Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

ЕРШОВСКИЕ ЛЕКЦИИ

(Памяти академика А.П. Ершова)

ООО «Сибирское Научное Издательство»
Новосибирск 2009

В 2006 г. Новосибирский региональный общественный фонд информатики и искусства программирования имени академика А.П. Ершова, Новосибирский университет и Институт систем информатики им. А.П.Ершова СО РАН предложили ежегодно, в день рождения выдающегося отечественного математика и программиста академика Андрея Петровича Ершова (1931–1988), проводить публичные лекции по информатике. К чтению Ершовских лекций предполагается привлекать известных ученых – математиков, программистов, создателей вычислительных машин – тех, кто внес значительный вклад в теорию и практику информатики и в становление и развитие вычислительной техники.

Предлагаем вниманию читателей первый цикл из трех лекций, прочитанных в 2007–2009 гг. Открыл этот цикл директор Института математики им. С.Л.Соболева СО РАН, академик РАН Ю.Л. Ершов лекцией на тему «Математическая логика и теоретическая информатика», в которой он высказал мысль о взаимовлиянии этих двух дисциплин. Со следующей лекцией выступил член-корреспондент РАН, один из создателей отечественной серии вычислительных машин «Эльбрус», первый европейский ученый, удостоенный звания Intel Fellow, Б.А. Бабаян. Она называлась «История развития архитектуры вычислительных машин». И, наконец, завершала цикл лекция д.т.н. А.А Берса (ИСИ СО РАН) «Об основаниях информатики». А.А. Берс показал, что глубокое философское содержание информатики не менее важно, чем ее практическая суть. Оно определяет современную картину мира, где информация играет роль кровеносной системы. Объединенные под одной обложкой, эти три лекции дают представление о теоретической, практической и мировоззренческой составляющих современной информатики.

Лекции были записаны на видео и помещены в электронный архив Института систем информатики. Транскрипция видеозаписей лекций в текст была добросовестнейшим образом выполнена Ю. С. Кашниковым. Представленные тексты согласованы с авторами докладов.

Д.ф.-м.н. ***А.Г. Марчук***

ISBN 978-5-91124-035-6

I Ершовская лекция по информатике
19 апреля 2007 г., Академгородок
Конференц-зал ИВМиМГ СО РАН

МАТЕМАТИЧЕСКАЯ ЛОГИКА И ТЕОРЕТИЧЕСКАЯ ИНФОРМАТИКА

Юрий Леонидович Ершов
Институт математики им. С.Л. Соболева СО РАН
Новосибирск, Россия

А.А. Берс: Спасибо всем, что пришли к нам в Институт систем информатики им. А.П. Ершова. Сегодня день рождения Андрея Петровича. Новосибирский региональный общественный фонд информатики и искусства программирования имени академика А.П. Ершова, Новосибирский университет и ИСИ стали инициаторами проведения *Ершовских лекции по информатике,* которые будут читаться каждый год в день его рождения.

Директор ИМ СО РАН, академик Ю. Л. Ершов согласился открыть этот цикл и прочитать первую лекцию.

Как человек, который знает и того, и другого Ершова, я вспомнил, что когда Вас, Юрий Леонидович, выдвигали в члены-корреспонденты, выборы происходили в день рождения Михаила Алексеевича Лаврентьева, в ноябре. И Михаил Алексеевич сказал: «Пожалуйста, сделайте мне подарок, а чтобы вы не запутались, голосуйте за двух Ершовых», – вот вы оба и стали членкорами в один день. Затем, хотя и в разное время, стали и академиками.

Слово предоставляется академику Ю. Л. Ершову.

Я благодарен за представившуюся мне возможность выступить с докладом, который будет построен в несколько свободном стиле.

Ведь одной из замечательных черт А.П. Ершова было то, что он на свою профессиональную деятельность смотрел с широких позиций: и методологических, и философских, и социальных. Поэтому я думаю, что уместно в таком широком контексте и поговорить на тему соотношения математической логики и теоретического программирования. Что у них общего? И является ли одно из них частью другого? Может быть, это совершенно разные вещи?

Я готовился к лекции, но не так, как сейчас принято. Обычные доклады сейчас – это PowerPoint, темная аудитория, нажимаешь кнопку – и на экра-

не красивые картинки. Я считаю, что в публичной лекции лектор должен брать на себя ответственность и держать аудиторию как умеет: если не в напряжении, то, по крайней мере, поддерживать интерес своей речью.

Я приготовил несколько книг и хочу сделать небольшой литературный обзор. Начну с книжки, которая, может быть, не всем известна, хотя люди, сидящие здесь в зале, к ней причастны. Оказывается, есть серия книг, которая называется «Высшая информатика», хотя не исключено, что эта серия состоит пока из единственной книги.

Редакционный совет серии: сопредседатели – А. Г. Марчук и Н. Н. Непейвода. Члены совета: С. Н. Васильев, С. С. Гончаров, Б. К. Гергиль и С.С. Лавров, к сожалению, уже покойный. Не знаю, все ли члены редакционного совета знают, что они туда входят, но я спрашивать об этом не буду.

Книга Н.Н. Непейводы с соавтором И.Н. Скопиным «Основания программирования»[1] содержит много страниц. Я к этой книге время от времени буду обращаться. Я бы так сказал: эта книга претендует быть новой Библией, но вот пока еще нет вселенских соборов, которые утверждают такие книги как канонические, поэтому пока что это апокриф, но, может быть, он когда-нибудь и станет канонической книгой. Тем не менее, в ней есть глава 16 «Подведение итогов», где много всяких мудростей.

Здесь вообще много мудростей. Все ли знают теорию сфер Белопольского-Белозерского? Это достаточно любопытная вещь. Не знаю, почему Николай Николаевич ее привязывает к занятию программированием, но, возможно, какая-то связь есть?

Так вот, пункт 16 из подведения итогов: «Читайте книги, которые выпущены 20 и более лет назад, если они при первом просмотре показались вам интересными. Они дадут вам в жизни гораздо больше, чем какое-либо “Программирование в системе .NET версии 2.00 для профессионалов”, которую через два года нужно будет выбросить и забыть. Не думайте, что ваши предки были глупее вас. Опыт подсказывает, что скорее наоборот».

Я обратился к двум книгам, которые вышли 30 лет тому назад. Вышли в одном и том же издательстве «Наука» в 1977 году. Одна – книга А.П. Ершова «Введение в теоретическое программирование», вторая – моя книга «Теория нумераций».

Андрей Петрович в своей книге «Введение в теоретическое программирование» на самом деле недвусмысленно говорит о том, что он понимает теоретическое программирование как часть математики. В частности, в

[1] Непейвода Н. Н., Скопин И. Н. Основания программирования. – М.: Ижевск, 2003. – 868 с. (Здесь и далее примечания составителей).

