списки ее предикатных и функциональных формальных параметров пустые. В противном случае Σ-схема называется о т к р ы т о й.

ПРИМЕР 3 · (замкнутая Σ-схема):

$$\text{HOD}\ (\text{GCD}(x,y,z),\text{MIN}(x_1,y_1)=z_1,\ \text{DIV}(x_2,y_2),\ x_3 \le y_3,\ x_4 > y_4):$$

$$\underline{\text{def}}\ \text{GCD}(x,y,z):(x>0\ \&\ y>0\ \&\ \underline{\text{DIV}}(z,x)\ \&\ \underline{\text{DIV}}(z,y)\ \&$$

$$\&(\forall w \le \underline{\text{MIN}}(x,y))(\underline{\text{DIV}}(w,x)\ \&\ \underline{\text{DIV}}(w,y) \to w \le z));$$

$$\underline{\text{def}}\ \underline{\text{MIN}}(x_1,y_1) = z_1:\ \underline{\text{if}}\ x_1 \le y_1\ \underline{\text{then}}\ z_1 = x_1\ \underline{\text{else}}\ z_1 = y_1;$$

$$\underline{\text{def}}\ \text{DIV}(x_2,y_2):(x_2 > 0\ \&\ y_2 > 0\ \&\ \exists z(z * x_2 = y_2));$$

$$\underline{\text{def}}\ x_3 \le y_3:\ \exists z(x_3 + z = y_3);$$

$$\underline{\text{def}}\ x_4 > y_4:\ [(y_4 \le x_4)\ \&\ \neg(x_4 = y_4)];$$

$$\underline{\text{end}}.$$

Заметим, что в определении $\underline{\text{MIN}}$ можно было бы привести еще более недетерминистское определение

$$\underline{\text{def}}\ \underline{\text{MIN}}(x_1,y_1)=z_1:\ ((x_1 \le y_1 \to z=x_1) \vee (y_1 \le x_1 \to z_1 = y_1)).$$

Следующее замечание, касающееся примера 3, связано с тем, что определение наибольшего общего делителя можно было бы свести к единственному Σ-определению. Но, во-первых, нам хотелось проде -

монстрировать идею структуризации знаний о "мире" и роль Σ-схем в этой структуризации, а, во-вторых, Σ-схема со многими входами, как мы убедимся позже, является более богатой и многоцелевой структурой по сравнению с одновходовой.

Из теоремы 5 получаем следующий результат, имеющий непосредственное отношение к исполнению Σ-программ Σ-машиной.

ТЕОРЕМА 6. Пусть J – замкнутая Σ-схема. Тогда J определяет монотонный оператор Γ_J и наименьшая неподвижная точка этого оператора является Σ-определимым множеством.

Пусть $J(S_1(\bar{x}_1);\ldots;S_n(\bar{x}_n);\ldots)$ – некоторая Σ-схема со входами $S_i(\bar{x}_i)$, $i = 1,\ldots,n$. Выражение вида $[\bar{t}=:x]J.S_i$, где $\bar{t}$ – набор термов сигнатуры σ^*, в котором число членов не меньше числа элементов набора $\bar{x}_i$, называется вызовом Σ-схемы J в точке S_i с фактическими параметрами $\bar{t}$. Далее мы разрешаем встречаться в Σ-формулах вызовам Σ-схем, рассматривая последние как атомарные формулы, при условии, что они входят в Σ-формулы позитивно.

ПРИМЕР 4.

```
УПОР (ORDLIST(α), FALSE, TRUE):
 def ORDLIST(α): if ¬S¹(α) then FALSE
 else[(∀x ∈ α)[¬S¹(x)&  if (tail(α) = nil) then TRUE
     else (∀β ⊆α)(cons(β,x) ∈ α → [x, head(β),
     head (β)=: x1,y1,z1] HOD . MIN);
 def FALSE:  O=1;
 def TRUE: O=O;
          end.
```

Заметим, что можно привести рекурсивную Σ-схему, определяющую предикат ORDLIST:

```
УПОР1 (ORDLIST1(α), FALSE, TRUE):
   def ORDLIST1(α): if ¬S¹(α) then FALSE
      else[(∀x ∈ α)[¬S¹(x) & if (tail(α) = nil)
       then TRUE
        else (∀β ⊆ α)(ORDLIST(β) &
```

& [head(β), head(α)=: x_4, y_4] HOD . ≤)]);

def FALSE: O=1;

def TRUE: O=0; end.

Расширив понятие Σ-определения (разрешая в Σ-формулах ссылаться на Σ-схемы), мы тем самым расширили и понятие самой Σ-схемы. Это и позволит нам ввести понятие Σ-программы.

