

Введение

В ИТМ и ВТ в течение ряда лет проводились научно-исследовательские работы в области построения архитектурной и технической базы для создания вычислительных машин с высоким параллелизмом вычислений. Эти работы проводились коллективом сотрудников Института под руководством А.М.Степанова и В.И.Смирнова. Наиболее сложной задачей, как впрочем и ожидалось, оказалась выработка единого подхода, учитывающего как архитектурные требования, так и требования технической реализации. Выявились не только трудности, связанные с выбором элементной базы и конструкции, но и принципиальные вопросы, касающиеся последующей эксплуатации аппаратуры, в частности, диагностики, ремонта и т.п. Традиционного в таких случаях компромисса между архитектурой и технической реализацией достигнуть не удалось. Тем не менее представляется целесообразным опубликовать ряд результатов, относящихся к этой работе и имеющих самостоятельное научное и техническое значение. В данном пропринте приводится описание абстрактной сетевой машины (AMC), разработанной под руководством А.М.Степанова. В последующем предполагается изложение В.И.Смирновым оригинальных технических решений, обеспечивающих исполь-

зование элементной базы с предельными частотными характеристиками.

AMC – это вычислительный механизм высокого уровня ("семантический" уровень представления информации, вычислительный механизм близкий к логическому выводу), обладающий большой степенью внутреннего параллелизма и рядом полезных свойств, объединяющих в себе различные положительные черты, свойственные другим вычислительным моделям.

Работа над AMC ведется уже много лет. За это время на МВК "Эльбрус" была реализована экспериментальная система программирования, в основе которой лежит программируемая модель AMC и входной непроцедурный язык ПАРС. Накоплен большой экспериментальный материал.

В разд. I излагаются некоторые идеальные предпосылки этой работы. В разд. 2 дается описание AMC и обсуждаются основные свойства модели. В приложении приводятся наметки математического аппарата и дается формальное доказательство некоторых важных свойств машины.

Член-корреспондент АН СССР

Г.Г. Рябов

I. О ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЯХ

В настоящее время достаточно очевидным представляется тот факт, что дальнейшее повышение эффективности обработки информации уже невозможно без перехода к параллельным вычислениям и без построения соответствующих аппаратных структур, способных к высокопараллельной обработке данных. Речь идет о переходе к так называемому массовому параллелизму, характеризующемуся сотнями и тысячами элементарных операций ЭВМ, выполняемых одновременно. Уже разработан и успешно эксплуатируется ряд вычислительных систем, про которые можно сказать, что они до некоторой степени обладают массовым параллелизмом, – это векторные и матричные ЭВМ. В этих вычислительных системах во многом сохраняется традиционный принцип последовательной обработки данных, однако регулярный характер обрабатываемых структур и их высокая размерность (длина вектора, порядок матрицы) позволяют либо использовать SIMD принцип, выполняя одну и ту же операцию одновременно над всеми элементами вектора или матрицы, либо построить высокоскоростную конвейерную аппаратную структуру из многих исполнительных устройств. Высокий коэффициент загрузки конвейера достигается за счет многократной повторяемости некоторой последовательности операций, выполняемой покомпонентно над элементами нескольких векторов. Несмотря на ограниченность такого подхода, который не решает проблему параллельных вычислений

в целом, направление это бурно развивается и, без сомнения, будет развиваться и в дальнейшем, так как оно "покрывает" весьма широкий и практически важный класс задач (задачи линейной алгебры, сеточные методы решения задач математической физики и др.). Если же говорить о более широком классе задач, включающем скалярную обработку данных, а также обработку произвольных сложных структур (деревья, списки, графы, сети и др.), то переход к массовому параллелизму диктует необходимость в более революционных шагах и в пересмотре ряда принципов вычислений. Мнения многих специалистов сходятся на том, что такой шаг невозможен без отказа от традиционной архитектуры ЭВМ, известной как архитектура фон-Неймана. Многие исследователи /1,2,3/ считают именно фон-неймановский принцип вычислений тем "узким местом", которое тормозит дальнейшее развитие вычислительной техники и программирования в сторону повышения параллелизма вычислений.

Второе направление, которое хотелось бы здесь отметить, относится к вопросам общения человека с машиной в процессе постановки и решения задач на ЭВМ и, в частности, к развитию языков программирования. Имеется ввиду тенденция к повышению уровня языков программирования. Что такое уровень языка? Принято считать, что уровень языка программирования тем выше, чем с меньшей степенью детализации программист описывает вычислительный процесс, протекающий в реальной аппаратуре, и чем в меньшей сте-

пени он конкретизирует те вычислительные ресурсы, которые для этого используются. Ясно, что понятие "язык высокого уровня" есть понятие относительное. Традиционно к языкам высокого уровня относятся такие языки, как ФОРТРАН, АЛГОЛ 60, Паскаль, ПЛ/И и др. Все эти языки являются процедурными языками программирования. Несмотря на высокий уровень абстракции, применяемый во многих из них для описания типов данных, а также на разнообразные средства управления и структуризации программы, все они служат в конечном итоге для задания последовательности действий, выполняемых на некоторой абстрактной машине (Алгол-машине, Паскаль-машине и т.д.) для решения данной задачи, то есть для описания алгоритма решения этой задачи. В последнее время все большее развитие получают языки качественно более высокого уровня, а именно языки непроцедурные. К таким языкам относятся в первую очередь базовый (или "чистый") ЛИСП и ПРОЛОГ. Обсуждая свойства языка ПРОЛОГ, Робинсон /4/ вводит более общее понятие языка утверждений (statement language) - языка, программа на котором есть неупорядоченный набор утверждений об объектах предметной области, высказанных в терминах некоторой формальной системы (не обязательно связанной с исчислением предикатов), а вычислительный процесс - процесс вывода на основе правил вывода, принятых в данной формальной системе. Появление непроцедурных языков - подлинно революционный шаг, качественно повышающий уровень языков программирования, так как он при-

водит к отказу от понятия алгоритма. Такой язык, пожалуй, уже не следовало бы называть языком программирования, а скорее языком описания или даже постановки задач. Следует отметить, однако, что реальная практика в области непроцедурных языков еще далека от идеала и поэтому следует говорить именно о тенденции развития языков, а не о совершившейся революции в этой области.

Рассмотрим теперь третью линию, которая в последнее время наметилась в развитии вычислительной техники — это линия интеллектуализации ЭВМ. Наиболее ярко она проявилась в японском проекте вычислительных машин 5-го поколения. Речь идет о предполагаемом существенном расширении круга пользователей ЭВМ с включением в него лиц, далеких от вычислительной техники и программирования. В этих условиях первостепенное значение приобретают задачи искусственного интеллекта (ИИ). Это распознавание и "понимание" зрительных образов (для облегчения общения с ЭВМ на уровне понятной любому неподготовленному пользователю зрительной информации), распознавание и синтез звучащей речи, понимание текстов на естественном языке (роль такого рода информации для неподготовленного пользователя особенно очевидна), автоматическое решение задач (problem solving), использование экспертных систем, основанных на "знаниях" и т.п. Таким образом, возникла задача внедрения в промышленно выпускаемое программное обеспечение и аппаратуру "интеллектуальных" программ и механизмов, до сих пор рассматривавшихся как чисто экспериментальные.

Процесс этот уже активно идет (достаточно указать на бум в области экспертных систем или на широко распространявшиеся за рубежом системы восприятия и синтеза речи).

Наконец, отметим те тенденции, которые существуют в области технологии проектирования и изготовления элементов и конструктивных единиц ЭВМ. Техническая база современных вычислительных средств связана с микросхемами средней и большой степеней интеграции (СИС и БИС). Уровень их развития в настоящее время может характеризоваться такими показателями, как плотность, достигающая значения десятков и сотен тысяч вентилей на один кристалл. В связи с дальнейшим совершенствованием технологии, схемотехники и конструкции ожидается повышение этих параметров на порядок уже в ближайшие годы. Это означает, что в ближайшее время будут практически доступны огромные аппаратные ресурсы.

Очень важным обстоятельством является то, что эти различные "течения" взаимно дополняют и обусловливают друг друга, слияясь в одно "русле", по которому движется в своем развитии вычислительная техника и программирование.

Действительно, непроцедурные языки программирования возникли первоначально как языки искусственного интеллекта и это их основная сфера применения, поэтому интеллектуализация ЭВМ невозможна без реализации непроцедурных языков. В то же время практика показала, что реализация непроцедурных языков на традиционных маши-

нах фон-неймановского типа сильно затруднена из-за огромного семантического разрыва между языком и аппаратурой. С другой стороны, программам, написанным на этих языках, присущ значительный внутренний параллелизм, так как они не содержат описания каких-либо последовательностей действий, а тем самым в них отсутствует понятие времени. В частности, процесс логического вывода может, в принципе, производиться в параллель. Заметим также, что в настоящее время существенным фактором, ограничивающим развитие и практическое применение методов ИИ, является недостаточная вычислительная мощность ЭВМ, необходимая для решения большинства серьезных задач ИИ, имеющих огромную трудоемкость. Ряд исследователей считает, что качественный скачок в развитии ИИ возможен только путем перехода к аппаратным структурам, способным к массовому параллелизму вычислений /5/.

Итак, мы видим, что задачи интеллектуализации ЭВМ и необходимость в эффективной реализации непроцедурных языков естественным образом диктуют нам необходимость в переходе к аппаратным архитектурам не фон-неймановского типа, способным к высокопараллельной обработке информации. Однако существует и менее очевидная связь в обратном направлении: переход к массовому параллелизму в аппаратуре (независимо от степени "интеллектуальности" задачи) требует качественного повышения уровня языка программирования в сторону непроцедурного программирования (случай регулярного параллелизма - векторных и

матричных вычислений рассматриваться не будет).

В связи со сложностью параллельного режима в случае нерегулярного массового параллелизма неизбежно возникает задача: освободить программиста от необходимости представлять себе этот параллельный вычислительный процесс, дать ему возможность составлять программу, в значительной степени абстрагируясь от того факта, что она будет выполняться параллельно. Само же "распараллеливание" должно происходить автоматически на этапе трансляции или же в процессе ее выполнения. Другими словами, необходимо повысить уровень языков программирования в той их части, которая связана с параллелизмом.

Можно ли решить эту задачу, оставаясь в рамках традиционного процедурного программирования? Существует целое направление, призванное это сделать - автоматическое распараллеливание последовательных программ. Это своего рода "обходной путь" - как повысить уровень языка, не повышая его. Программирование ведется "по струнке", например, на ФОРТРАНЕ. Программа пропускается через специальную систему распараллеливания, которая преобразует последовательную программу в параллельную, выявляя в ней "естественный параллелизм, присущий данной задаче". Ограниченностю такого подхода очевидна. На самом деле, тот естественный параллелизм, который был присущ данной задаче до того, как она была запрограммирована на ФОРТРАНЕ, и отражал ее физический смысл, после программирования был заменен на естественный параллелизм, присущий данной конкретной последовательной про-

грамм (то есть какие операторы можно выполнять параллель), а это далеко не одно и то же. Тем не менее, выявить ограниченный "фортрановский" параллелизм можно (хотя и это далеко не простая задача) и поэтому автоматическое распараллеливание последовательных программ имеет несомненную практическую ценность, в особенности когда речь идет о переносе на параллельную машину старых последовательных программ, уже накопленных к этому времени (подробнее об автоматическом распараллеливании см. разд. II, п. 8.4). Если же имеется в виду составление новых программ, специально предназначенных для параллельного выполнения на аппаратуре, способной к массовому параллелизму, то писать их надо на таком языке, который максимально сохраняет естественный параллелизм задачи. Такой язык должен быть как можно более высокого уровня, то есть как можно ближе к языку постановки задачи. В наибольшей степени удовлетворяют этому требованию непроцедурные языки. Подчеркнем еще раз, что о степени "интеллектуальности" исходной задачи ничего не говорилось, следовательно, для эффективного использования естественного параллелизма, присущего даже обычным вычислительным задачам (а не только задачам ИИ), необходимо более "интеллектуальное" их программирование. Необходимость в "интеллектуальном" программировании "неинтеллектуальных" задач уже существует сама по себе в рамках традиционных ЭВМ как одно из средств выхода из кризиса программирования. Примером удачной работы в этом направлении являются ре-

зультаты Тытугу /6/. Задача описывается на непроцедурном языке УТОПИСТ, а затем система синтезирует по этому описанию программу (в данном случае для традиционной последовательной машины). Другой пример, относящийся уже к векторным вычислениям, — это язык НОРМА — непроцедурный язык для описания "сеточных" задач математической физики.

Таким образом, можно сделать следующий вывод: переход к массовому параллелизму, отказ от фон-неймановской архитектуры ЭВМ, качественное повышение уровня языка программирования в сторону непроцедурного программирования, интеллектуализация ЭВМ — все это как бы разные стороны одного процесса на пути создания ЭВМ нового поколения.

Следует также отметить, что тенденции в развитии технологии и схемотехники идут в одном русле с рассматриваемым направлением, так как, с одной стороны, значительное увеличение аппаратных ресурсов позволяет закладывать в аппаратуру более "интеллектуальные" механизмы, а с другой стороны, интересы эффективного производства и использования систем на основе БИС сами наталкивают разработчиков на архитектуры, основанные на взаимодействии большого числа однородных функциональных элементов, работающих параллельно.

К перечисленным направлениям в вычислительной технике и программировании примыкает еще одна область исследований, связь которой с общей тенденцией развития в настоящее время еще недостаточно осознается. Имеется

ввиду теория смешанных вычислений, разработанная под руководством академика А.П.Ершова /7/.