первом абзаце он пишет: «Для автора эта книга носит экспериментальный характер. Положительным исходом этого эксперимента будет заполнение некоторого пробела в математической литературе. И более точно, теоретического программирования (называемого также в ряде переводных работ теоретической вычислительной наукой или математической теорией вычислений). Это новый раздел математической науки, чьим объектом изучения являются математические абстракции программ, предписаний, выраженных на специальных алгоритмических языках обладающих определенной информационной и логической структурой». Таким образом, Андрей Петрович вполне определенно рассматривал теоретическое программирование как часть математики.

Далее он ссылается на школу для вузовских преподавателей на Дальнем Востоке 1970-го года, мне эта школа тоже запомнилась. Там был и Николай Григорьевич Загоруйко, он помнит, что были всякие дискуссии, в том числе и горячие. И не со всеми идеями Андрея Петровича соглашались. Тогда он высказал идеи, которые были реализованы только много лет спустя. В частности, о том, что специалистов по теоретическому программированию нужно готовить по специальным программам, а большинство преподавателей ММФ НГУ, которые там присутствовали, достаточно резко этому противились и возражали.

Сейчас, как вы знаете, в НГУ есть факультет информационных технологий (ФИТ). Нужно сказать, что Н.Н. Непейвода в момент создания этого факультета сыграл немаловажную роль в формировании его учебного плана. Книга Непейводы «Прикладная логика» была написана как учебное пособие для студентов ФИТа, им было много вложено в формирование факультета, в концепцию преподавания.

Теперь я хочу вспомнить о событии, которое произошло в 1979 году в городе Ургенч в Узбекистане, где был проведен необычный международный симпозиум – «Алгоритмы в современной математике и ее приложениях». Сопредседателями программного комитета были Андрей Петрович Ершов и Дональд Кнут. На симпозиуме выступили многие известные ученые, в том числе С. Клини, я думаю, он в этой аудитории достаточно хорошо известен. Там были Ю.И. Манин, В.А. Успенский, Ян Барздинь, сотрудники ВЦ СО АН СССР, в частности, В.Е. Котов и Б.А. Трахтенброт.

Почему Ургенч? Ургенч был выбран как центр Хорезмской области, часть древнего Хорезмского оазиса, где родился известный математик Аль-Хорезми. С его именем, и с его деятельностью связаны два раздела и два важнейших понятия в современной математике, а также в информатике. От его имени произошло слово алгоритм. А от названия одной из его книг

появился термин алгебра. На хорезмской земле, где он родился, и состоялся представительный симпозиум. Принимающей стороной была Узбекская академия наук, академик В.К. Кабулов был хозяином. Это была первая и последняя конференция, на которой участников возили на автомобилях в сопровождении милиции с мигалками, а обед сопровождался национальными танцами. В этом отношении событие было уникальное. Ургенч сам по себе – достаточно провинциальный город, в сравнении со знаменитыми городами Средней Азии: Бухарой, Самаркандом и Хивой.

Симпозиум был уникален и с интеллектуальной точки зрения, понятие алгоритм обсуждалось с разных точек зрения: был довольно большой в двух частях доклад президента ИФИП Х. Земанека, в котором всесторонне рассматривалась биография Аль-Хорезми, его творчество и влияние на современную математику. У меня возникла ассоциация: последнее время из Ирака приходят тревожные новости. Вчера там больше 100 человек погибло от взрыва в городе Багдаде. Аль-Хорезми всю свою сознательную жизнь работал в Багдаде. Свои концепции и свои книги он писал в ныне несчастном, а когда-то в одном из интеллектуальных центров Мира. Такова жизнь. Все меняется.

Присутствие древней арабской культуры оказало сильное влияние на участников симпозиума. Царила непринужденная атмосфера, и было много интересных разговоров о том, что такое алгоритм, как на это смотрят математики, как на это смотрят информатики? Эта тема: «математическая логика и теоретическая информатика» фактически происходит оттуда.

Я не случайно назвал обе эти книги: «Введение в теоретическое программирование», и «Теория нумераций». В чем проблема и есть ли проблема? Есть математическая логика – часть математики и есть теоретическая информатика. Более того, есть два разных факультета, и можно точно указать: математическую логику изучают математики, а теоретическую информатику изучают специалисты по информатике.

Если взглянуть на содержание, например, журнала Theoretical Computer Science издательства Эльзевир, у которого в год выходит много выпусков, то можно увидеть многие статьи, где вводится некоторый формальный язык или рассматривается исчисление и изучаются семантики этого языка. В общем, это то, чем занимается математическая логика, и хотя математическая логика как устоявшееся учебная дисциплина выглядит немножко по-другому, потому что в результате определенной эволюции, в результате экспериментов над разными формальными языками, центральное место в современных курсах математической логики заняло исчисление предикатов первого порядка – и по заслугам. Оказалось, что это единственное исчисле-

ние, для которого справедливы и теоремы о полноте, и абстрактная теория моделей. Есть теоремы, доказывающие уникальность этого исчисления: теорема компактности, теорема о полноте. Очень четкая семантика, предложенная А. Тарским в конце 1920-х годов, сделала эту часть логики вполне математической, и математики к этому довольно быстро привыкли.

Возникновение математической логики берет свое начало от трудных дней, когда были обнаружены парадоксы. Все вы об этом слышали, но, тем не менее, чтобы не пропускать этот момент, заметим, что попытки формализации логики были и раньше. Булевы алгебры носят имя английского математика, написавшего книгу о законах логики. Это была вещь в себе некоторое время, и математики позднее ощутили необходимость математической логики, точнее, ощутили необходимость в анализе логических проблем, которые возникли в математике на рубеже 19–20 веков. Тогда успешно стала развиваться теория множеств, которая выглядела после трудов Вейерштрасса как весьма солидное основание, чтобы построить всю математику, исходя из минимального числа понятий. Но возникли парадоксы, и необходимость их анализа привела к созданию математической логики.

Я всегда привожу пример, который сегодня немножко расширю. В этой аудитории всем известны имена Джона фон Неймана, Норберта Винера и Алана Тьюринга. Первые два из них стояли у истоков развития современной вычислительной техники в США, а Тьюринг – у истоков создания соответствующих машин в Англии. Если посмотреть на их научные карьеры, все трое начинали свои занятия с аксиоматической теории множеств. То есть они начинали работать с первыми достаточно богатыми формальными языками, настолько богатыми, что с их помощью можно было изложить фактически всю математику. Таким образом, в математике фрагменты формальных языков: равенство, алгебра – давно существовали и развивались как элементы исчислений и языков, и значимость этих формальных языков, как и интегральных, дифференциальных уравнений, была хорошо известна.

Но по-настоящему богатые языки появились, когда стало понято, что можно все определить с помощью множеств. Другими словами, есть довольно бедное отношение принадлежности (эпсилон), через которое в математике можно определить практически все. Наверное, это неслучайно.