Пусть J (открытая)-Σ-схема и Q-ее формальный n-местный предикатный параметр (случай с функциональным параметром рассматривается аналогично). Рассмотрим некоторую Σ-схему J1 и ее вход $S_i(\bar{x})$ с формальными параметрами $\bar{x} = \langle x_1, \ldots, x_n \rangle$. Запись вида $\langle$[J1.$S_i(\bar{x})$=: Q]J; J1 $\rangle$, обозначающая подстановку вместо каждого вхождения Q в J выражения J1.$S_i(\ldots)$, называется J1-к о н к р е т и з а ц и е й Σ-схемы J по формальному параметру Q. Будем говорить, что Σ-схема J к о н к р е т н а, если все ее формальные и функциональные символы конкретизированы.

ОПРЕДЕЛЕНИЕ 5. Конкретная Σ-схема J называется Σ-п р о г р а м м о й, если все Σ-схемы, участвующие в ее конкретизации и называемые Σ-подпрограммами J, также являются конкретными Σ-схемами.

Из определения следует, например, что каждая замкнутая Σ-схема является Σ-программой. И опять-таки, как и в случае замкнутых Σ-схем, можно показать, что верна

ТЕОРЕМА 7. К а ж д а я Σ-п р о г р а м м а о п р е д е л я е т м о н о т о н н ы й о п е р а т о р, н а и м е н ь ш а я н е п о д в и ж н а я т о ч к а к о т о р о г о Σ-о п р е д е л и м а.

Говоря о том, что J - Σ-программа, мы тем самым подчеркиваем, что рассматривается не только Σ-схема J, а весь комплекс конкретных Σ-схем, участвующих в ее определении. Некоторые из этих Σ-схем могут быть с т а н д а р т н ы м и (встроенными) конструкциями Σ-языка. Конкретный набор стандартных Σ-программ во многом определяет удобство и проблемную направленность Σ-языка.

Инициализация Σ-программы J происходит следующим образом. Пусть $S_i(\bar{x}_i)$ - некоторый вход в J (в том числе и вход в любую ее подпрограмму). Предполагается, что все имена Σ-программ, участвующих в определении Σ-программы, являются уникальными. Пусть In и Out - служебные слова, обозначающие входные и выходные данные. Разобьем список $\bar{x}_i$ на два непересекающихся подсписка $\bar{y}$ и $\bar{z}$. Тогда

запись

$$\langle \underline{\text{In}}{:}\bar{a}{=:}\bar{y}{:}\ \underline{\text{Out}}{:}\bar{z};\ J.S_i\rangle$$

или

$$\langle \underline{\text{In}}{:}\bar{a}{=:}\bar{y};\ \underline{\text{Out}}{:}\bar{z};\ J.J'.S_i\ \rangle,$$

где J' – та Σ-подпрограмма, которой принадлежит вход S_i, называется **обращением** к Σ-программе J.

Мы говорим, что Σ-программа J **вычисляет** значения переменных из $\bar{z}$ в модели $\mathcal{A}(\mathcal{M})$ при обращении $\langle \underline{\text{In}}{:}\bar{a}{=:}\bar{y};\ \underline{\text{Out}}{:}\bar{z};\ J.S_i\ \rangle$, если найдется набор элементов $\bar{b}\in|\mathcal{A}(\mathcal{M})|$ такой, что

$$\mathcal{A}(\mathcal{M}) \models \Phi_{\Gamma_{\Phi_i}}(\bar{a},\bar{b}),$$

где Φ_i – соответствующая Σ-формула в определении S_i, Γ_{Φ_i} – соответствующий этой формуле монотонный оператор, а $\Phi_{\Gamma_{\Phi_i}}$ – определение его наименьшей неподвижной точки. О том, как происходит вычисление Σ-программ, будет идти речь в последующих публикациях.

Заканчивая статью, хотелось бы отметить, что авторы не ставили перед собой цель исчерпывающего изложения концепции Σ-программирования. Цель была более простой – продемонстрировать теоретическую возможность построения языка и соответственно стиля программирования на новых, логических основаниях, отличных от традиционных, общепринятых. Предстоит большая работа по дальнейшему развитию идей Σ-программирования, превращение их в практически работающий аппарат.

Хотелось бы еще раз обратить внимание читателей, что все вышеприведенные построения (Σ-определения, Σ-схемы, Σ-программы) имеют списковую структуру и потому сами по себе могут рассматриваться как данные Σ-языка.

Таким образом, Σ-язык является языком не только для построения Σ-программ, но и языком, в котором можно преобразовывать эти программы, исследовать их свойства, описывать формальные конструкции, возникающие при проектировании Σ-программ и т.п. Σ-язык является языком высокого уровня, ориентированным на работу со сложно логически-структурированной информацией.

Литература

I. BARWISE J. Admissible sets and structures.- Berlin: Springer-Verlag,1975.- 393 S.

2. ЕРШОВ Ю.Л. Принцип Σ-перечисления. -Докл. АН СССР, 1983, т.270, № 5, с.786-788.

3. ЕРШОВ Ю.Л. Динамическая логика над допустимыми множествами. -Докл. АН СССР, 1983, т.273, №5, с.1045-1048.

Поступила в ред.-изд.отд.
25 февраля 1985 года