Теория смешанных вычислений предусматривает возможность частичных вычислений по некоторой программе, когда не все ее исходные данные определены. Программа - это алгоритм, записанный на некотором языке программирования. Для того, чтобы провести "обычные" вычисления по этой программе (при полностью заданных входных величинах), нужно применить "обычный" вычислитель, то есть интерпретатор данного языка, реализованный тем или иным способом, в том числе, может быть, аппаратно. Если же входные данные определены неполностью, то в этом случае можно реализовать так называемый смешанный вычислитель. Работа смешанного вычислителя напоминает работу обычного интерпретатора, отличаясь от нее тем, что включает в себя деятельность двух видов (отсюда слово "смешанный"). Во-первых, это обычные вычисления, проводимые в соответствии с программой в тех случаях, когда заданных входных данных для этого достаточно. Во-вторых, это деятельность, связанная с составлением остаточной программы, в которую заносятся специальным образом формируемые отложенные операторы, получаемые из тех операторов исходной программы, которые в силу неполноты определенных входов не могли быть выполнены или же выполнились только частично. Сформированная таким обра-

зом остаточная программа в качестве своих входов имеет те входные данные исходной программы, которые первоначально не были определены (или, как принято говорить, были "заморожены"). Если на выходе обычного вычислителя (интерпретатора) при полностью заданных входах интерпретируемой программы имеется результат работы программы, то на выходе смешанного вычислителя получается некоторая новая программа (остаточная программа). После подачи на ее вход недостающих входных данных она завершает ту обработку информации, которая предусматривалась в исходном алгоритме, но была на этапе смешанных вычислений выполнена только частично.

Заметим, что если все входные данные были определены, то результатом смешанных вычислений будет пустая остаточная программа, а исходный алгоритм будет выполнен полностью, то есть в этом случае работа смешанного вычислителя совпадет с работой обычного интерпретатора. Таким образом, смешанные вычисления можно считать обобщением обычных вычислений. Введение такого обобщения оказалось чрезвычайно плодотворным и позволилольному взглянуть на многие понятия программирования.

Рассмотрим пример. Пусть в исходной программе записан алгоритм вычисления $Z = f(X, Y)$, где X и Y - исходные данные, а Z - результат. Подадим на вход программы величину $X = a$, а переменную Y заморозим. После проведения смешанных вычислений получаем остаточную программу для вычисления $Z = g(Y)$, где $g(Y) = f(a, Y)$. Подавая на вход остаточной программы $Y = b$, по-

на языке L_1 и программа Π^I на языке L_2 выполняют одно и то же преобразование информации, и, следовательно, программу Π^I можно рассматривать как результат трансляции программы Π с языка L_1 на язык L_2 . Итак, оказалось, что трансляция – это результат частичной интерпретации при замороженных исходных данных.

В дальнейшем А.П. Ершов ввел понятие трансформаций программ и трансформационной машины /12/. Идея состоит в следующем. Для данного конкретного языка вводится система трансформаций. Каждая трансформация программы состоит в замене некоторой синтаксической конструкции на другую (более простую). Возможность трансформации того или иного типа определяется соответствующим правилом. Примеры трансформаций: конструкция $3+5$ заменяется на 8 ; конструкция если истина то $X:=1$ иначе $X:=2$ заменяется на $X:=1$ и т.д. Каждая трансформация не только производит некоторое вычисление по программе, но и одновременно преобразует (упрощает) саму программу. Если исходные данные были полностью определены, то при помощи последовательности трансформаций программа редуцируется к единственному оператору вывода (вычисленного) результата. Если же часть входных величин была заморожена, то на некотором шаге ни одно из правил трансформаций не удается применить и состояние программы в этот момент оказывается остаточной программой, готовой к продолжению вычислений после доопределения входов. Таким образом, в трансформационной машине не существует разницы между обычными и смешан-

ними вычислениями. Заметим также, что работе трансформационной машины (состоящей в проверке применимости правил и выполнении возможных трансформаций) присущ определенный внутренний параллелизм, так как все возможные в данном состоянии программы трансформации можно выполнить в любом порядке, а многие из них – параллельно. К сожалению, когда речь идет о трансформациях последовательных программ, основанных на фон-неймановских вычислениях с оператором присваивания, проявляются весьма сложные трансформации, носящие глобальный характер. Так, например, если в программе имеется оператор присваивания $X := 1$, то возможна трансформация программы, состоящая в замене всех вхождений имени X на константу 1 в некоторой части программы, являющейся "областью действия" этого оператора. Выделение области действия нетривиально и требует анализа информационно-логических связей в программе. Значительно естественнее выглядит трансформационный принцип вычислений, когда он применяется к функциональным рекурсивным программам. Из-за отсутствия понятия общей памяти и операторов присваивания трансформации приобретают чисто локальный характер. Процесс трансформаций фактически совпадает с известным в функциональном программировании процессом редукционных вычислений /2/.

Нам представляется, что результаты, полученные в теории смешанных вычислений, затрагивают фундаментальные свойства вычислений и органично вписываются в русло рассматриваемого направления. Полностью осоз-

го" синтеза, оптимизации и распараллеливания программ.

Проблема параллельных вычислений имеет множество различных аспектов, и существует ряд различных направлений, так или иначе связанных с параллельными вычислениями.

Эту проблему часто понимают как проблему распараллеливания, при этом в сочетании слов "параллельные вычисления" как бы делается ударение на первом слове, то есть считается, что "вычисления" - это то, что мы хорошо понимаем, а вот как их сделать "параллельными", как их распараллелить - в этом-то и состоит проблема. Приходится встречаться даже с таким крайним мнением, что в сущности никакой проблемы не существует, так как каждый алгоритм изначально содержит в себе те или иные возможности параллельного выполнения, поэтому мы имеем воего лишь техническую задачу - выявить и реализовать тот параллелизм, который либо есть, либо его нет, и тогда уже ничего не предпринимаешь. В этом рассуждении за исходный пункт взято понятие алгоритма и в этом-то и состоит ошибка. Нам представляется, что "ударение" следует поставить именно на втором слове "вычисления", то есть для решения проблемы параллельных вычислений нужно прежде всего понять, какими должны быть собственно вычисления, чтобы они были параллельными. Эта позиция требует некоторого разъяснения. Дело в том, что в математике существует понятие вычислимости, причем разработано несколько различных механизмов реализации вычислимости в

виде "абстрактных машин", одной из которых является машина фон-Неймана. Есть и другие, например, машина Тьюринга или нормальные алгоритмы Маркова. В последнее время появилось несколько новых механизмов реализации вычислимости, к ним можно отнести, например, "ПРОЛГ-машину", "Рефал-машину" /17/, "Клубные системы" /18/ и описанную в настоящей работе сетевую машину АМС /19,20/. В связи с этим существует постановка проблемы параллельных вычислений как фундаментальной теоретической проблемы выбора, а точнее, разработки нового, нетрадиционного механизма реализации вычислимости, который, в свете приведенных выше рассуждений, должен обладать, кроме присущего ему параллелизма, также "непроцедурностью" и "интеллектуальностью". Мы считаем, что это центральная проблема в области параллельных вычислений, наименее разработанная на современном этапе.

II. АБСТРАКТНАЯ СЕТЕВАЯ МАШИНА

I. Представление информации. Простая сеть

Основой представления информации в АМС является ассоциативная сеть - ориентированный граф (с помеченными ребрами) особого вида. Узлы сети будем называть объектами. Объекты бывают трех типов: а) простые или атомы; б) сложные и в) пустые (рис. I). Атомы изобра-

жены на рис. I в виде кружков. С каждым атомом связано его изображение (называемое также его описанием), показанное на рис. I внутри соответствующего кружка. Из узлов-атомов не может выходить стрелок. С каждым сложным объектом связано мно-

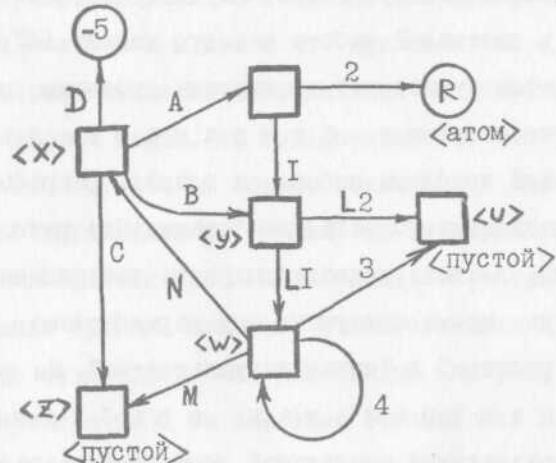


Рис.1. Ассоциативная сеть

жество его атрибутов, называемое описанием сложного объекта. Каждый атрибут имеет вид (A, X) , где A – это имя атрибута (некоторый идентификатор или положительное целое число), X – значение атрибута (это некоторый объект сети). На рис. I сложные объекты показаны в виде прямоугольников с выходящими из них стрелками – атрибутами. Имя атрибута написано на стрелке, а объект, соответствующий значению атрибута, находится на конце стрелки. Пустые объекты – это объекты, имеющие "пустое" описание. На рис. I пустые объекты показаны в виде пря-

моугольников, из которых не выходит ни одной стрелки. Пустой объект нельзя отнести ни к простым, ни к сложным объектам, о нем просто ничего не известно, кроме того, что он существует. Имеется следующее ограничение на вид описания сложного объекта: описание сложного объекта не может содержать одноименных атрибутов (то есть пометки на стрелках, выходящих из какого-либо узла сети, должны быть все различны). Никаких других ограничений нет, в частности, допускаются циклические структуры и петли.

Атомы назовем статическими объектами сети, а сложные и пустые объекты динамическими (на рис. I статические объекты – кружки, динамические – прямоугольники).

Информация, заключенная в угловые скобки (рис. I), является комментарием, относящимся к содержательной интерпретации объектов.

Все изложенное можно считать неформальным определением простой сети. В дальнейшем определение будет расширено.

2. Основные принципы представления информации в виде простой сети

Все изложенное в настоящем разделе относится к вопросам содержательной интерпретации простой сети и ничего не добавляет к ее определению как формального объекта.

2.1. Атомы

Атомами представляются те "реальные объекты" (объекты предметной области), которые на данном уровне рассмотрения не имеют внутренней структуры. Атомами могут быть вещественные и целые числа, символы, цепочки символов, истинностные значения (истина и ложь или соответственно Т и F) и т.п. В принципе, в машине достаточно иметь только два атома - Т и F (или 1 и 0). Тогда символы, числа и т.д. могут быть представлены своими двоичными кодами, которые окажутся сложными объектами (рис.2). Атом однозначно определяется своим изображением. Два различных атома, имеющие

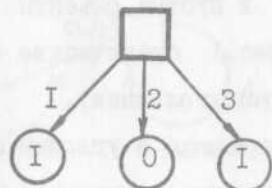
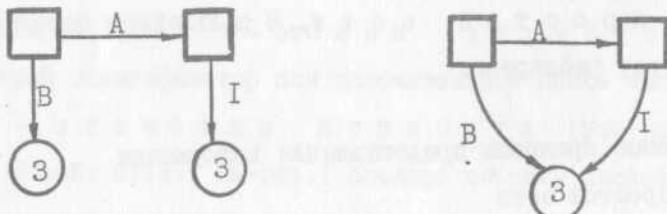


Рис.2. Двоичное представление числа 5



одинаковые изображения, можно не различать между собой (рис.3).

2.2. Интерпретация атрибутов сложных объектов

Атрибуты сложных объектов могут интерпретироваться различным образом. Приведем примеры подобных интерпретаций:

а) атрибут (A,У) объекта X - это свойство объекта X, где A - имя свойства, У - значение свойства. Описание сложного объекта - это его список свойств (рис.4);

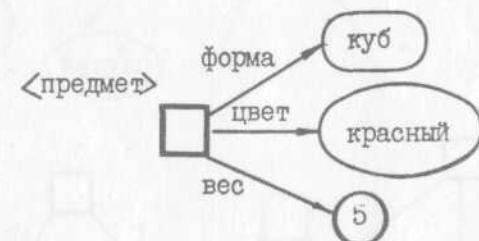
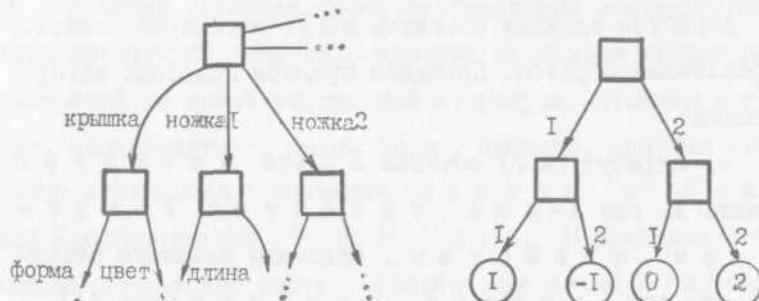


Рис.4. Атрибуты-свойства

б) атрибут (A,У) объекта X - это отношение структурного подчинения объекта У объекту X. У - это элемент структуры X, А - это "локальное имя" (или номер) элемента У в структуре X (рис.5,а,б,в,г);

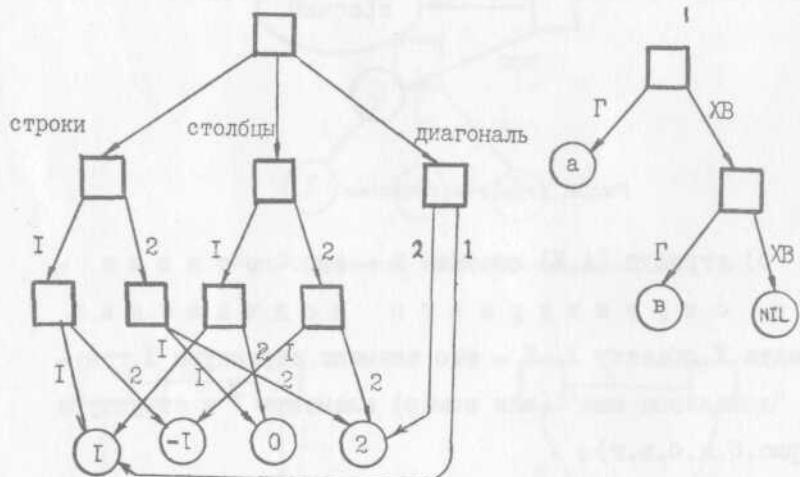
в) сложный объект изображает в сети факт существования некоторого отношения между теми объектами, в которые из него ведут стрелки. Атрибут (A,У) такого объекта означает, что У - это один из аргументов отношения, причем А - это имя (или номер) этого аргумента (рис.6,а,б,в).