Я немножко дальше продвинусь, хотя это может расходиться с такой красивой гипотезой, что занятия формальной, аксиоматической теорией множеств приводили к машинам.

Сибирскому отделению 17 мая исполняется 50 лет. Главным человеком, создателем СО АН СССР был академик Михаил Алексеевич Лаврентьев.

М.А. Лаврентьев, будучи еще вице-президентом Украинской академии наук, был инициатором создания того, что называют Лебедевский институт – ИТМиВТ. Потом этот институт перешел в Москву. Очевидно, что Михаил Алексеевич явно осознавал важность создания такого института. Почему? Он ведь начинал в школе Лузина, занимаясь дескриптивной теорией множеств. Сам Лузин ставил вопросы континуум-гипотезы и акцентировал на том, что на самом деле причины логических трудностей лежат в природе вещей. Фактически, серьезная математическая логика в России и в Советском Союзе возникла тоже из школы Лузина. Можно вспомнить П.С. Новикова, он также из этой школы, и он, как и Михаил Алексеевич Лаврентьев, занимался дескриптивной теорией множеств, затем стал нашим известнейшим логиком. Так вот, у истоков развития вычислительной техники в Советском Союзе стоял Михаил Алексеевич, который тоже начинал свои занятия с теории множеств.

Итак, математическая логика, формальные языки не чужды ни математикам, ни народившемуся классу программистов, сейчас более массовому. Я думаю, что программисты составляют социально класс больший, чем математики, хотя не все они занимаются теоретическим программированием. Правда, я считаю, что для успешного занятия программированием нужно иметь подходящее образование. И если мы с Андреем Петровичем спорили по поводу того, какую математику надо преподавать программистам, то сам факт того, что математика должна лежать в основе теоретического программирования, – это не подвергалось сомнению, это в явном виде им написано.

К симпозиуму в Ургенче Андрей Петрович написал большой обзор по понятию алгоритма[2], а значит, и раздумывал перед этим, какие возможны обобщения, какие там расширения, и так далее и так далее. Такой солидный обзор, который вполне мог бы написать специалист по математической логике. Я зачитаю некоторые начальные фразы из этой работы, она достаточно объемная. Начинается она так: «Нас интересует абстрактная вычислимость, т. е. общая теория вычислимых функций, по отношению к которой предметная область и элементарные правила вычислений выступают в качестве формальных параметров, наделенных некоторыми аксиоматически заданными свойствами. И хотя автора этот вопрос волнует уже 25 лет, одного интереса, пожалуй, было бы недостаточно, чтобы решиться, что-то сказать по поводу этой фундаментальной проблемы, традиционно находя-

[2] Ершов А.П. Абстрактная вычислимость в алгебраических системах // Материалы Междунар. симпоз. «Алгоритмы в современной математике и ее приложениях» в 2-х частях, Ургенч, 16–22 сент. 1979 г. – Новосибирск: ВЦ СО АН СССР, 1982. – Ч. 2. – С. 194–299.

щейся в сфере и компетенции специалистов по математической логике и основаниям математики. За эти 25 лет …», – сам симпозиум состоялся в 1979 году, его «Труды» опубликованы в ВЦ СО АН в 1982 г., значит, эти 25 лет нужно отсчитывать назад примерно от 1980 года.

«… за эти 25 лет сложилась, однако, вычислительная наука (информатика, машинная математика, программирование), которая нуждается в своем взгляде на фундаментальные концепции теории вычислений. Диалог между логиками и вычислителями, характерный для этого симпозиума, должен помочь выработать вычислителям более просвещенный взгляд на основы их науки, отражающий в то же время ее особенности». Далее Андрей Петрович говорит о следующем. Есть взаимный интерес, но логики, которые занимаются понятием вычислимости, часто не очень отчетливо мотивируют интерес к своим исследованиям. С другой стороны, люди, пришедшие из программирования, часто случайным образом берут модели и методы, которые есть, так что на самом деле желательно было бы работать вместе. Андрей Петрович взялся проанализировать то, что в современной на тот момент математической логике было предложено по поводу понятия алгоритма, его обобщений, расширения и т. д.

Кроме того, есть попытки и у логиков, которые предлагают концепции сигма-программирования или семантического программирования, претендуя на то, что, возможно, их видение могло бы быть полезным и для специалистов по информатике, для тех, кто программирует. Этими логиками являются три человека: С.С. Гончаров, Д.И. Свириденко и я. У Свириденко был какой-то определенный опыт занятий программированием, но, к сожалению, наша с Сергеем Савостьяновичем надежда на то, что Дмитрий Иванович и будет двигателем, как говорится, в этом направлении, не совсем реализовалась. Однако лекции Н. Н. Непейводы показали, что эти претензии логиков очень полезно было бы предложить информатикам: они имеют под собой реальные основания. Он предложил реализованный язык СЛЭНГ, на котором некоторые классы задач решаются более эффективно, чем раньше. Поэтому определенный интерес есть.

Я уже говорил, что есть много общего и в работах по теоретическому программированию, есть формальные языки, есть семантики, понятие алгоритма. А.П. Ершов пишет, что в течение 25 лет интересовался общей теорией вычислимых функций. Я тоже хочу сказать, что для меня понятие алгоритма было и до сих пор остается таким понятием, которое я хочу понять.

Но я думаю, что есть разница в понимании у математика и информатика. Я из своей книжки, более поздней зачитаю, одну цитату. Ей уже более

десяти лет. Здесь говорится об одной из моделей обобщения понятия алгоритма. «Конечно, эта теория обобщает недетерминированную вычислимость в отличие от обобщений, основанных на расширении понятия абстрактного вычислительного устройства. Поэтому было бы справедливо указать на то, что это есть теория конструктивно познаваемых свойств, распознаваемых предикатов. Если развитие классической теории вычислимости показало, что изучать всюду определенные вычислимые функции естественно лишь вместе с частично вычислимыми функциями, то вычислимость произвольных допустимых множеств показывает, что естественным контекстом изучения частичных вычислимых функций являются вычислимые предикаты. Можно даже сказать что понятие вычислимого предиката, является более фундаментальным, чем понятие частичной вычислимой функции»[3].

Я здесь акцентирую на следующем. Для меня существо алгоритма не в его реализации, а можно сказать, в рациональной процедуре познания. Что можно познать рациональным образом? Г.С. Цейтин на симпозиуме в Ургенче говорил о процедурализмах. Видимо, это та другая сторона, которая характерна для информатиков.