Существуют и другие способы интерпретации, связанные с такими терминами, как параметр, актант, роль, слот



а

б

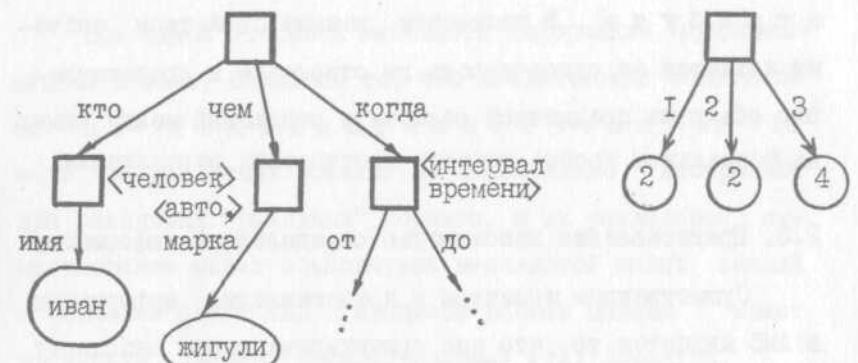


в

г

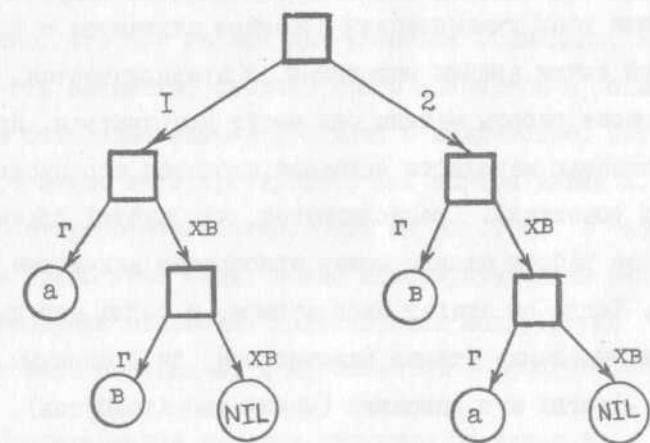
Рис.5. Структурные объекты:

а - стол; б - матрица $\begin{bmatrix} 1 & -1 \\ 0 & 2 \end{bmatrix}$; в - другое представление той же матрицы ; список (а в)



а

б



в

Рис.6. Отношения:
а - факт владения; б - отношение СУММА; в - отношение реверсирования

и т.п. С точки зрения АИС все эти способы интерпретации неразличимы, поскольку всех их объединяет одно

свойство (которое только и используется в механизмах работы АМС), а именно: имя атрибута однозначно определяет значение атрибута. В частности, важным свойством системы является ее однородность по отношению к представлению объектов предметной области и отношений между ними: на формальном уровне объекты и отношения неразличимы.

2.3. Представление неполнотой определенной информации

Существенным моментом в представлении информации в АМС является то, что оно ориентировано на неполноту этой информации. В АМС определенными считаются только атомы (поэтому их и назвали статическими объектами). Описания всех динамических объектов считаются с формальной точки зрения неполными, и предполагается, что в процессе работы машины они могут дополняться. Крайней степенью неполноты описания является его пустота (в пустых объектах). Предполагается, что пустой объект в процессе работы машины может приобрести некоторое описание. Тогда он станет либо атомом, и тогда его дальнейшее изменение станет невозможным, либо сложным объектом – тогда его описание (множество атрибутов) может продолжать уточняться путем добавления новых атрибутов и уточнения значений "старых". Разумеется, в сети могут присутствовать и полностью определенные (с точки зрения программиста) сложные объекты, однако с точки зрения АМС эти объекты ничем не отличаются от неполностью определенных, и машина работает с ними по од-

ним и тем же правилам. Таким образом, вопрос о полноте описания сложного объекта вынесен за пределы АМС и имеет смысл только на уровне интерпретации.

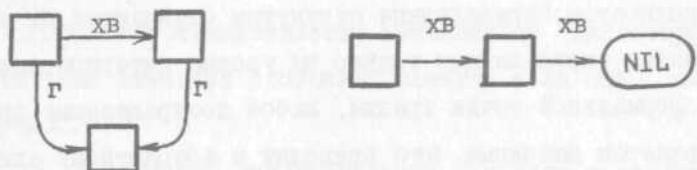
Еще одной стороной неполноты информации, представленной в сети, является то, что динамические объекты сети не индивидуализированы, то есть два различных объекта не обязательно изображают два различных "реальных" объекта, и их раздельное существование может объясняться неполнотой наших знаний о реальной ситуации. В процессе работы машины может произойти слияние (отождествление) этих объектов.

Рассмотрим примеры (рис.7). Сеть, показанную на рис.7, а, можно интерпретировать как список, про который известно, что его первые два элемента совпадают, каковы же эти элементы, сколько всего элементов в списке, каковы остальные элементы списка – неизвестно. Сеть на рис.7, б можно интерпретировать как список длины 2, элементы которого неизвестны. Сеть на рис.7, в, точнее, объект ТРАНС этой сети, можно интерпретировать как факт существования отношения транспозиции между двумя матрицами 2-го порядка M1 и M2. Элементы матриц неизвестны.

2.4. Представление типовых структур объектов и отношений. Фреймы

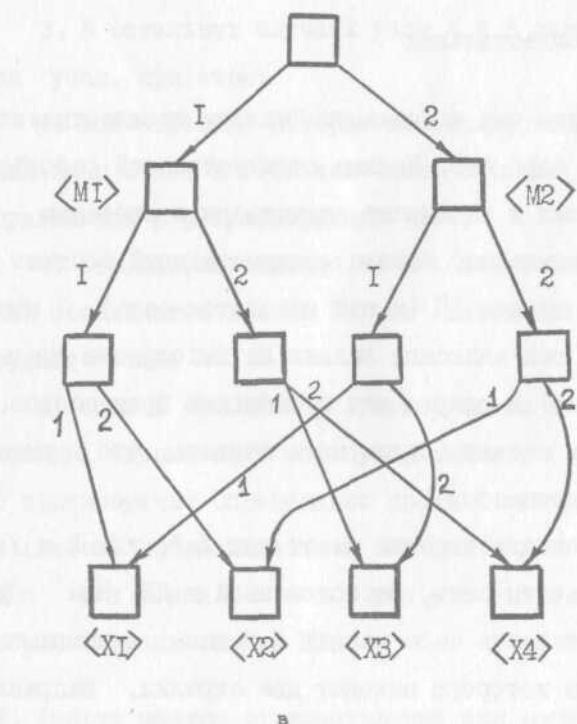
Любая сеть, интерпретируемая как неполноту определенная информация, может также рассматриваться как типичный представитель целого класса сетей, а именно, множества всех сетей, которые можно получить

из данной сети при всех возможных (в рамках данной интерпретации) способах ее доопределения. Например, рис.7, б можно интерпретировать как типовой список длины 2. Аналогично, рис.7, в можно интерпретировать как описание типового отношения транспозиции между матрицами 2-го порядка, поскольку любое конкретное отношение транспозиции между двумя конкретными матрицами может быть получено после доопределения пустых объектов X_1, X_2, X_3, X_4 . Все это напоминает фреймы /21,22/. Фрейм – это типовая структура, содержащая неопределенные объекты, так называемые слоты. Фрейм служит для описания некоторого класса структур (сложных объектов, отношений, ситуаций и т.п.). Подстановка на место слотов конкретных объектов называется конкретизацией фрейма, в результате получается конкретизированный экземпляр данного фрейма – конкретный представитель данного класса. Сети можно считать некоторым обобщением понятия фрейма, при котором конкретизация понимается как доопределение исходной структуры. Доопределение может включать в себя не только замену пустых объектов на атомы или сложные объекты, но и "наращивание" структуры на любых уровнях (рис.7, б), а также склеивание некоторых объектов исходного фрейма. Поскольку в АМС предусматривается работа с неполностью определенной информацией, то в нашем случае допускается неполное доопределение фрейма, при котором получается "более конкретный" фрейм, описывающий более узкий



а

б



в

Рис.7. Неполностью определенные сети:
а,б – списки; в – отношение ТРАНС

класс структур, чем первоначальный. Однако поскольку в АМС полностью определенные структуры отличаются от не-полностью определенных только на уровне интерпретации, то, с формальной точки зрения, любое доопределение фрейма является неполным, что приводит к абсолютному единству представления конкретной информации ("фактов") и типовой информации ("законов"), и работа с этими двумя видами информации производится в АМС по одним и тем же правилам.

3. Машина отождествлений

Рассмотрим так называемую "машину отождествлений", составляющую ядро АМС. Машина отождествлений работает с простыми сетями и выполняет единственную операцию – операцию отождествления. Машина отождествлений состоит из: динамической памяти ДП (в ней находится сеть), памяти заявок ПЗ (в ней записаны заявки на выполнение операций отождествления) и одного или нескольких процессоров, выполняющих эти заявки (будем пока считать, что имеется только один процессор).

Заявка отождествления имеет вид $X=Y$, где X и Y – ссылки на объекты сети, расположенной в ДП. Нам будет удобно представлять себе заявку $X=Y$ как специальный узел сети, из которого выходят две стрелки, направленные на объекты X и Y . Для удобства работы процессора узлы-заявки вынесены в отдельную память (ПЗ).

3.1. Алгоритм отождествления

1. Заявка отождествления выполняется "вхолостую", то есть не изменяет состояния памяти, если она имеет вид $X=X$.

2. Выполнение отождествления приводит к противоречию ("автост" машины), если:

- отождествляются два атома с различными изображениями (например, $T=F$) или
- атом отождествляется со сложным объектом.

3. В остальных случаях узлы X и Y склеиваются в один узел, при этом:

a) все стрелки, которые до отождествления были направлены в X или в Y (в том числе стрелки узлов-заявок), направляются в результирующий узел;

b) стрелки, выходившие из X и/или Y (если такие были), соединяются в один "пучок", выходящий из результирующего узла;

b) если в описании результирующего объекта оказывается два одноименных атрибута, скажем, (A, Z^1) и (A, Z^2) (что противоречит определению простой сети), то один из них (любой) выбрасывается и порождается новая заявка отождествления $Z^1=Z^2$, которая записывается в ПЗ.

3.2. Работа машины отождествлений при решении задач

В ДП помещается исходная сеть, один из объектов которой помечается как "результат" (как правило, это пустой объект). В ПЗ заносится одна или несколько зая-

вок отождествления. Все это вместе называется заданием. Машина запускается. Процессор считывает заявку из ПЗ и выполняет ее в соответствии с алгоритмом отождествления, изложенным выше. Выполненная заявка удаляется из ПЗ (стирается). При выполнении отождествления могут быть порождены новые заявки на отождествление (стр.35, Зв), они записываются в ПЗ. Процессор считывает следующую заявку, выполняет ее и т.д.

Нормальной остановкой машины называется ситуация, когда на очередном шаге работы процессора память заявок оказывается пустой. Состояние помеченного объекта в этот момент выводится в качестве результата выполнения задания.

3.3. Свойства машины отождествлений

Верны следующие два утверждения:

1'. При любом задании работа машины прекращается через конечное число шагов (то есть завершается нормальной остановкой или авостом).

2'. Результат не зависит от порядка выполнения заявок.

Первое утверждение почти очевидно. Действительно, "зацикливание" машины означало бы, что по мере выполнения "старых заявок" все время порождаются "новые", и этот процесс порождения новых заявок никогда не кончается. Но это невозможно потому, что, согласно алгоритму отождествления, порождение заявок всегда сопровождается уменьшением числа узлов сети, а исходная сеть

предполагалась конечной.

Второе утверждение менее очевидно. Сформулируем его более точно. Пусть N_0 - начальное состояние ДП и R_0 - начальное множество заявок, записанное в ПЗ. Рассмотрим следующий недетерминированный режим работы машины. Из множества R_0 случайным образом выбираем заявку и выполняем ее. Получаем новое состояние памяти N_1 . Удаляем из R_0 выполненную заявку и добавляем новые, порожденные заявки (если они были) - получаем новое множество заявок R_1 . Случайным образом выбираем заявку из R_1 , выполняем ее, получаем новое состояние памяти N_2 и новое множество заявок R_2 и т.д. Рассмотрим множество всех возможных реализаций нашего недетерминированного режима или, для краткости, множество всех процессов. Верны следующие два утверждения:

2.1'. Если хотя бы один процесс завершается нормальной остановкой, то и все процессы завершаются нормальной остановкой (следствие: если хотя бы один процесс заканчивается авостом: то и все процессы заканчиваются авостом).

2.2'. В случае нормальной остановки конечные состояния памяти во всех процессах совпадают.

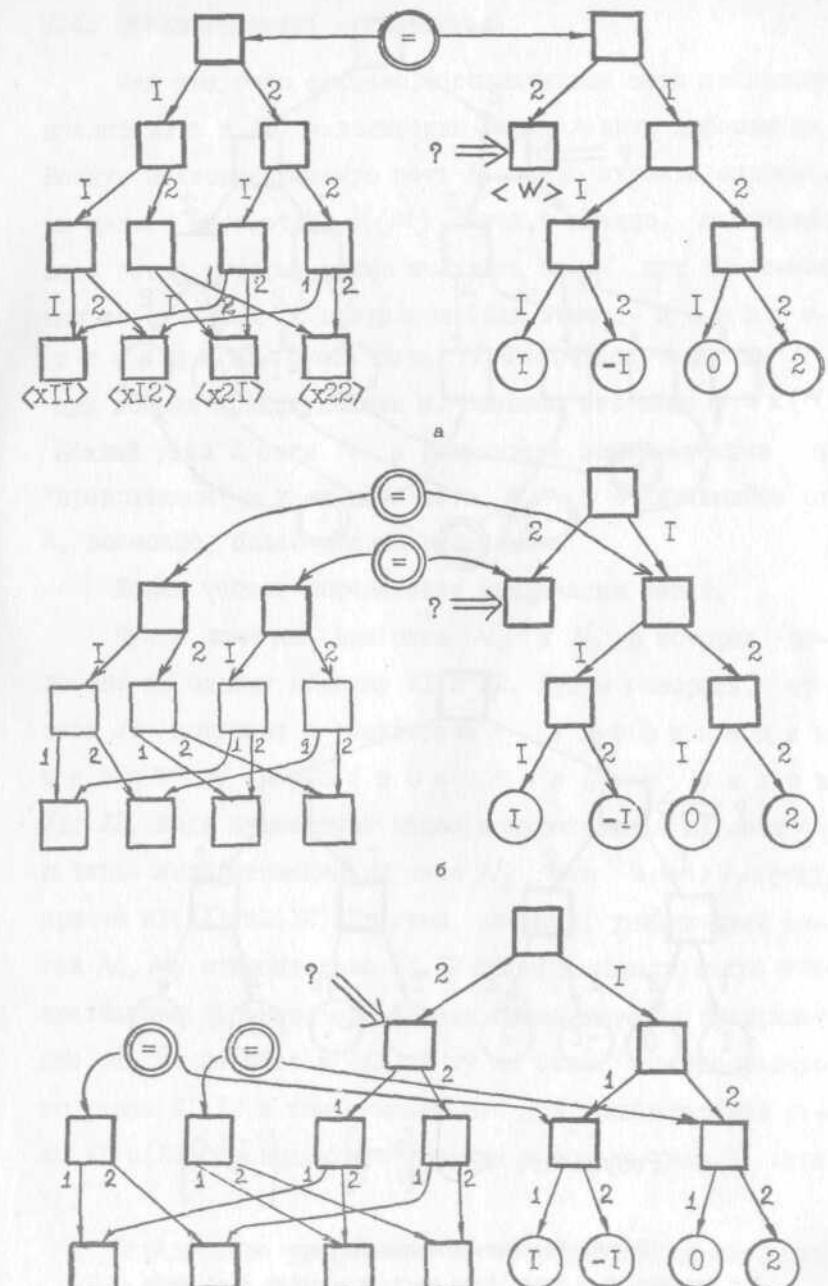
Эти два утверждения имеют строгое математическое доказательство (см. Приложение).