В трудах симпозиума есть любопытная статья Д. Кнута, где он задается этим же вопросом и предлагает довольно любопытную модель в попытке понять, есть ли качественная разница между математиками, математическими логиками и информатиками. Он говорит, что взял девять классических книг по математике, на 100-й странице каждой книги выделил произвольное утверждение и проанализировал его с точки зрения того, какие там присутствуют алгоритмические и не алгоритмические процедуры. Он пришел к довольно любопытному выводу, что никаких особенностей там нет. Есть и алгоритмические процедуры, и не алгоритмические. В то же время он приводит интересный социологический факт, что, по его представлению, у двух процентов занимающихся математикой есть врожденное влечение к информатике, к алгоритмической природе. Это некоторое наблюдение, вопрос о психологической установке на то, что человеку интересно. Абстрагироваться от этого невозможно. Потому что если занятие не интересно, учитель или родитель может тебя в детстве заставлять, но если сердце к чему-либо не лежит, то и не лежит.

Поэтому я думаю, что одно из реальных различий лежит в психологической установке и разных акцентах, и если для математиков (для некоторых из них) важен сам факт существования алгоритма и это, в принципе уже

[3] Ершов Ю. Л. Определимость и вычислимость. – Новосибирск: Научная книга, 1996.

есть решение вопроса, то вопрос его сложности – это уже второй вопрос. В то же время, для других алгоритм, который требует полного перебора всех слов, для них это даже не алгоритм, поскольку ясно, что никакая реальная процедура это сделать не может. А вот для меня этого достаточно. Если хотя бы перебором можно сделать, значит это алгоритмически разрешимо, значит, я могу такую вещь сказать. Это моя психологическая установка.

Что касается разных формализаций, то для всего, связанного с понятием алгоритма, классическая математика предложила понятие частично рекурсивной функции. Есть тезис Черча, который утверждает, что класс вычислимых функций определен и, если оставаться на уровне вычислимых функций для арифметических понятий, то есть удовлетворительный ответ, который все признают. Для людей, которые ощущают, что надо еще и понять, как это наилучшим образом реализовать, – для них проблемы остаются. Кстати, Андрей Петрович в своей книге посвятил немало страниц проблеме экономии памяти, которой он занимался профессионально и использовал различные средства. Проблема, которая меня никогда бы не интересовала, хотя я не говорю, что она не интересна.

Алгоритм как функция – математика это удовлетворяет. С другой стороны, информатики какие-то атрибуты всегда рассматривают. Если есть алгоритм, то есть протокол вычислений, шаги, быть может, теоретическое абстрактное устройство, которое это делает. Но, тем не менее, многие из этих идей, как только они осознаны и ясно сформулированы, – они могут быть формализованы, и они снова могут быть отданы математикам. Например, абстрактная теория сложности Блюма. Она была формализована и получены очень красивые теоремы. Хотя, быть может, к реальности они и не имеют никакого отношения. После того как теория осознана и сформулирована, она может перейти в ведение логиков.

Есть и обратное. Наиболее трудные в техническом отношении теоремы в классической теории вычислимости решаются так называемым методом приоритета. Он был независимо создан А. А. Мучником у нас и Фридманом в Америке в связи с решением проблемы Поста. Там доказываются некоторые свойства рекурсивных функций, исходя из того, что есть процесс вычисления, и в зависимости от того, что делаешь, какие приоритеты расставишь, можно получить другой ответ. На самом деле полностью абстрагироваться нельзя, даже просто рассуждая про рекурсивные функции.

Есть работы по теоретическому программированию, его математической части, как говорил Андрей Петрович. И если в них и есть различие с чистой логикой, то оно состоит в каких-то неуловимых оттенках, в установках. Имеет место перетекание одного в другое. Пример: осознание Хоа-

ром динамической логики, абстрагирование и понимание того, что необходимо перейти от предусловия к постусловию. Но после того, как это формализовано, перед нашими студентами-логиками можно ставить задачу: доказать теорему Хоара и т. д.

И это правильное разделение труда. Я думаю, что из того, что происходит в настоящее время, математическая логика получила довольно большую пользу. Появились новые языки, новые семантики. Пример взаимовлияния информатики и математики: у логиков сложилось дескриптивное понимание алгоритмов, т. е. функции, которая есть, а у информатиков – императивное представление. Если говорить о семантике, то выявились два вида семантики: денотационная семантика и операционная семантика. После того, как это хорошо и правильно формализовано, математики могут развернуться и заниматься этим, что они и делают.

В отношении языков программирования. Математическая логика на самом деле в своей начальной фазе имела довольно большую экспериментальную часть. В то время возникли языки лямбда-исчисления, тогда же возникла комбинаторная логика. В результате эволюции, пока пользователями являлись только математики, они отбраковали их как слишком сложные, с непонятной семантикой, и, в результате, отобрался и выкристаллизовался весьма достойный инструмент – исчисление предикатов первого порядка, ИП-1.

После того как потребности той же информатики показали, что нужны языки более высокого порядка, может, даже бесконечного, с самоприменимостью, то оказалось, что уже есть чистое лямбда-исчисление как пример абстрактного языка, который отражает многие существенные свойства языков программирования. Оно вернулось в обиход, и математики приняли совсем нетривиальное участие в его развитии. В частности, Д. Скотт предложил нетривиальную модель для денотационной семантики, которая поначалу казалась темной. Поэтому логики в лице информатиков имеют, может быть в неявном виде, коллег, а информатики являются источником разных задач, служащих вдохновением для логиков.

Находят ли информатики источники вдохновения для себя среди задач логиков? Мне трудно судить, я не являюсь информатиком. Я знаю, что Андрей Петрович с уважением относился и изучал обобщения алгоритмов. Для меня довольно любопытным является опущение той модели обобщенной вычислимости, которая для математиков является наиболее принципиальной, а именно, вычислимости в допустимых множествах. Хотя в то вре-

мя (1975 год) книжка Барнайса уже была[4]. Он ссылается на работы Крайзеля, отмечает важность его идей об инвариантной определимости, но работы Крайзеля и Сакса по метарекурсии остались вне его внимания и вне его интереса. Я думаю, что это тоже в какой-то мере не случайно, поскольку там нет процедурализма – только сигма-вычислимость, предикаты.

Подытожу вышесказанное. Я, как и Кнут, но по прошествии куда большего времени, не мог бы сформулировать четко, что в работах по теоретическому программированию отличает подходы математиков от подходов информатиков, кроме установки. Часто информатики занимаются задачами, поставленными в логике, и логики занимаются задачами информатики. И мне кажется, что в отношении формальных языков, исчислений, семантик каких-то реальных противоречий или непониманий просто нет. Всегда такие сравнения слишком больших вещей довольно трудно делать.

Мои дальнейшие рассуждения будут не очень серьезными. Что значит: не очень серьезными? На уровне интеллектуального трепа. Вот любопытная задача, которую поставил Станислав Николаевич Васильев, который до недавнего времени был директором иркутского Института динамики систем и теории управления, а сейчас уехал в Москву и возглавил Институт управления. Он задался следующим вполне конкретным вопросом. В России, в Советском Союзе, было создано много нетривиальных алгоритмов. По прошествии времени, когда языки изменились, а автор недостаточно специфицировал программу, они стали вещью в себе и практически недоступны. А жаль, потому что некоторые из этих программ являются произведениями искусства.