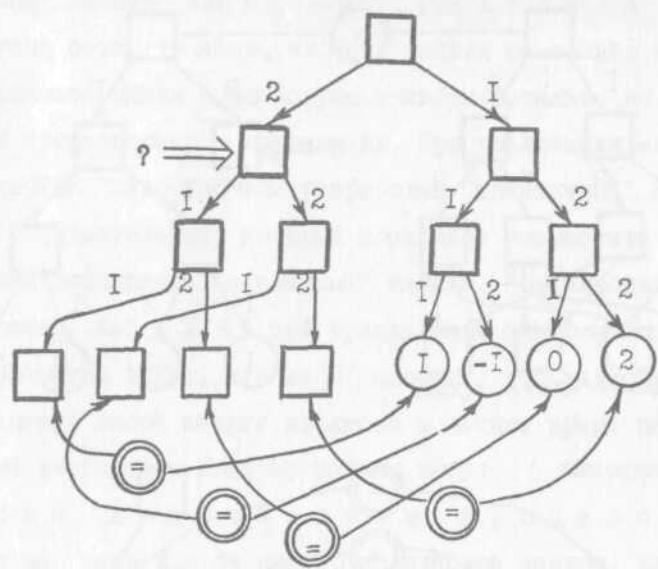
Независимость результата от порядка выполнения заявок - важное свойство АМС, являющееся источником параллелизма, присущего вычислительному процессу, протекающему в АМС. Действительно, если имеются две "неза-

"всесимые" заявки $X=Y$ и $Z=W$, где X, Y, Z, W 4 разных узла сети, то ясно, что эти заявки не только могут выполняться одним процессором в любом порядке, но и двумя процессорами - параллельно. При реализации модели АМС на МВК "Эльбрус" был разработан "практичный" алгоритм отождествления, который позволяет совместить по времени выполнение "всесимых" заявок отождествления (например, $X=Y$ и $X=Z$) при тривиальной синхронизации.

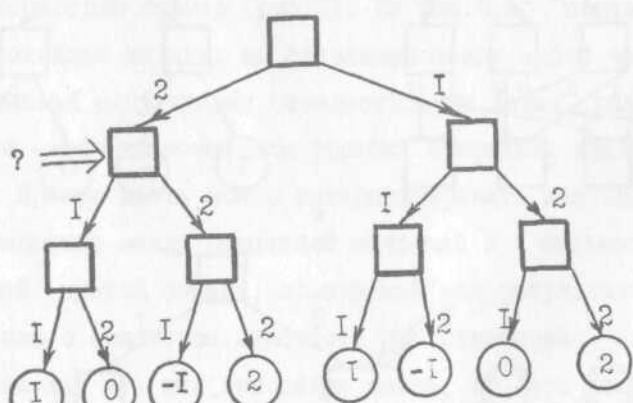
Заметим также, что из 2' следует, что задержка в выполнении любой заявки на любое конечное время не влияет на результат. Поэтому заявки могут выполнятся любым количеством процессоров не зависимо от общего количества заявок, ждущих своего выполнения.

Рассмотрим пример (рис.8). На рис.8, а показана сеть, состоящая из двух не связанных между собой частей и содержащая одну заявку отождествления. Левая половина сети - это знакомый нам "фрейм" отношения транспозиции. Правую часть можно интерпретировать как некоторое отношение между известной матрицей и неизвестной матрицей (пустой объект, помеченный как результат). Утверждение о тождестве означает, что отношение справа устроено так же, как отношение слева, то есть является отношением транспозиции. Итак видим, что процесс отождествления (всего 9 отождествлений) за 4 параллельных шага приводит к доопределению пустого объекта-результата в виде матрицы, транспонированной по отношению к заданной.





г



д

$$\text{Результат: ТРАНС} \begin{pmatrix} 1 & -1 \\ 0 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -1 & 2 \end{pmatrix}$$

Рис.8. Транспонирование матрицы :

а - задание; б - после 1-го шага; в - после 2-го шага ;
г - после 3-го шага; д - после 4-го шага

3.4. Отождествление и унификация

Как уже было сказано, ассоциативные сети позволяют представить в АМС неполностью определенную информацию. Всякую недоопределенную сеть N можно считать описанием целого множества $K(N)$ сетей, а именно, множества всех сетей, которые можно получить из N при всевозможных ее способах доопределения (или иначе - конкретизации). Пусть сеть $N_1 \in K(N)$ получена из N при помощи конкретизации κ . Запишем это так: $N_1 = \kappa(N)$. Каждый узел X сети N в результате конкретизации κ "превратился" в узел $\kappa(X)$ сети N_1 , отличающийся от X , возможно, более полным описанием.

Дадим теперь определение унификации сетей.

Пусть имеются две сети N_1 и N_2 , в которых выделено по одному объекту X_1 и X_2 . Будем говорить, что сеть N получена в результате унификации сетей N_1 , N_2 относительно узлов X_1 , X_2 , если существует такая конкретизация κ_1 сети N_1 и такая конкретизация κ_2 сети N_2 , что $\kappa_1(N_1) = \kappa_2(N_2)$, причем $\kappa_1(X_1) = \kappa_2(X_2)$. Другими словами, унификацией сетей N_1 , N_2 относительно X_1 , X_2 будем называть такую конкретизацию (доопределение) обеих сетей, которая превращает две различные сети в одну и ту же сеть, согласованную по узлам X_1 , X_2 в том смысле, что при доопределении узлы X_1 и X_2 "превратились" в один и тот же узел X сети N .

Определение унификации сетей можно легко обобщить

на случай многих (больше двух) сетей с несколькими парами "согласующих" узлов.

Машину отождествлений можно было бы с полным основанием назвать "машиной унификаций", поскольку верно следующее утверждение:

Если в ДП поместить две сети N_1 и N_2 , а в ПЗ - заявку $X_1=X_2$ и запустить машину, то в случае нормальной остановки конечное состояние памяти окажется сетью, являющейся результатом унификации сетей N_1 и N_2 относительно узлов X_1 , X_2 . Авост может возникнуть только в том случае, когда унификация невозможна (см. Приложение).

Термин "унификация" обычно связывают с нахождением общего частного случая двух (или более) атомарных формул исчисления предикатов I-го порядка /23,24/. Унификация - это основная операция, выполняемая при интерпретации языка ПРОЛОГ. Здесь этот термин употреблен намеренно. Дело в том, что унификация сетей - это обобщение "обычной" унификации. Машина отождествлений "умеет" производить унификацию (а на самом деле можно показать, что минимальную унификацию) атомарных формул исчисления предикатов I-го порядка как унификацию сетей частного вида, которыми можно эти формулы "закодировать". (Например, на рис.8,а левой сетью представлена прологовская формула $\text{ТРАНС}(((X_1.X_2).(X_{21}.X_{22}),((X_1.X_{21}).(X_{12}.X_{22})))$, описывающая общий вид предиката (отношения) ТРАНС, а правой сетью представлена формула-запрос $\text{ТРАНС}((I.-I).(0.2)),?w$).

Минимальная унификация (в обычном смысле) этих формул приводит к подстановке $W=((I.0).(-I.2))$, то есть к решению задачи.

Унификация сетей - это значительно более мощная операция по сравнению с прологовской унификацией. Отметим некоторые наиболее важные ее свойства:

возможность работать с циклическими структурами; отсутствие на формальном уровне фиксированных "арностей" отношений и структур;

возможность иметь недоопределенные отношения и функции (не определенные в очень широком понимании - вплоть до неопределенной арности), в связи с однородностью представления объектов и отношений, что в логическом смысле придает унификации сетей высокий порядок (выше I-го).

4. Схемы. Подобие. Машина отождествлений со схемами

Вернемся к рис.8,а. Ясно, что в содержательном плане левая часть задания служит аналогом программы ("закон транспонирования"), правая же часть - это конкретный запрос на транспонирование - аналог обращения к программе. Однако это всего лишь наша интерпретация сети на рис.8,а - с точки зрения механизмов работы АМС правая и левая части сети равноправны. Достигнутая при этом симметрия создает, тем не менее, определенное неудобство в ситуации, когда решается задача транспонирования многократно с различными исходными матрицами. В этом случае придется многократно вводить в машину зада-

ния, отличающиеся только своими правыми частями, левые же части всех заданий будут полностью совпадать. С практической точки зрения представляется удобным хранить в машине только один экземпляр левой части (схему транспонирования), ограничиваясь в каждом задании только правой частью вместе со ссылкой на схему.

Будем называть схемой изолированный фрагмент сети с одним выделенным объектом — главным объектом схемы, называемым ее входом. Введем также новый тип заявки — заявку подобия. Заявка подобия имеет вид $X:S$ (читается: "объект X подобен схеме S"). Подобие объекта схеме — это фактически есть тождество этого объекта входу схемы. Чисто "техническое" отличие от тождества заключается в том, что при выполнении заявки подобия соответствующая схема копируется и объект отождествляется не с самим входом, а с его копией, что позволяет использовать схему многократно.

Одна из возможных интерпретаций понятий схемы и подобия. Схема описывает структуру некоторого типового объекта (входа схемы), типичного представителя некоторого класса объектов. Подобие объекта схеме — это утверждение о принадлежности данного объекта этому классу. Выполнение заявки подобия — это присвоение объекту типовой структуры, точнее, унификация структуры объекта со структурой типового объекта.

Если с некоторой сетью связана заявка $X:S$, где

- 44 -

X — один из объектов сети, условимся говорить, что в сети имеется обращение к схеме S (по аналогии с обращением к подпрограмме). Естественным расширением возможностей структуризации сетей является введение средств для обращения изнутри схем к другим схемам. Это позволяет, давая определение типового объекта, в частности отношения, описывать его в терминах других типовых объектов, подобно тому, как, давая определение функции, мы представляем ее как суперпозицию других "более элементарных" функций.

Итак, схема задается:

- a) телом схемы (это сеть);
- b) входом схемы (это один из объектов тела);
- c) локальной памятью заявок (ЛПЗ) (если она не пуста, в ней находятся заявки подобия некоторых объектов тела схемы другим схемам; из соображений общности будем допускать также заявки отождествления объектов схемы друг с другом).

Машина отождествлений со схемами, кроме динамической памяти (ДП), имеет постоянную память (ПП) — в ней хранятся схемы. Задание — это исходная сеть в ДП, а также одна или несколько заявок (как правило, подобий) в ПП. Заявки отождествления выполняются как раньше (разд. II, п. 2.3).

Выполнение заявки подобия $X:S$:

- I) тело схемы S переписывается (копируется) в ДП;

- 45 -

- 2) копии заявок из ЛПЗ схемы S записываются в ПЗ;
- 3) копия входа схемы S отождествляется с объектом X .

В отличие от машины отождествлений машина отождествления со схемами может "зацикливаться". Источником "зацикливания" может стать рекурсивное подобие, то есть обращение из некоторой схемы (прямое или косвенно через другие схемы) к самой себе. Однако утверждения I' и $2'$ разд. 2.3.3. остаются верными.

На рис.9 показано задание для решения задачи транспонирования матрицы 2-го порядка на машине со схемами.

Введение схем ничего не добавляет к алгоритмическим возможностям машины отождествлений, это всего лишь средство структуризации простых сетей. В следующем разделе мы расширим определение сети.

5. Условная конструкция. Расширенная сеть.

Рекурсивные схемы. Базовая машина.

Заявки отождествления и подобия, записанные в ПЗ, всегда готовы к выполнению. Как только имеется свободный процессор, любая из них может быть выполнена — будем называть эти заявки **безусловными**. После копирования некоторой схемы копии заявок, находящиеся в ее ЛПЗ, помещаются в ПЗ и сразу же могут выполняться, поэтому "локальные" заявки схем, хранящиеся в

ЛПЗ, мы тоже будем называть безусловными. В настоящем разделе рассмотрим так называемые **условные**

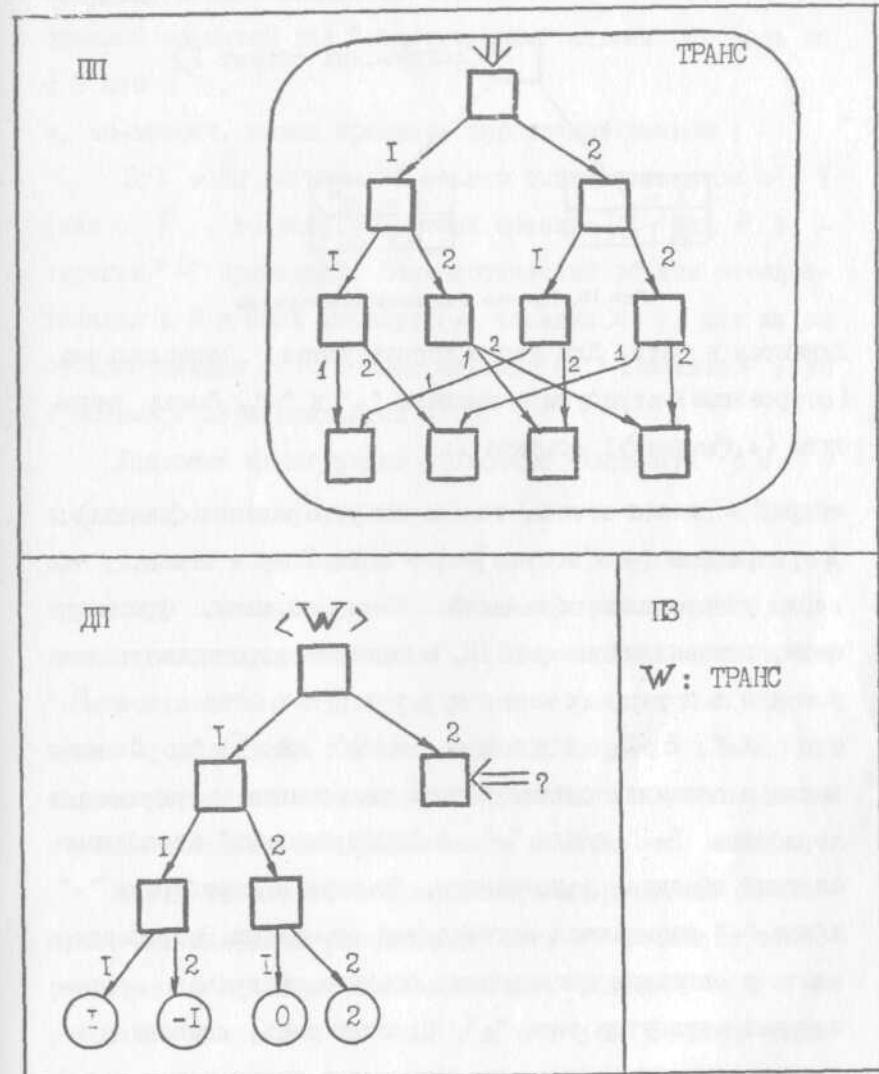


Рис.9. Пример задания на машине со схемами заявки. Они "спрятаны" в тела схем и в ДП и переходят в безусловное состояние только после выполнения

некоторых условий (при этом они переписываются в ПЗ). На рис.10 показано, как списки условных заявок подсое-

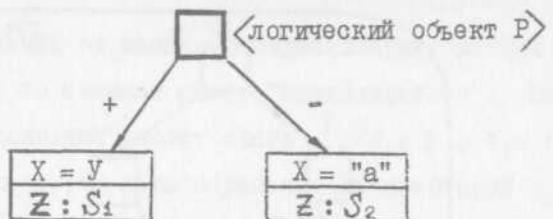


Рис.10. Пример условной конструкции

динутся к сети. Для этого используются специальные (встроенные) атрибуты с именами "+" и "-". Смысл атрибута $(+, \langle \text{заявки} \rangle)$ объекта X:

если X – истина, то верны утверждения $\langle \text{заявки} \rangle$; для атрибута $(-, \langle \text{заявки} \rangle)$ – если X – ложь, то верны утверждения $\langle \text{заявки} \rangle$. Таким образом, фрагмент сети, показанный на рис.10, можно интерпретировать как условную конструкцию "если P то $X=Y, Z:S_1$, иначе $X='a', Z:S_2$ ". Совмещение в описании одного и того же объекта встроенных атрибутов "+" и/или "-" с "программными" атрибутами сложных объектов запрещается. Объекты с атрибутами "+" и/или "-" называются логическими объектами. Разрешается иметь в описании логического объекта несколько одинаковых атрибутов типа " \pm ". Простую сеть, дополненную логическими объектами, назовем расширенной сетью. В расширенной сети существует четыре типа объектов: атомы, сложные, логические и пустые объекты. Правила отождествления (см.стр.35) дополняются,

во-первых, новым случаем аваста по несовпадению типов:

2в) если отождествляется логический объект со сложным объектом или "нелогическим" атомом (то есть не с T или F),

и, во-вторых, новым правилом порождения заявок;

3г) если логический объект отождествляется с T (или с F), то результирующий объект – T (или F) – стрелки " \pm " пропадают. Если логический объект отождествлялся с T и имел атрибут $(+, \langle \text{заявки} \rangle)$, или же он отождествлялся с F и имел атрибут $(-, \langle \text{заявки} \rangle)$, то $\langle \text{заявки} \rangle$ переписываются в ПЗ.

Условные конструкции позволяют создавать рекурсивные схемы, то есть схемы, содержащие условные обращения к самим себе (безусловное рекурсивное обращение – это ошибка программиста, приводящая к зацикливанию машины). Возможность описывать "многовариантные" структуры, а также рекурсивные структуры произвольной глубины (справки, деревья и т.п.), которая появляется с введением логических объектов, качественно повышает алгоритмические возможности машины.

Можно показать, что машина отождествлений со схемами, использующая расширенные сети и соответственно расширенные правила отождествления, обладает алгоритмической полнотой, то есть является одним из возможных механизмов реализации вычислимости (подобно машине Тьюринга или нормальным алгоритмам Маркова). При этом достаточно иметь только два атома – T и F (или I и 0). Будем называть такую ма-

шину базовой машиной или базовой АМС.

6. Встроенные механизмы

Базовая машина - это весьма непрактичный вариант АМС. Эта машина хорошо приспособлена для работы с произвольной структурной информацией, однако вся работы с числовой или символьной информацией в ней крайне неэффективна, так как числа и символы представляются при помощи двоичного кодирования, являются структурами, и все операции над ними должны быть реализованы программно (например, чтобы уметь складывать числа, нужно составить схему сложения двоичных кодов). Поэтому представляется практически целесообразным расширить множество атомов, включив в него вещественные и целые числа, символы, последовательности символов (слова). При этом сразу же возникает необходимость во введении в машину встроенных механизмов для работы с атомами этого типа (арифметические операции, сравнение символов и т.п.).

Существует два вида встроенных механизмов - встроенные атрибуты и встроенные отношения.

Встроенные атрибуты служат для "аппаратного" вычисления значений функций от одного аргумента. Встроенный атрибут отличается от обычного (программного) тем, что он имеет зарезервированное, известное машине имя.

Наличие у объекта X атрибута вида $\langle f, Y \rangle$, где f - зарезервированное имя некоторой встроенной функции, означает, что X и Y связаны между собой зависимостью $Y = f(X)$. Если X отождествляется с некоторым атомом a, то после отождествления:

вычисляется значение $f(a)$;

порождается заявка $Y = f(a)$;

сам атрибут удаляется (у атомов не может быть атрибутов).

Примеры встроенных функций, задаваемых при помощи встроенных атрибутов: знак числа, булева функция НЕ и т.д.

Правила "склеивания" одноименных атрибутов при отождествлении (см.стр.35, Зв) сохраняются и для встроенных атрибутов. Так, если после отождествления в описании объекта X окажется два атрибута вида $\langle f, Y \rangle$ (f, Z), то один из них удаляется и порождается заявка $Y = Z$ (то есть еще до вычисления значения (X) уже известно, что $Y = f(X)$ и $Z = f(X)$, а значит, $Y = Z$).

Встроенные отношения задаются так же, как и "программные" отношения, то есть сам факт существования отношения изображается в сети сложным объектом X, атрибуты которого соответствуют аргументам этого отношения. Какое именно отношение имеется в виду определяется безусловной или условной заявкой подобия X:R, где R - имя отношения. В случае "программного" отношения R - это имя схемы, записанной в III. В случае же встроенного отношения R - это зарезервированное имя встроенного от-

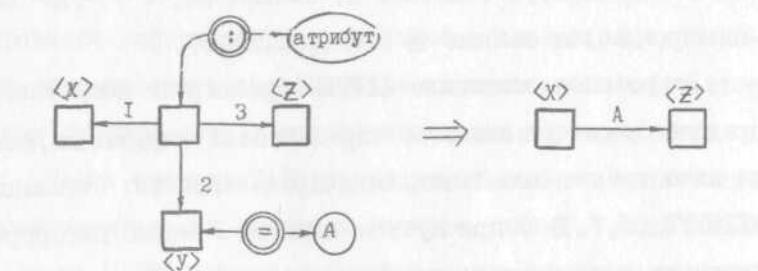
нашения. Не будем описывать подробно механизмы работы встроенных отношений, скажем только, что при выполнении заявки подобия $X:P$ происходит "включение" данного конкретного отношения X – начиная с этого момента, при отождествлении какого-либо аргумента отношения X с атомом, отношение "срабатывает" и делается попытка на основании уже "вычисленных" значений аргументов определить значения остальных аргументов. Если эта попытка удаётся, то порождаются соответствующие заявки отождествления.

Примеры встроенных отношений.

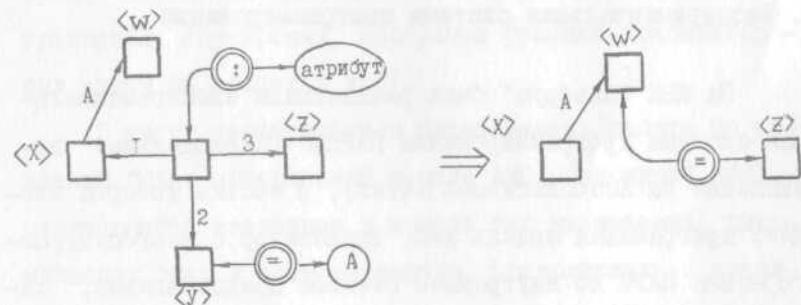
СУММА (X, Y, Z). Формула отношения: $Z = X+Y$. Если после включения отношения определились значения X и Y , то есть объекты X и Y отождествились с атомами, например, $X=I$ и $Y=2$, то вычисляется сумма $I+2=3$ и порождается заявка $Z=3$. Подчёркнем, что отношение СУММА – это именно отношение, а не операция сложения, поэтому оно способно работать в разных направлениях. Например, если произойдут отождествления $Z=3$, $Y=2$, то отношение "сработает", произведя вычитание $3-2=1$, и будет порождена заявка $X=I$.

EQ(X, Y, Z). Формула отношения: $Z = (X=Y)$. Объект Z принимает значение T , если $X=Y$, и F в противном случае. Работа отношения: а) если определились значения X и Y , то происходит их сравнение, после чего в зависимости от результата сравнения Z отождествляется с T или с F ; б) если выяснилось, что $Z=T$, то порождается заявка $X=Y$.

Сеть, составленная из встроенных атрибутов и отношений при своей работе, реализует в АМС режим, близкий к data-flow – режиму с возможностью распространения информации в различных направлениях, так как многие встроенные отношения имеют многофункциональный характер.



а



б

Рис. 11. Встроенное отношение АТРИБУТ:
а – первый случай; б – второй случай

Особое место в системе имеет встроенное отношение АТРИБУТ (X, Y, Z). Смысл отношения АТРИБУТ(X, Y, Z): объ-

ект X - сложный объект, имеющий атрибут вида (Y, Z). Отношение срабатывает после отождествления Y с атомом, например Y=A. При этом(рис. II):

если в описании объекта X не было атрибута с именем A (рис.II,a), то в него включается новый атрибут (A, Z);

если атрибут с именем A, скажем (A,W), уже был, то порождается заявка Z = W (рис.II,b).

Встроенное отношение АТРИБУТ расширяет возможности представления неполностью определенной информации, вводя качественно новый вид недодопредельности: отношение АТРИБУТ (X,Y,Z) при пустом объекте Y можно интерпретировать как стрелку, проведенную из X в Z с неопределенной пометкой на ней.

7. Экспериментальная система программирования

На МВК "Эльбрус" была реализована экспериментальная система программирования ПАРНАС (ПАРаллельные вычисления На Ассоциативных Сетях), в состав которой входят: программная модель АМС, транслятор с непроцедурного языка ПАРС во внутреннее сетевое представление, система поддержки архива схем и ряд вспомогательных программ. Язык ПАРС можно считать "автокодом" АМС, в то же время это язык высокого уровня, непроцедурный. В настоящее время разработан более удобный и богатый по своим выразительным возможностям язык ПАРС-86 и транслятор с этого языка в сетевое представление.

В процессе разработки программной модели АМС был

конкретизирован ряд механизмов, в частности, были выбраны конкретный вид представления сети в памяти, конкретный механизм отождествления, механизм сборки мусора и т.п. Было введено понятие виртуального копирования схем и соответствующие механизмы копирования, позволяющие, во-первых, во многих случаях не производить настоящего (реального) копирования информации, заключенной в схемах, и, во-вторых, организовать параллельное копирование элементов схемы одновременно с наложением копии схемы на сеть.

Через систему ПАРНАС пропущено большое количество экспериментальных программ (точнее, схем), написанных на языке ПАРС("лиспоподобные" программы обработки списков, программы транспонирования, обращения и перемножения матриц, решения систем алгебраических линейных уравнений, сортировки, программы решения комбинаторных задач на графах и др.).

В чисто теоретическом плане велась работа по созданию более совершенной версии АМС ("сложной" АМС), отличающейся введением в модель так называемых динамических схем и схем-вариантов, позволяющих значительно повысить "интеллектуальный уровень" системы.

8. Свойства АМС

Ограничимся кратким описанием тех основных свойств АМС, которые были установлены теоретически и затем подтверждены в многочисленных экспериментах.

8.1. Параллельность вычислений

Оценивая результаты моделирования в системе ПАРНАС параллельного вычислительного процесса решения задач, следует учитывать, что программная модель АМС – это не модель конкретной аппаратуры, а модель абстрактной машины. Значит, и выявляемая при этом степень достигаемой параллельности вычислений носит "абстрактный" характер в том смысле, что она может служить лишь оценкой той максимальной параллельности, которую может дать моделируемый способ вычислений (без учета тех поправок, которые может внести ограниченность ресурсов реальной аппаратуры). Достигаемая степень такой "абстрактной" параллельности оказалась очень высокой, особенно на переборных задачах, например, при поиске всех гамильтоновых циклов в графе из 15 вершин количество одновременно выполняемых (и вполне осмысленных) операций доходило до 5 тыс. Учитывая определенную неточность количественных оценок параллельности, сделаем упор на качественной стороне параллельного режима вычислений, наблюдаемого в АМС.