Он поставил задачу в духе русского философа Николая Федорова, который выдвинул идею, что можно воскресить всех предков в полном объеме, никто из человечества не будет утерян. Станислав Николаевич сказал, что если остался текст программы, то можно восстановить всю функциональность программы со всеми достоинствами, представив текст на современных языках. Во-первых, я не думаю что это так, а во-вторых, что так и нужно делать.

Тем не менее, сейчас я обращусь к области исследования, вдохновленной Непейводой. Объективным является следующее: от появления информатики, не теоретической информатики, а информатики в целом, от появления вычислительных машин и информационных технологий в нашем обществе, оно – наше общество – сильно изменилось. Во-первых, появи-

[4] Гильберт Д., Барнайс П. Основания математики. Логические исчисления и формализация арифметики. – М.: Наука, 1979.

лось киберпространство или то, что называется виртуальным пространством, которое не такое уж и виртуальное: многие взрослые, а еще больше дети уходят в это пространство и, может быть, даже не возвращаются. Это есть нечто реальное.

Во-вторых, субъектами этого виртуального пространства являются программы и программные комплексы, к которым можно относиться и, в частности, говорить о них, как о живых существах. Если говорить об алгоритме не в теоретическом смысле, а о какой-нибудь реализованной действующей программе, то у нее есть цикл развития: программа сначала создается, потом отлаживается, потом находятся ошибки, потом она морально устаревает. Это цикл ее жизни. Если раньше программирование было в кодах машины, то потом появились языки программирования, затем языки спецификации. Спектр языков, который идет от абстрактной задачи к реализации, на самом деле весьма велик и т. д.

Программа имеет свой жизненный цикл, и вопрос состоит в следующем: а что в конце этого жизненного пути от программы должно остаться с точки зрения общества? Чтобы все осталось – это невозможно. Тем не менее, люди, которые эти программы создавали, корректировали и сопровождали, вложили в них много интеллектуальных усилий, много находок, иногда совершенных и красивых. И то, что все это растворяется бесследно, это обидно, вообще говоря. Поэтому предложение могло бы состоять в следующем: Есть серьезные программы, которые специфицированы, документированы, а главное – работают. Может быть, стоило бы продумать регламенты, определяющие сущности, которые должны остаться от этих программ?

Если сопоставить, то есть сущности программы, а есть люди и их сущности. Я подумал, а что от человека остается: есть личное дело в отделе кадров, есть история болезней в поликлинике. Я вижу в этом довольно тяжелую иронию, например, с точки зрения пессимизма медицины. Не история преодоленных болезней, а история болезней. Если говорить о личном деле, то, как мы прекрасно понимаем, личное дело и личность – это тоже разные сущности, не будем говорить о других записях – единая запись нарушения ПДД и т. д. На самом деле человека сопровождает много записей и очень многие из них формализованы. Чтобы получить тот же медицинский сертификат, визу в другую страну, степень, звание и пр. Жизнь человека сопровождается массой обязательных записей. Поэтому я думаю, что наряду со спектром языков, которые ориентированы на создание программ, должен быть другой спектр языков, которые описывают на разных уровнях

приближения, что же там есть, какие идеи там есть. Нахождение правильного уровня средств – это задача весьма и весьма интересная.

Есть и другая задача. Мы много сейчас говорим об интеллектуальной собственности. Объектом интеллектуальной собственности может быть песня, статья, название, программа и т. п. Часто в технических устройствах есть некоторые «фишки», есть находки, есть уникальные идеи, которые иногда патентуются. Конечно, могут быть изделия, в которых ни одной новой идеи, зато они хорошо работают – там патентовать нечего. Это и к математике относится, и к информатике. Есть сама теорема, а есть ее доказательство. Конечно, мы понимаем, что есть одно доказательство, есть другое, которое может быть принципиально другим, которое может быть более интересно и т. д. Но не всегда в интересном доказательстве есть какие-то идеи.

И я бы сказал, что важно умение или постановка задачи, как выделять вот такие «фишки», те идеи, которые внутри есть, чтобы они не гибли. С одной идеей можно тысячи теорем доказать, а идея – она одна и та же. И она ведь иногда даже не сформулирована в явном виде. Нахождение патентуемых интеллектуальных находок – это задача, которая и непроста для науки, и актуальна. А для людей, которые создают программы, где имеет место комбинация науки и инженерного искусства, они тоже весьма важны. Поэтому нужно создавать механизмы выявления и сохранения, может быть, патентования вот этих действительно новых оригинальных идей.

ВОПРОСЫ:

П.Г. Емельянов (ИСИ СО РАН): Я хотел узнать, интересуется ли математическая логика задачей, известной как «P равно NP или не равно». Я задаю этот вопрос, потому что алгоритмисты, которые работают над реальными проблемами, говорят, что для них статус этой проблемы вообще не важен. Они для себя умеют решать сложные задачи с точностью такой-то и получать полиномиальные алгоритмы, которые хорошо работают. Поэтому они этой проблемой не занимаются – она им не интересна. И вообще, есть такое мнение, что она и не является алгоритмической проблемой, а скорее, лежит ближе к общей теории вычислимости. Вы бы могли сказать о статусе этой проблемы в рамках математической логики?

Ответ: Я думаю, что с точки зрения математической логики эта проблема очень четко и понятно сформулирована без всякого противоречия. Вопрос о том, кого она за сердце трогает, а кого нет. Это в любой деятельности, как Вы знаете, кого-то трогает, а кого-то нет. Хотя когда она стала

миллионником, т. е. за ее решение предлагают миллион, она, может быть, трогает сердца бóльшего числа людей. С точки зрения ее статуса она очень понятна, и я не удивлюсь, что многие попытаются ее решить, хоть я не знаю конкретных таких людей.

Есть ведь стандартные уловки, есть опыт ее решения при некоторых предположениях. Если проблема не решается, то ее начинают решать в близких условиях, чтобы лучше понять. В других предположениях, с оракулами или иначе, не знаю. И она решается, и получается вполне однозначный ответ. Я готов с вами согласиться, что для ее решения нужны не какие-то интеллектуальные ухищрения, а другой философский взгляд на вещи.

Вот тот же Н.Н. Лузин, которого я упоминал, он, говорят, однажды пришел читать свой спецкурс и сказал: «А вы знаете, чему равна мощность континуума? – алеф-17». С одной стороны, это наполовину шутка, с другой стороны, сейчас известно, что это ничему не противоречит. Это понимание или осознание того, что некоторые техники, которые сейчас пытаются применить, они внешние, они не зацепляют существо самой проблемы, и я допускаю, что, может быть, осмысление ее и видение в некотором соответствующем контексте может привести к простому решению, если правильно взглянуть на нее. Но это из разряда гипотез. С точки зрения правильности постановки, это вполне нормальная и непротиворечивая постановка, которой могут и математики и информатики заниматься, кого это волнует. Сам я уже признавался, что меня перебор не пугает – я этим не занимаюсь.