Параллелизм, присущий АМС, если рассматривать его на самом детальном уровне, сводится к параллельному выполнению отождествлений и подобий. Однако возможен такой уровень рассмотрения, при котором можно обнаружить в АМС различные виды параллелизма, по отдельности характерные для различных существующих моделей и проявляющиеся в АМС одновременно:

"data-flow" – параллелизм. Проявляется при работе встроенных атрибутов и отношений;

"структурный" параллелизм (параллельный доступ к элементам структур). При отождествлении двух однотипных структур их элементы попарно отождествляются параллельно. Поскольку в АМС структурные объекты и отношения формально неразличимы, частным случаем такого структурного параллелизма является параллельная подстановка аргументов в отношения;

"функциональный" параллелизм. Если преобразование информации, выполняемое на некотором фрагменте сети, означает по смыслу вычисление значения функции, например, $F(G(X), H(Y))$, то схемы, при помощи которых реализованы функции G и H , вызываются и накладываются на сеть независимо друг от друга и одновременно. Таким образом, происходит параллельное вычисление аргументов функций (и отношений);

"конвейерный" параллелизм. Проиллюстрируем этот вид параллелизма на примере. Пусть производится вычисление $Y = G(H(L(X)))$, где X – структура, например, список. Тогда сеть, соответствующая этому вычислению, будет иметь вид, показанный на рис. I2. Предполагается, что функции G , H и L реализованы в виде рекурсивных схем как бинарные отношения между соответствующими аргументами и результатами. Все три схемы копируются одновременно и каждая "накладывается" на свой участок сети. Каждая схема при своем копировании вносит в память условную конструкцию для рекурсивного обращения к

себе самой, имеющую следующий смысл: "если входной список не пуст, то перейти к его хвосту". В начальный момент объекты X_1 и X_2 не определены, поэтому условные конструкции схем G и H не сработают, что же касается

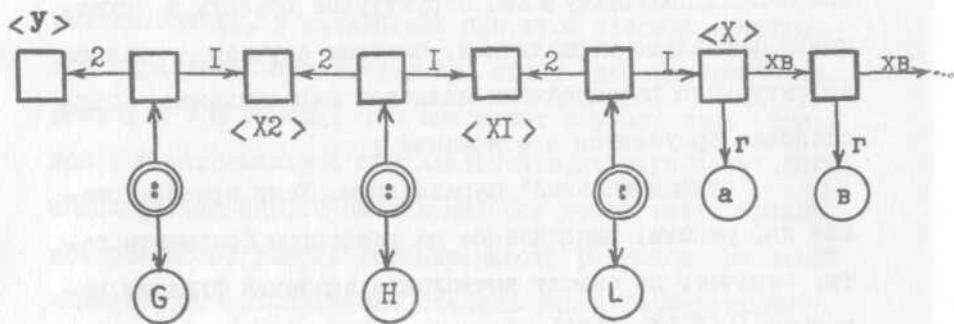


Рис.12. Конвейер

схемы L , то она сразу же вызовет себя рекурсивно и перейдет к хвосту списка X и затем, двигаясь от одного шага рекурсии к другому, будет "просматривать" список X , одновременно формируя (постепенно доопределяя) на выходе список X_1 . Для того, чтобы сработала условная конструкция схемы H , совсем не обязательно, чтобы список X_1 был полностью определен, достаточно того, чтобы он был не пуст, поэтому схема H будет продвигаться по списку X_1 по мере его формирования одновременно с работой схемы L . При этом схема H будет постепенно доопределять список X_2 , элементы которого будут сразу же "подхватываться" схемой G . Таким образом, все три схемы будут работать од-

новременно. Образуется своеобразный "конвейер". Подобный же "эффект конвейера" наблюдается в системе распараллеливания ЛИСПа, использующей "опережающие вычисления" /21/.

8.2. Обратимость вычислений

Ввиду того, что основным понятием в АМС является понятие отношения (в частности, функции рассматриваются как частный случай отношений), в АМС естественным образом возникает эффект обратимости вычислений. Другими словами, вычислительный процесс носит во многих случаях ненаправленный характер. Направление часто зависит от того, какие именно аргументы того или иного отношения заданы в данном конкретном случае применения этого отношения. Такой же эффект наблюдается в языке ПРОЛОГ, где также все построено на понятии отношения. В АМС обратимость вычислений связана с обратимостью (или, точнее, многофункциональностью) встроенных отношений и с ненаправленным характером унификации сетей (полная симметрия, равноправность сетей, участвующих в унификации). Например, в задании (см. рис. 9) можно было бы поменять местами стрелки у объекта W , и отношение ТРАНС в этом случае сработало бы в противоположном направлении. Иногда бывает вообще невозможно говорить о каком-либо направлении вычислений (то есть где исходные данные и где результат). Например, если две матрицы неполностью определены, то использование того факта, что они находятся между собой в отношении

TRANSC, позволяет их доопределить (рис.13).

Разумеется, обратимость возникает далеко не везде просто потому, что это не всегда возможно в принципе (теоретически). В АМС есть один важный фактор, который уменьшает эффект обратимости в некоторых случаях,

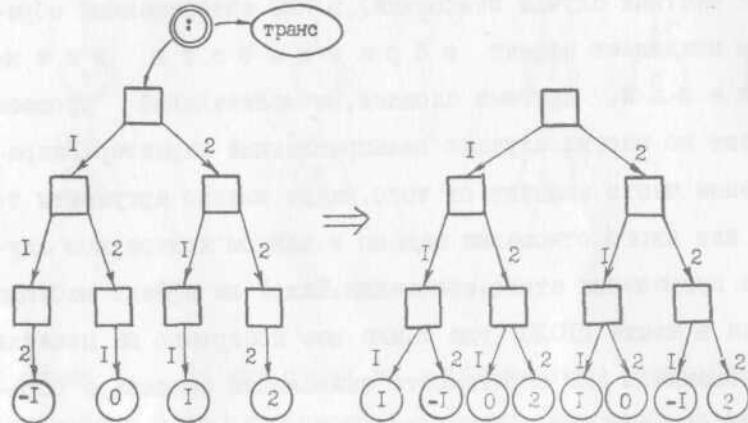
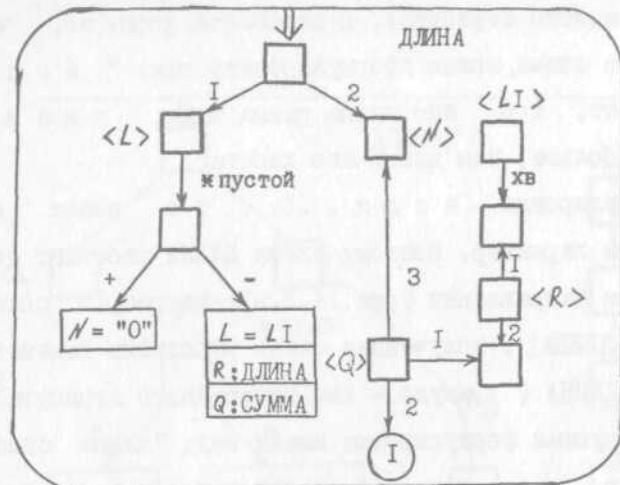
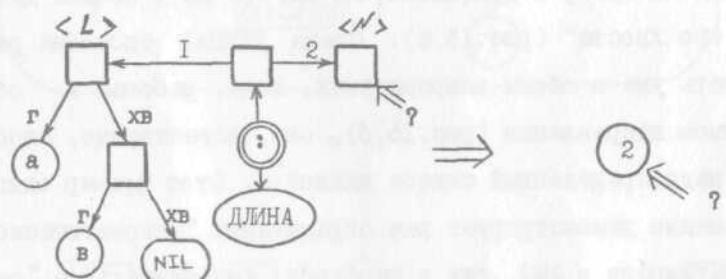


Рис.13. Взаимное доопределение матриц

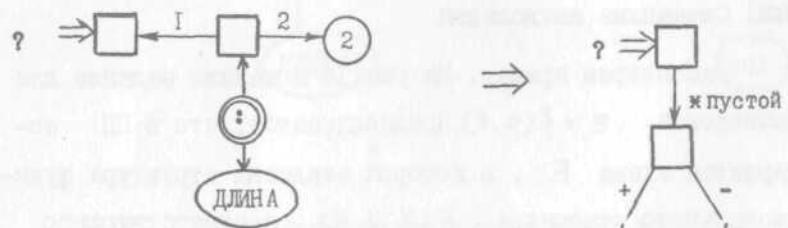
когда он теоретически и мог бы возникнуть (и действительно существовал бы в ПРОЛОГе) – это условная конструкция, которая носит направленный характер. В этих случаях при необходимости можно доопределить обратимости программно, ценой, как правило, небольших усилий. В качестве примера (рис.14) приведем рекурсивную схему ДЛИНА (рис.14, а), при помощи которой реализуется функция "длина списка" (* пустой – это



а



б



в

Рис.14. Однонаправленное отношение ДЛИНА :

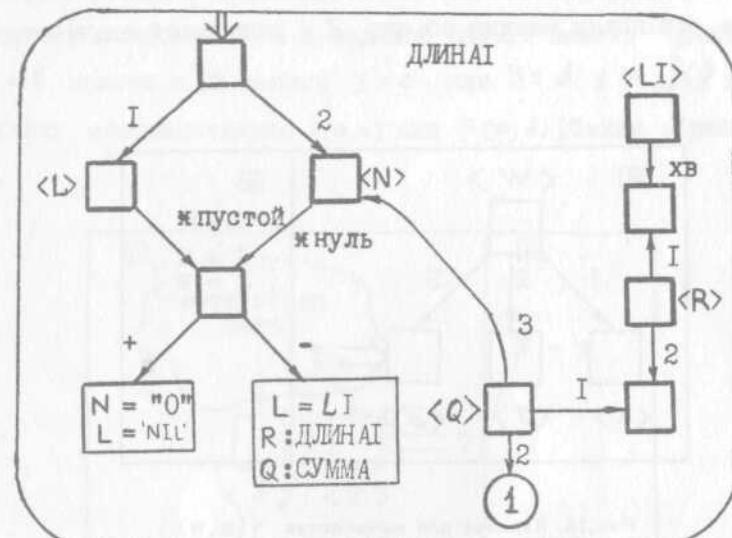
а – схема ДЛИНА ; б – работа в прямом направлении; в – работа в обратном направлении

имя встроенного атрибута). В словесной форме то, что записано в схеме, можно сформулировать так: "если список пуст, то его длина равна нулю, иначе она на 1 больше, чем длина его хвоста".

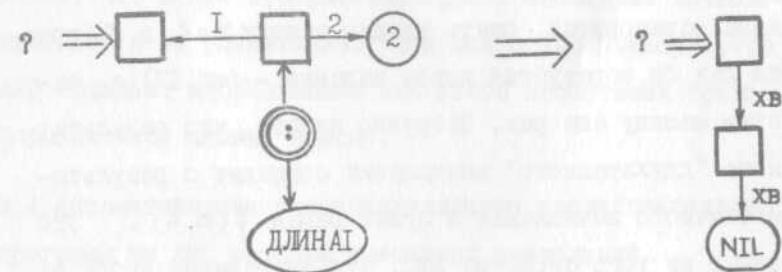
Формулировка если....то имеет направленный характер. Поэтому схема ДЛИНА работает только в одном направлении (рис. I4, б, в). На рис. I5 показана схема ДЛИНА1, полученная очень небольшим изменением схемы ДЛИНА («нуль - имя встроенного атрибута»). Соответствующая формулировка имеет вид: "длина списка равна нулю тогда и только тогда, когда он пуст, в противном случае она на 1 больше длины его хвоста" (рис. I5, а). Схема ДЛИНА1 способна работать уже в обоих направлениях, хотя, работая в обратном направлении (рис. I5, б), она, естественно, строит недоопределенный список длиной 2. Этот пример одновременно демонстрирует как ограничения "автоматической" обратимости в АМС, так и наоборот, дополнительные "резервы" обратимости, которые имеются в АМС благодаря возможности использовать недоопределенную информацию.

8.3. Смешанные вычисления

Рассмотрим пример. На рис. I6 показано задание для вычисления $Z = f(a, b)$. Предполагается, что в ПП содержится схема F , в которой записана структура функционального отношения $F(X, Y, Z)$, соответствующего формуле $Z = f(X, Y)$. После запуска машины схема F копируется и накладывается своим входом на объект W .



а



б

Рис.15. Двунаправленная схема ДЛИНА 1:
а - схема ДЛИНА 1; б - работа схемы в обратном направлении

Одновременно с этим X и Y отождествляются с a и b и тем самым значения аргументов подаются на обработку. После остановки машины объект Z принимает значение $f(a, b)$.

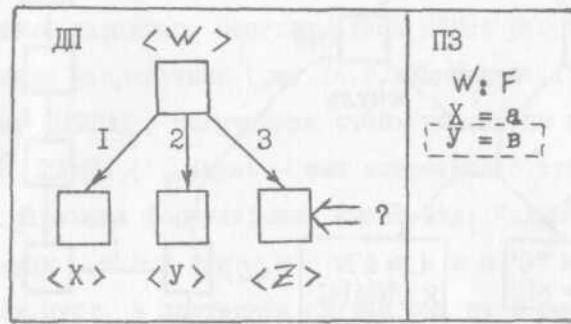


Рис.16. Задание для вычисления $f(a, b)$

Проделаем следующий эксперимент. Временно удалим заявку $Y = b$ из ПЗ и запустим машину на выполнение задания рис.16 без этой заявки. Затем, после того как машина остановится, опять введем заявку $Y = b$ в ПЗ (при этом как бы образуется новое задание – рис.17) и запустим машину еще раз. Нетрудно видеть, что результат такого "двухэтапного" вычисления совпадет с результатом обычного вычисления и будет равен $f(a, b)$. Это следует из того свойства АМС, что выполнение любой заявки можно отложить на любое конечное время и результат от этого не изменится. Что же представляет собой сеть, связывающая объекты Y и Z на рис.17? Очевидно, что эта сеть реализует функцию $Z = g(Y)$, где $g(Y) = f(a, Y)$. Действительно, вид этой сети не зависит от

вида заявки $Y = b$, так как машина "не знала", какая именно заявка удалена, когда эта сеть формировалась. Поэтому можно было бы в задании рис.17 вместо заявки $Y = b$ ввести в ПЗ заявку $Y = c$ или $Y = d$ и т.д. и получить соответственно $f(a, c)$ или $f(a, d)$. Таким образом,

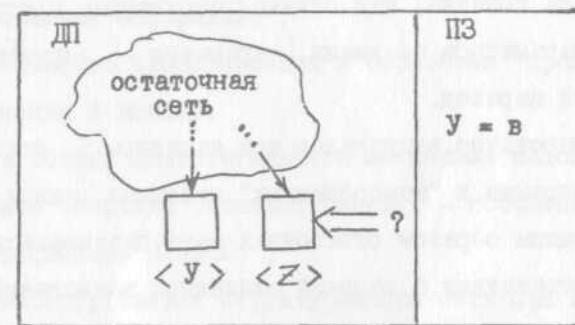


Рис.17. Задание после смешанных вычислений

видим, что в АМС проявляется эффект смешанных вычислений. Это и не удивительно, так как можно показать, что АМС обладает необходимыми для этого свойствами трансформационной машины Ершова.

8.4. Автоматическое распараллеливание последовательных программ на АМС методом смешанных вычислений

Эксперименты, проведенные на системе ПАРНАС, продемонстрировали широкие возможности для оптимизации и синтеза схем при помощи смешанных вычислений. Наиболее значительным экспериментом такого рода явилась работа А.Н.Черных по автоматическому распараллеливанию прог-

рамм, написанных на языке АЛГОЛ 60 /9,25/. Идея работы такова. На языке ПАРС был написан интерпретатор АЛГОЛа 60. Это оказалось непростой задачей, так как для этого потребовалось сформулировать непроцедурную (точнее, реляционную - основанную на понятии отношения) семантику языка АЛГОЛ 60, включая такие сложные для данного подхода понятия, как блочная структура программы, передача параметров по имени, параметры - процедуры, безусловный переход.

Интерпретатор запускался при заданном исходном тексте программы и "замороженных" исходных данных. Получаемая таким образом остаточная сеть оказывалась в полном соответствии с теорией смешанных вычислений результатом трансляции исходной программы с АЛГОЛа 60 на внутренний сетевой язык АМС. В результате многочисленных экспериментов (50 алгольских программ) была продемонстрирована возможность выявления естественного параллелизма АЛГОЛ-программ при помощи этого метода. Метод позволяет транслировать программы на внутренний язык АМС с любого последовательного языка. Замечательным свойством этого метода также является то, что процесс трансляции производится на АМС и обладает, как оказалось, значительной параллельностью.

9. Выводы

АМС - это вычислительный механизм, характеризующийся следующими свойствами:

"семантический" уровень представления информации (ассоциативная сеть) ;

возможность представления и обработки неполностью определенной информации ;

единство представления и обработки "программной" информации и данных;

в основу вычислительного механизма положена единственная операция (отождествление) - обобщение операции унификации ПРОЛОГа ;

многуровневая структуризация сети при помощи схем; наличие условной конструкции, обеспечивающей возможность описания и построения рекурсивных структур;

непроцедурный внешний язык, не содержащий средств явного задания параллелизма;

наличие параллелизма на микроуровне (на уровне элементарных операций);

проявление на "макроуровне" различных видов параллелизма, характерных для других моделей параллельных вычислений;

"обратимость" вычислений;

наличие свойств трансформационной машины Ершова и, как следствие, смешанные вычисления; благодаря которым открываются широкие возможности для:

синтеза схем;

оптимизации схем;

трансляции традиционных последовательных программ на внутренний сетевой язык АМС с одновременным автоматическим распараллеливанием этих программ;

концептуальная простота машины, основанной на ограниченном числе базовых понятий.

Изложение представляет собой последовательность пунктов, каждый из которых – определение либо утверждение. Большинство утверждений очевидно. Наименее очевидные будут доказаны.

I°. Пусть \mathcal{A} – конечное или счетное множество атомов – элементов, природа которых для нас безразлична. Пусть далее Ω – конечное или счетное множество "имен".

Определим простую сеть N над (\mathcal{A}, Ω) как пару

$$N = (X_N, Q_N),$$

где X_N – конечное множество динамических объектов сети;

Q_N – множество связей (или стрелок) вида xBy , где $x \in X_N$, $B \in \Omega$, $y \in (X_N \cup \mathcal{A})$. Для любых x из X_N и B из Ω в Q_N содержится не более одной связи вида xBy .

Условимся также использовать обозначение Z_N для множества $(X_N \cup \mathcal{A})$ всех объектов, которым "разрешено" находиться на концах стрелок сети.

Множество всех возможных сетей над (\mathcal{A}, Ω) обозначим $\mathcal{N}(\mathcal{A}, \Omega)$.

В дальнейшем изложении будем говорить просто "сеть", опуская слово "простая".

2. Сеть N_2 называется частным случаем или конкретизацией сети N_1

(краткая запись: $N_1 \geq N_2$), если существуют два отображения $\varphi: Z_{N_1} \rightarrow Z_{N_2}$ и $\psi: Q_{N_1} \rightarrow Q_{N_2}$ таких, что

a) $\varphi(a) = a$ для всех $a \in A$;

b) $\psi(x \vee y) = \varphi(x) \vee \varphi(y)$ для всех $x \vee y \in Q_{N_1}$.

Из соображений удобства изложения объединим эти два отображения в одно:

$$\varPhi_{N_1 \rightarrow N_2}: (Z_{N_1} \cup Q_{N_1}) \rightarrow (Z_{N_2} \cup Q_{N_2}).$$

Примечание. Определение 2° формализует наше представление о конкретизации (доопределении) сети. Если $N_1 \geq N_2$, то N_2 представляет собой образ сети N_1 при отображении $\varPhi_{N_1 \rightarrow N_2}$, возможно дополненный новыми объектами и связями. Кроме того, N_2 может отличаться от N_1 тем, что некоторые объекты и/или стрелки в образе этой сети в N_2 "склеены" (так как два различных объекта из N_1 могут переводиться отображением \varPhi в один и тот же объект).

3°. Пусть $N_1 \geq N_2$. Сеть N_2 называется склейкой сети N_1 , если отображение $\varPhi_{N_1 \rightarrow N_2}$ сюръективно. В этом случае будем писать $N_1 \gg N_2$.

4°. Будем говорить, что сеть N_1 изоморфна сети N_2 и писать $N_1 = N_2$, если отображение $\varPhi_{N_1 \rightarrow N_2}$ биективно. Изоморфные сети мы не будем различать между собой.

5°. Сети N_1, N_2 называются эквивалентными (краткая запись $N_1 \sim N_2$), если $N_1 \geq N_2$ и $N_2 \geq N_1$ (эквивалентные сети, вообще говоря, не изоморфны).

6°. Пусть $N \in \mathcal{N}(A, \Omega)$. Обозначим через $K(N)$ множество всех конкретизаций сети N :

$$K(N) = \{N_i \mid N \geq N_i\}.$$

Примечание. Возможна такая интерпретация множеств $K(N)$: $K(N)$ – это некоторый класс ситуаций, описываемый фреймом N .

7°. Сеть N является максимальным элементом (в смысле отношения \geq) множества $K(N)$ своих конкретизаций. Сеть N – это вообще говоря не единственный максимальный элемент $K(N)$, так как любая сеть, эквивалентная N , также является максимальным элементом.

$$8°. N_1 \geq N_2 \Leftrightarrow K(N_1) \supseteq K(N_2)$$

$$N_1 \sim N_2 \Leftrightarrow K(N_1) = K(N_2)$$

(" \Leftrightarrow " означает "тогда и только тогда, когда...").

9°. Пусть $L \subseteq \mathcal{N}(A, \Omega)$ – некоторое множество сетей. Множество L называется N -выразимым, если существует такая сеть N , что $L = K(N)$. Множество всех N -выразимых подмножеств $\mathcal{N}(A, \Omega)$ обозначим через $\mathcal{H}(A, \Omega)$. Очевидно, что

$$\mathcal{H}(A, \Omega) = \{K(N) \mid N \in \mathcal{N}(A, \Omega)\}.$$

10°. Множество всех склеек сети N обозначим через $K^*(N)$:

$$K^*(N) = \{N_i \mid N \geq N_i\}.$$

Множество $K^*(N)$ – конечное, частично упорядоченное (отношением \geq) множество.

II°. Сеть N является единственным максимумом (относительно \geq) множества $K^*(N)$.

I2°. Сеть N называется объединением сетей N_1 и N_2 (кратко: $N = N_1 + N_2$), если она имеет вид

$$N = \{X'_N \cup X''_N, Q'_N \cup Q''_N\},$$

причем а) $X'_N \cap X''_N = \emptyset$, $Q'_N \cap Q''_N = \emptyset$;

б) сеть N_1 изоморфна сети $\{X'_N, Q'_N\}$,

сеть N_2 изоморфна сети $\{X''_N, Q''_N\}$.

Примечание. Для того чтобы получить объединение сетей N_1 и N_2 , нужно "нарисовать" их рядом, и получившаяся несвязная сеть как раз и будет их объединением.

$$I3°. K(N_1 + N_2) = K(N_1) \cap K(N_2).$$

I4°. Множество $\mathcal{H}(A, \Omega)$ является полурешеткой относительно операции \cap .

I5°. Пусть N - сеть, а R - множество тождеств вида $x = y$ ($x, y \in Z_N$) (R - множество тождеств над N). Будем называть сеть N_1 фактор-сетью сети N относительно R , если

а) $N \geq N_1$,

б) существует отображение $\varphi_{N \rightarrow N_1}$, удовлетворяющее 2° а, б, такое, что

$$[(x = y) \in R] \Rightarrow [\varphi_{N \rightarrow N_1}(x) = \varphi_{N \rightarrow N_1}(y)]$$

(" \Rightarrow " означает "влечет за собой").

Множество всех фактор-сетей сети N относительно R обозначим $F(N, R)$.

$$I6°. F(N, \emptyset) = K(N).$$

I7°. Положим $F^*(N, R) = \{N_1 \mid N_1 \in F(N, R), N \geq N_1\}$. $F^*(N, R)$ - конечное, частично упорядоченное (отношением \geq) множество. Очевидно, $F^*(N, \emptyset) = K^*(N)$.

I8°. Теперь рассмотрим работу машины тождествений при начальном состоянии (N_0, R_0) (N_0 - начальная сеть в ДП, R_0 - начальное множество заявок вида $x = y$, $x, y \in Z_{N_0}$). Будем считать, что фиксирован некоторый произвольный процесс P (см. разд. II, п. 3.3), завершившийся на i -ном шаге нормальной остановкой или автостом. Процесс порождает последовательность состояний машины

$$(N_0, R_0), (N_1, R_1), \dots, (N_n, R_n).$$

Таким образом, на каждом i -ом шаге работы машины мы имеем некоторую сеть N_i и систему R_i тождеств над ней.

$$I9°. F(N_i, R_i) = F(N_{i+1}, R_{i+1}) \quad (i=0, 1, \dots, n-1).$$

Доказательство. Рассмотрим заявку $\tau_i \in R_i$, выполненную на i -ом шаге. Если эта заявка была "холостой", то есть вида $x = \infty$, то утверждение I9° очевидно. Предположим поэтому, что тождество τ_i имело вид $p = q$ и при переходе от N_i к N_{i+1} два (различных) объекта p и q были заменены на один объект t . В этом случае можно написать, что $Z_{N_i} = \{p, q\} \cup Z'$ и $Z_{N_{i+1}} = \{t\} \cup Z'$, где Z' - множество динамических объектов сети N_i , не

"затронутых" отождествлением $p=q$ и сохранившихся в N_{i+1} .

Докажем сначала, что $F(N_i, R_i) \subseteq F(N_{i+1}, R_{i+1})$. Для этого докажем, что для любой сети N верно

$$N \in F(N_i, R_i) \Rightarrow N \in F(N_{i+1}, R_{i+1}).$$

Пусть $N \in F(N_i, R_i)$, тогда, согласно I5°, $N_i \geq N$, и существует отображение $\Psi_i = \Psi_{N_i \rightarrow N}$ удовлетворяющее I5°). Пользуясь Ψ_i , построим отображение $\Psi_{i+1} = \Psi_{N_{i+1} \rightarrow N}$, также удовлетворяющее этому условию, и тем самым покажем, что $N \in F(N_{i+1}, R_{i+1})$:

1) положим $\Psi_{i+1}(t) = \Psi_i(p) = \Psi_i(q)$ (равенство $\Psi_i(p) = \Psi_i(q)$ следует из I5°) для Ψ_i);

2) положим $\Psi_{i+1}(x) = \Psi_i(x)$ для всех $x \in Z'$;

3) продолжим Ψ_{i+1} на $Q_{N_{i+1}}$, положив для всех xBy из $Q_{N_{i+1}}$, $\Psi_{i+1}(xBy) = \Psi_{i+1}(x)B\Psi_{i+1}(y)$.

По построению отображения Ψ_{i+1} очевидно, что для него выполняются требования 2°а, б), а значит, имеем $N_{i+1} \geq N$. Для доказательства остальной части п. I5° рассмотрим тождества из R_{i+1} . Их можно разбить на две группы: "старые" – входившие в состав R_i , и "новые" – порожденные при выполнении отождествления $p=q$. Для старых заявок выполнение п. I5° очевидно, так как по построению из $\Psi_i(x) = \Psi_i(y)$ следует $\Psi_{i+1}(x) = \Psi_{i+1}(y)$. Рассмотрим заявку $x=y$, порожденную отождествлением $p=q$. Из правила порождения заявок (см.разд. II, п.3. I) следует, что в Q_{N_i} существовали связи pBx , qBy . Отображением Ψ_i они переводятся в $\Psi_i(p)B\Psi_i(x)$ и $\Psi_i(q)B\Psi_i(y)$, и, поскольку $\Psi_i(p) = \Psi_i(q)$, мы имеем $\Psi_i(x) = \Psi_i(y)$ (в

противном случае объект $\Psi_i(p) = \Psi_i(q)$ имел бы два различных одноименных атрибута, что противоречит определению простой сети п. I°). Итак, $\Psi_i(x) = \Psi_i(y)$, откуда по построению отображения Ψ_{i+1} следует искомое равенство $\Psi_{i+1}(x) = \Psi_{i+1}(y)$.