Д. Пономарев (ИСИ СО РАН): Вопрос об аксиоматическом методе, который успешно применяется в математике и для которого известны ограничения. Сейчас его начинают применять для описания разных нематематических объектов. Как Вы относитесь к этому?

Ответ: Я тут принес одну книжечку – «Логическая физика». Автор очень известный человек: недавно скончавшийся философ Александр Александрович Зиновьев. Бывший диссидент, вернувшийся в Россию. Ну, в общем, яркий такой человек. Зиновьев хоть и говорил, что его сильно обижали, написал шесть или семь книжек в 1960–1970 годы. Одна из них «Логическая физика», в которой предложен формализм для физики. И вот одна из бед, я когда-то даже философскую статью об этом написал,– что когда исследование заканчивается формулировкой исчисления и больше ничего, то, на мой взгляд, – это, мягко сформулирую, неправильно. А это то, чем грешит философская логика или формальная логика, то, чем занимается философия, – они считают, что если они какое-то явление пообсуждали на полуформальном уровне, предложили какой-то формальный язык и даже ка-

кое-то исчисление, то этим задача решена. Я большой пользы в такой деятельности не вижу.

Чем важны и хороши формализмы? Тем, что можно, когда мы формализовали, т. е. отсекли какие-то внешние части и сделали это правильно, то упростили ситуацию. После этого давайте возьмем и воспользуемся этой пользой, сделаем выводы и т. п.

Вот деятельность А.И. Мальцева, я про локальную теорему компактности уже говорил. В его деятельности заметна одна вещь – это в какой-то ситуации введение алгебры иногда даже странной сигнатуры и такой, что в ней есть какие-то тождества. Это выглядит абстрактно, а с другой стороны, с вычислительной точки зрения, алгоритм, который называется переписывание термов, – на самом деле один из наиболее быстрых. И когда можно сформулировать задачу не в виде исчисления, а в алгебраической части только в виде тождеств, часто получается потрясающий эффект, на самом деле. Например, есть две сложные работы: одна из конечных групп, а другая – из алгебр Ли. А.И. Мальцев ввел формализм и на трех страницах получил результат, из которого следуют обе эти работы.

Суть подхода Анатолия Ивановича к развитию алгебраических систем не в том, что он общее понятие алгебраической системы ввел, а в том, что он сконцентрировал внимание на двух вещах. Начиная с многообразий, а это правила переписывания термов, а потом, я это относительно поздно понял, изучение квазимногообразий, а это есть квазитождества. Язык Пролог на самом деле – это квазитождества, и Анатолий Иванович с алгебраической точки зрения осознал, что представление какого-то объекта сначала в виде многообразий, а затем, когда этих средств не хватает, то в виде квазимногообразий, – это эффективный способ изучения этого объекта. Поэтому после формализации должен быть анализ, нужно получать следствия, а иначе это бессмысленно. Я не знаю, может, на философском уровне можно спорить: я формализовал так, а я так, кто лучше, – ну, не знаю.

А. Г. Марчук (ИСИ СО РАН): Есть такая сейчас уже непопулярная задача – называется искусственный интеллект. Вообще говоря, интеллект в человеческом представлении очень связан с понятием логики, обычной человеческой логики. Потом алгебраисты предложили механизм известный как логический вывод, системы продукций. То есть в этом плане алгебраисты, не занимаясь данной областью, тем не менее, может быть, самый большой вклад в этом направлении сделали. Мой вопрос заключается в том: это как-нибудь продолжается? т .е. есть ли целенаправленные шаги в сторону анализа особенностей мышления?

Ответ: Я не отслеживаю этого сейчас, хотя могу сказать, что у меня в кабинете целых две полки книг, которые называются «Введение в искусственный интеллект». Мне представляется, что это попытки сформулировать точно вопрос. Я считаю, что тест Тьюринга про диалог с машиной: «Можно ли так запрограммировать, что ты не узнаешь, что ты разговариваешь с машиной?» – это хорошая модель того, что можно считать искусственным интеллектом. Если возможно такое программирование машины, по крайней мере, для ограниченного класса вопросов, то это и есть искусственный интеллект. С другой стороны, существует ряд технических вопросов. Я говорю, что самых последних вещей я не знаю, но ответ может быть совершенно неожиданный.

Может быть, большая программа это и есть, грубо говоря, искусственный интеллект. Уже, в сущности, живущий. Суть ведь не в имитации, чтобы быть похожим, а в том, чтобы на самом деле эффективно выполнять некоторые функции. В чем состоит развитие техники? Ведь ясно, что экскаватор сильнее человека. В том, что многие интеллектуальные действия человека могут быть переданы информационным технологиям, например, шахматная программа – в этом для меня никаких сомнений нет, вопрос, до какой степени. Если начальная постановка состоит в том, чтобы проимитировать все до мельчайших подробностей, то, по-моему, это какая-то неинтересная задача. С другой стороны, понятно, что колесо для транспорта важнее, чем ноги. Можно все движение сделать, имитируя ход ногами, но ясно, что эффективнее на колесах ездить. Искусственный интеллект зависит от определения. Я считаю, что те радужные наивные взгляды на искусственный интеллект, они немножко отходят в сторону, а реальная передача интеллектуальных функций, отчуждение интеллектуальных функций в форме информационных технологий – оно происходит.

Н. Г. Загоруйко (ИМ СО РАН): Вопрос по поводу взаимного влияния, проникновения математики и информатики. Скажем, вероятностники уже воспринимают как нормальный факт, как серьезно обоснованное доказательство результат, полученный путем машинного моделирования. На большом количестве экспериментов, что-то там подтверждается какая-то там вероятность, подтверждается регулярно и это считается результатом, сравнимым по своей значимости и достойности с доказательством теоремы. В логике в алгебре я знаю, что есть факты доказательства с помощью машинного моделирования, там доказывались некоторые вещи для больших, просто переборных задач. А как в сообществе логиков котируются такие результаты? Считается ли, что это нечто достойное данного сообщества

или это уникальный частный факт, а вообще серьезным логикам заниматься этим не рекомендуется? Студентов к этому не призывают? Так или не так?

Ответ: Во-первых, нет однородного сообщества математических логиков. Логики разные есть и по-разному относящиеся к разным вопросам, поэтому я могу ответить только за себя. Есть такой знаменитый пример, который волнует философов, и у нас есть продолжающийся грант с Институтом философии, – это проблема четырех красок.

Известно, что она была решена с помощью машинных программ. Она была сведена к большому числу случаев и к их перебору. Если там возможна раскраска, то тогда и все возможно. Это машина и сделала. Это было объявлено, более того, опубликовано в математическом журнале. И кому-то там не нравилось, да, у машины могли быть сбои. Потом была другая программа, которая это сделала, т. е. убедительность была добавлена. Но есть другая сторона этого вопроса.

Я в 1974 году был на конгрессе математиков в Хельсинки, и там Хакен делал доклад, у которого один из авторов не математик – инженер. Кажется, это Хакен и есть, и он решил нас математиков поучить. Он сказал, что его вообще удивляет, что утверждения, про которые уже доказано, что они в 99,99 % случаев справедливы, не принимают как теорему. Он был искренним. Но это для математика неприемлемо. Вы говорите о статистике, так там сам подход с точностью до вероятности и т. д. Но это не относится к пересмотру вопроса о том, доказана ли теорема, или нет, если она верна в 99 и миллион девяток процентов случаев. Мой ответ – НЕТ, потому, что один контрпример порушит всю конструкцию.

Вопрос – как относиться к такому использованию машин? Я отношусь спокойно. Я считаю, что это как раз помощь, интеллектуальная помощь. Как экскаватор. Когда надо выкопать большую яму, я могу сам копать, но могу позвать экскаваторщика, который это сделает. Поэтому я отношусь к этому нормально, и более того, я считаю, что это неизбежно.

Почему? Потому что меня сейчас на другую философию потянет, но, тем не менее, я пропагандирую эту точку зрения, это вопрос о познаваемости мира. Нас учили по диалектическому материализму, что все познаваемо. Но что значит познаваемо? Предполагается, что каждый человек сможет познать, имея нормальные умственные способности. Но, тем не менее, тут есть одна некоторая из неявных предпосылок, что вот все эти истины – они должны быть достаточно короткими, чтобы человек просто мог ухватить их целиком.

А если эта истина объективно такова, что у человека физиологических возможностей нет для ее познания. И можно ли утверждать, что всякую

истину можно свести к короткой, или аппроксимировать сколь угодно хорошо короткими истинами. Это допущение, которое, когда оно явно сформулировано, то я не думаю, что оно всем покажется очевидным. А оно присутствует. Как можно познать простое число, содержащее миллион знаков? Если считать, что у нас есть машины, мощность которых увеличивается, то и растет класс истин, которые мы можем узнать с их помощью, а без их помощи даже и не сможем. Я считаю, что так машины использовать необходимо и надо привыкать работать с ними в паре.

Не нужно вспоминать начальные времена, Н. Винера или С. Лема, когда боялись робота. Кстати, Роджер Пенроуз в некоторых своих писаниях возрождает эту атавистическую боязнь, что машины станут умнее человека, захватят власть, и люди у них будут рабы, – это настолько наивно и подетски выглядит. Машины – это наши партнеры, мы их создаем, чтобы они решали то, чего мы не можем сделать. Раз уж мы их создали, так давайте будем им доверять.

С другой стороны, есть проблема увеличения доверия. Есть доверие к написанным обычным способам рассуждениям. Есть книжка Адяна «Решение проблемы Бернсайда» – 300 страниц. Есть другой текст, классификация конечных простых групп, там общий объем текста, который нужно собрать, более 1500 страниц. Какова степень убедительности этого текста, написанного разными людьми? Один человек это все равно не сможет воспринять. Поэтому проблемы есть и в восприятии обычных таких вещей. И как проверить, что на самом деле в каком-то месте нет опечатки? Ошибки и в обычных математических статьях тоже есть.

Другое дело, к чему надо стремиться. Я думаю, что математики от принципа не принимать за доказательство, если в 99 случаях она верна, никогда не отступят. Это существо математики, существо понятия «доказательство». А использовать машины, как дополнительные средства – это возможно.

В. И. Шелехов (ИСИ СО РАН): Вопрос относительно истории, столкновений, которые были раньше между программистами и логиками. Когда я в 1971 году пришел после университета в институт, то застал отголоски тех споров. Но к этому времени спорящие стороны разошлись по разным ученым советам и споры, в общем, как-то прекратились. В электронном архиве А. П. Ершова некоторая информация об этом есть. И меня как раз интересует отрезок времени предшествующий. Вы в своем докладе касались уже более позднего, спокойного состояния. Ситуация такая, что в то время программисты пришли в математику с нестандартной математикой, чем, есте-

ственно, вызвали раздражение у традиционных логиков. Происходили конфликты. Меня интересуют ваши ощущения того отрезка истории.

Ответ: Если даже не очень далеко спуститься в историю, то можно увидеть, что и математическая логика пришла в математику нежданной. Ее в математике никто не ждал, все относились к ней ровно так же, если не хуже. Какие насмешки там у Пуанкаре по поводу того что, «видите ли Пеано какие-то там крючочки пишет?». Умный человек был, исключительно умный. Это нормально, это процессы привыкания к чему-то необычному. Здесь процессы, связанные не только с программмистами. Например, маткибернетика. Маткибернетиков с программистами вот так прямо связать нельзя, у них там свои задачи, есть общая часть – теория автоматов. Потом тоже разошлось, это проблемы роста и они совершенно нормальны.

Я уже упомянул про конференцию 1979 года, и Николай Григорьевич подтвердит, в чем я категорически был не согласен с Андреем Петровичем. Он тогда заявил следующее – будущих информатиков математике нужно учить, но им не нужно читать матанализ, вместо матанализа им нужно читать теорию графов. Вот что он сказал. Я тогда это не принимал и сейчас это не принимаю, и считаю, что это не верно. На него позитивно повлияло, насколько я понимаю, общение с Бауэром из Мюнхена.

С другой стороны, сам Андрей Петрович окончил Московский университет, мехмат. Поэтому когда кто-то мне говорит, что химикам и биологам математику учить не надо, как мне В. Н. Пармон сказал, который сам-то окончил Физтех, то я к этому не могу отнестись серьезно. Я ему отвечаю – ты-то сам учил, тебя что, испортили, что ли? Не могу я такое понять. Также и Андрей Петрович, тоже учил матанализ и, может, прямой связи с машинами и не видно, но на самом деле изучение высокой математики – это выработка навыков системных понятий.

II Ершовская лекция по информатике
19 апреля 2008 г., Академгородок
Конференц-зал ИВМиМГ СО РАН

ИСТОРИЯ РАЗВИТИЯ АРХИТЕКТУРЫ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

Борис Арташесович Бабаян
Корпорация Intel
Москва, Россия

Хочу сказать, что мне было очень приятно принять это приглашение. Это для меня большая честь, и я очень доволен, так как вижу в зале много молодежи, но все-таки встречаются и знакомые лица, которые участвовали в той истории. История была очень интересной и важной. Я буду рад рассказать вам не только про историю, но и про тот дух, который царил в то время. Со мной приехал Владимир Пентковский, который является активным участником тех событий. Мы с ним разрабатывали первый и второй компьютеры «Эльбрус». Он перешел в «Интел» раньше, чем я, – лет 15 тому назад. И надо сказать, что он там достиг достаточно высокого уровня, он руководил группой архитекторов, которые работали над «Пентиумом-3» – одной из наиболее успешных машин «Интела». Это тоже следствие тех времен, так как он и образование, и опыт получил в нашей стране.