Доказательство противоположного включения $F(N_{i+1}, R_{i+1}) \subseteq F(N_i, R_i)$ проводится аналогичным образом. На этот раз наоборот – по отображению Ψ_{i+1} строится Ψ_i :

- 1) $\Psi_i(p) = \Psi_i(q) = \Psi_{i+1}(t)$;
- 2) $\Psi_i(x) = \Psi_{i+1}(y)$ для $x \in Z'$;
- 3) $\Psi_i(xBy) = \Psi_i(x)B\Psi_i(y)$.

Далее доказывается, что Ψ_i удовлетворяет условиям п. I5°. Доказать это проще, чем в первом случае, так как все тождества из R_i , кроме $p=q$, содержатся в R_{i+1} .

20°. Необходимым и достаточным условием нормальной остановки в процессе P с начальным состоянием (N_0, R_0) является непустота множества $F(N_0, R_0)$.

Доказательство. Рассмотрим процесс P :

$$(N_0, R_0), (N_1, R_1), \dots, (N_n, R_n).$$

Утверждение о нормальной остановке эквивалентно утверждению о том, что $R_n = \emptyset$. Таким образом, необходимо доказать, что

$$[R_n = \emptyset] \Leftrightarrow [F(N_0, R_0) \neq \emptyset].$$

Покажем сначала, что

$$[R_n = \emptyset] \Rightarrow [F(N_0, R_0) \neq \emptyset].$$

Действительно, из $R_n = \emptyset$ и I6° следует, что

$$F(N_n, R_n) = F(N_n, \emptyset) = K(N_n).$$

Согласно $I9^\circ$, $F(N_0, R_0) = F(N_1, R_1) = \dots = F(N_n, R_n) = K(N_n)$, но $K(N_n) \neq \emptyset$, так как, например, $N_n \in K(N_n)$.

Теперь докажем, что

$$[F(N_0, R_0) \neq \emptyset] \Rightarrow [R_n = \emptyset].$$

Предположим, что $F(N_0, R_0) \neq \emptyset$, и предположим противное, то есть что $R_n \neq \emptyset$, что может быть только в случае авоста. Из $F(N_0, R_0)$ и $I9^\circ$ следует, что $F(N_n, R_n) \neq \emptyset$, значит, существует сеть $N \in F(N_n, R_n)$. Согласно п.15°, из принадлежности сети N к множеству $F(N_n, R_n)$ следует, что существует такое отображение $\Psi_{N_n \rightarrow N}$, что для любого тождества $(x=y)$ из R_n верно $\Psi(x)=\Psi(y)$. Теперь вспомним, в каком случае на n -ом шаге процесса может произойти авост. Это может быть, если

а) в R_n имеется тождество $a=b$ ($a, b \in A, a \neq b$) либо

б) в R_n имеется тождество вида $x=a$ (или $a=x$), где a – атом, а x – сложный объект (то есть в Q_{N_n} имеется связь вида xBy).

И тот и другой случай противоречат свойствам п.15° отображения Ψ , поскольку:

а) из $(a=b) \in R_n$ следует $\Psi(a)=\Psi(b)$, но $\Psi(a)=a$ и $\Psi(b)=b$ (см.п.2°), то есть $a=b$, что неверно;

б) из $(x=a) \in R_n$ и $(xBy) \in Q_{N_n}$ следует, что $\Psi(x)=\Psi(a)=a$ и $\Psi(xBy)=\Psi(x)B\Psi(y)=aB\Psi(y)$, что противоречит определению простой сети (из объектов-атомов не выходит стрелок).

21° . Рассмотрим множество всех возможных процессов, порожденных при недетерминированном режиме работы машины отождествлений с начальным состоянием (N_0, R_0) . Верно следующее утверждение: если хотя бы один процесс завершается нормальной остановкой, то и все процессы завершаются нормальной остановкой (см.стр.37,п.2.1').

Это утверждение следует непосредственно из 20° , если учесть, что непустота множества $F(N_0, R_0)$ зависит только от начального состояния и не зависит от конкретного процесса.

$$22^\circ. F^*(N_i, R_i) = F^*(N_{i+1}, R_{i+1}) \quad (i=0, 1, \dots, n-1).$$

Доказательство. Нужно повторить доказательство утверждения $I9^\circ$, дополнительно проверив, что из сюръективности отображения Ψ_i по построению следует сюръективность отображения Ψ_{i+1} (и наоборот).

23° . В случае нормальной остановки конечные состояния памяти всех процессов совпадают (см.стр.37,п.2.2').

Доказательство. Для произвольного j -го процесса можно написать:

$$F^*(N_0, R_0) = F^*(N_1, R_1) = \dots = F^*(N_{n_j}^j, \emptyset) = K^*(N_{n_j}^j)$$

и, поскольку $F^*(N_0, R_0)$ не зависит от j , получаем $K^*(N_{n_1}^1) = K^*(N_{n_2}^2) = \dots$, откуда следует (см. 11°), что $N_{n_1}^1 = N_{n_2}^2 = \dots$

24° . Пусть N – сеть и R – система тождеств над

N , тогда, если множество $F(N, R)$ не пусто, то оно N - выражимо (см. 9°) и равно $K(N')$, где N' - конечное состояние памяти машины отождествлений, запущенной из начального состояния (N, R) .

Примечание. Любое N - выражимое множество L описывается сетью N такой, что $K(N)=L$ (можно назвать такой способ описания множества L явным). Теперь мы видим, что сеть N вместе с системой тождеств R также описывает некоторое (может быть, пустое) множество сетей $L=F(N, R)$, будем называть такой способ описания неявным. Таким образом, задание (N_0, R_0) для машины отождествлений - это неявное описание результата. Вычислительный процесс, протекающий в машине отождествлений, - это процесс преобразования неявного описания в явное. На каждом шагу этого процесса производится некоторая локальная трансформация заданной структуры, сохраняющая ее смысл. Все это напоминает трансформационную машину Ершова /7/.

25°. Пусть N_1 и N_2 - сети, $x \in X_{N_1}$ и $y \in X_{N_2}$. Положим $U(N_1, N_2, x, y) = \{N \mid N \geq N_1, N_2 \geq N, \varphi_{N_1 \rightarrow N}(x) = \varphi_{N_2 \rightarrow N}(y)\}$. Про каждую сеть $N \in U(N_1, N_2, x, y)$ будем говорить, что она получена в результате унификации сетей N_1, N_2 относительно x, y .

26°. $U(N_1, N_2, x, y) = F(N_1 + N_2, \{(x=y)\})$.

Примечание. Из 26° и 24° следует, что унификацию сетей можно производить на машине отождествлений. Для этого нужно поместить в ДП обе сети N_1 и N_2 , в ПЗ -

заявку $(x=y)$ и запустить машину. Если унификация возможна, то есть $U(N_1, N_2, x, y) \neq \emptyset$, то после остановки машины в ДП окажется сеть, являющаяся максимальным (в смысле отношения \geq) элементом множества $U(N_1, N_2, x, y)$. В нашем случае "максимальный" означает "наиболее общий" или "наименее конкретизированный", что соответствует общепринятому понятию минимальной унификации.

Литература

1. J. Bachus. Can Programming be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs. // Communications of the ACM. - 1978. - V.21. - N 8. - PP. 613-641.
2. P.C.Treleaven, D.R.Brownbridge, R.P.Hopkins. Data-driven and Demand-driven Computer Architecture. // Computing Surveys. - 1982. - V.14. - N 1. - PP. 93-143.
3. R.A. Arvind, R.A.Iannicci. A Critique of Multiprocessing von Neumann Style. // Sigarch Newsletter, - 1983. - V.11. - N 3. - PP. 426-436.
4. J.A.Robinson. Logic Programming - Past, Present and Future. // New Generation Computing. - 1983. - V.1. - N 2. - PP. 107-124.
5. S.E.Fahlman, G.E.Hinton. Massively Parallel Architectures for AI: NEIL, THISTLE, BOLTZMANN Machines // Proc. Nat. Conf. on AI. - 1983. - PP. 109-113.

6. Тыту Э.Х. Концептуальное программирование. - М.: Наука, 1984. - 256 с. (проблемы искусственного интеллекта).
7. Ершов А.П. Смешанные вычисления: потенциальные применения и проблемы исследования.//Теория и практика программного обеспечения ЭВМ: Труды советско-французского симпозиума. Новосибирск, 1978. - С.5-40 (часть I).
8. Касьянов В.Н. Смешанные вычисления и оптимизация программ. //Кибернетика. - 1980. - № 2. - С.52-54.
9. Черных А.Н. Динамические смешанные вычисления и распараллеливание программ. - М., 1986. - 44 с.- (Препринт/ИГМ и ВТ им.С.А.Лебедева АН СССР; № 1).
- 10.L.Beckman, A.Haraldson, Oskarson, E.Sandewell.A partial Evaluator and its Use as a programming Tool// - Artificial Intelligence Journal.- 1976,-V.7,N 4, PP. 319-357.
- 11.Ершов А.П. О сущности трансляции.// Программирование. - 1977. - № 1. -С.21-39.
- 12.Ершов А.П. Трансформационная машина: тема и вариации. //Проблемы теоретического и системного программирования. Новосибирск, 1982.-С.5-24.
- 13.Попов З.В., Фирдман Г.Р. Алгоритмические основы интеллектуальных роботов и искусственного интеллекта. -М.: Наука, 1976.-456 с.
- 14.R.Hasegawa, M.Amamiya. Parallel Execution of Logic Programs Based on Dataflow Concept //Proceedings of the International Conference on Fifth Generation Computer Systems, 1984.-PP.507-517.
- 15.M.Amamiya, R.Hasegawa , H.Mikami. List Processing with a Data Flow Machine //Lecture Notes in Computer Science. - 1983. - N 147.-PP.165-190.
- 16.L.A. Lombardi, B.Raphael. Lisp as the Language for an Incremental Computer: The Programming Language LISP: its operation and applications //Information International Inc. Cambridge, 1964. - PP.204-209.
17. Базисный Рейфал и его реализация на вычислительных машинах. Методические рекомендации. - М., 1977.
18. Борщев В.Б. Схемы на клубных системах и вегетативная машина.//Семиотика и информатика. - 1983. - Вып. 22. - С.3-44.
19. Степанов А.М. Фреймы и параллельные смешанные вычисления. - Новосибирск, 1981. - 30 с. - (Препринт /ВЦ СО АН СССР; № 297).
20. Степанов А.М. Экспериментальная система программирования. - Новосибирск, 1981. - 34с. - (Препринт/ ВЦ СО АН СССР; № 305).
21. Минский М. Фреймы для представления знаний :Пер. с англ.- М.: Энергия, 1979. - 152 с.,илл.
22. Тыту Э.Х. Вычислительные фреймы и структурный синтез программ.//Изв. АН СССР. Сер. Техническая кибернетика. - 1982, - № 6.-С. 176-182.
23. Нильсон Н. Искусственный интеллект. Методы поиска решений. - М.:Мир,1973.
24. Kowalski R.A. Algorithm=Logic+Control //Communications of the ACM. - 1979. -V.22. - N 7. -PP.424-436.
25. Черных А.Н. Распараллеливание АЛГОЛа 60 в экспериментальной системе программирования. - М., 1985. - 63 с.- (Препринт/ИГМ и ВТ им.С.А.Лебедева АН СССР; № 14).

Содержание

стр.

Введение.....	3
I. О параллельных вычислениях.....	5
II. Абстрактная сетевая машина.....	23
I. Представление информации. Простая сеть.....	23
2. Основные принципы представления информации в виде простой сети.....	25
2.1. Атомы.....	26
2.2. Интерпретация атрибутов сложных объектов.....	27
2.3. Представление неполностью определенной информации.....	30
2.4. Представление типовых структур объектов и отношений. Фреймы.....	31
3. Машина отождествлений.....	34
3.1. Алгоритм отождествления.....	35
3.2. Работа машины отождествлений при решении задач.....	35
3.3. Свойства машины отождествлений...	36
3.4. Отождествление и унификация.....	41
4. Схемы. Подобие. Машина отождествлений со схемами.....	43
5. Условная конструкция. Расширенная сеть. Рекурсивные схемы. Базовая машина.....	46
6. Встроенные механизмы.....	50
7. Экспериментальная система программирования.....	54
8. Свойства АМС.....	55
8.1. Параллельность вычислений.....	56
8.2. Обратимость вычислений.....	59
8.3. Смешанные вычисления.....	62
8.4. Автоматическое распараллеливание последовательных программ на АМС методом смешанных вычислений.....	65
9. Выводы.....	67
Приложение.....	69

Андрей Михайлович СТЕПАНОВ

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ НА АССОЦИАТИВНЫХ СЕТЯХ

Отв. за подготовку рукописи к изд. О.И.МИРОНОВА

Сдано в набор 28.09.1990 г.

Подп. к печати 17.08.1990 г.

Формат 60x90/16

Усл.печ.л. 5.12. Уч.-изд.л. 3.18 Тир. 150

Зак. 1940

У. О. 15 квн.
ИТМ и ВТ АН СССР, 117333, Москва, Ленинский пр., 51



А. М. Степанов

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ
НА АССОЦИАТИВНЫХ СЕТЯХ

№ 2

МОСКВА - 1991

ОРДЕНА ТРУДОВОГО КРАСНОГО ЗНАМЕНИ
ИНСТИТУТ ТОЧНОЙ МЕХАНИКИ И ВЫЧИСЛИТЕЛЬНОЙ
ТЕХНИКИ им. С.А.ЛЕБЕДЕВА АН СССР

А.М.Степанов

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ
НА АССОЦИАТИВНЫХ СЕТЯХ

Препринт № 2 за 1991 г.

Москва