Прежде всего, я хочу рассказать о техническом содержании деятельности, которая происходила в то время. Я не хочу сказать, что сейчас время не то, сейчас тоже выдающееся время. Но то время было особенно выдающимся и впечатляющим.

Я прежде всего хочу вспомнить дух того времени. Этот дух был высоко новаторским, потому что все горели желанием работать в области вычислительной техники, когда она только-только появилась. И, надо сказать, работали активно. Присутствовал соревновательный дух, было много споров, разговоров, и, самое главное, у нас была работа, которая была востребована обществом, – это были восхитительные дни! Для многого из того, что сейчас делается, основы были заложены тогда. Я постараюсь это технически раскрыть.

То время очень было очень плодотворно потому, что много делалось нового. Появилось много выдающихся личностей – харизматических фигур. Одна из таких фигур – это, конечно, Андрей Петрович Ершов. Другим был Сергей Алексеевич Лебедев. С. А. Лебедев работал в Москве, а Андрей

Петрович здесь в Академгородке. Необходимо упомянуть третью фигуру – Святослава Сергеевича Лаврова, который трудился в Ленинграде.

Мы все работали в очень тесном содружестве, много общались, много говорили. Новосибирск, в каком-то смысле, выделялся, потому что это был центр вселенной программирования вычислительной техники. Сюда все стремились, приезжали со всех концов страны, здесь была масса обсуждений, проводились серьезные работы. Их, может быть, сейчас недооценивают, но их влияние очень серьезное. Центром этой деятельности в Новосибирске был Андрей Петрович.

Прежде чем рассказывать про архитектуру, про наши работы, я хочу просто подчеркнуть важность работ Андрея Петровича. Если вспомнить самое начало, когда машины только появились, программирование на машинах было загадкой, чем-то очень непонятным с практической точки зрения. Тогда, очень давно, на заре вычислительной техники появились языки программирования. Я хорошо это помню, существовало серьезное мнение, что настоящие мужчины, настоящие программисты программируют на ассемблере, а языки, это так – баловство.

Деятельность Андрея Петровича привела к опровержению этой истины, потому что тогда, насколько я помню, и как помнят люди, сидящие здесь, появился Алгол, т.е. язык на котором, безусловно, значительно удобнее было программировать, чем на ассемблере. Однако существовало мнение что, это конечно очень удобный язык, но оптимизирующий компилятор на нем построить абсолютно невозможно и нечего за это дело и браться. А Андрей Петрович сделал оптимизирующий компилятор на языке Алгол. Он показал, что это не только удобное программирование, но и эффективное, т. е. это был, безусловно, важнейший шаг в развитии всей вычислительной техники.

Сейчас то, что я сказал про программирование на ассемблере, вызывает смех, молодежь спросит: как это программировать на ассемблере? Ну, конечно, можно на нем какие-то вещи программировать, но сейчас в основном программирование идет на языках. И программирование осуществлялось даже не на ассемблере, а в кодах: нулики и единички. Я хорошо помню вариант, когда наши машины употреблялись на полигонах и т. д. У нас тогда был хороший программист, и он программировал в кодах. Так если для машины требовалась константа сдвига, он подбирал одну из команд, где младшие разряды совпадали с этой константой, и использовал ее как константу сдвига. Когда появились алфавитно-цифровые ассемблеры, даже мне казалось, что это потеря эффективности, как это константа будет занимать целую ячейку памяти? То есть все это прошло гигантский путь разви-

тия, и Андрей Петрович сделал шаг, конечно, фундаментальный. Насколько я помню, первый транслятор с Алгола был 29-проходным, поэтому сделать его было сверхтрудно, не только потому, что надо было показать, что Алгол можно оптимально использовать, но потому, что это надо было сделать с малюсенькой памятью, это просто десятикратное усложнение работы. Тем не менее, этот подвиг был совершен, что просто великолепно.

Андрей Петрович – многогранная фигура. Он создал мощный коллектив, который сейчас находится здесь. Он внимательно относился к сотоварищам, к молодежи, поощрял их развитие. Много внимания он уделял школьной программе. Кроме того, здесь в Академгородке велись работы не только по языкам, но и по операционной системе, и велись эти работы довольно серьезно. И организовал все это Андрей Петрович.

Он был не только фигурой, глубоко погруженной в свою специальность. Как-то мы были в командировке, и он мне подарил свои стихи, известный перевод Киплинга. Недавно, готовясь к выступлению, я нашел книжечку с его стихами. Великолепные стихи! Он был выдающейся личностью. Я очень доволен и безмерно рад, что мне посчастливилось с ним много работать и общаться. И я к нему приезжал в Академгородок, и он ко мне в Москву. Он был очень коммуникабельным. Мы сегодня вспоминали, что он был каналом связи с мировой общественностью. Вы все знаете, что здесь побывали все выдающиеся личности в области вычислительной техники со всего мира. Здесь был Маккарти, автор Лиспа, здесь был Дейкстра, были Хоар, Вирт, – ну все, кто представлял высший уровень мира Информатики. Это, конечно, тоже говорит о высокой оценке деятельности Андрея Петровича, о высокой оценке деятельности всего новосибирского вычислительного сообщества.

Я горд тем, что внесу свою лепту в память об этом замечательном человеке. Наша деятельность происходила в очень близком контакте с новосибирскими специалистами. Вы, наверное, знаете, что Московский институт точной механики и вычислительной техники учредил здесь свой филиал, с которым мы работали в тесном контакте и, конечно, в тесном контакте и с Андреем Петровичем.

Другой такой же выдающейся фигурой был Сергей Алексеевич Лебедев в Москве. Мне тоже повезло тесно сотрудничать с ним. Он был пионером, начинал делать машины на заре вычислительной техники. Пока вычислительная техника не перешла в коммерческую плоскость, пока фирмы не начали получать от нее доходы, наши разработки развивались довольно близко с западными. Но как только подключились фирмы, Россия, конечно, отстала. Это просто воспоминания о том времени. Важно то, что все, зало-

женное в то время – это наши сегодняшние успехи, это следствие тех работ, которые были тогда выполнены.

Перейду к описанию технической стороны нашей деятельности. Я всю жизнь работаю в области вычислительной техники, первая моя студенческая работа относится к 1954 г. До сих пор я активно занимаюсь разработкой архитектуры, фактически, я в жизни делал только один проект, проект по совершенствованию архитектуры вычислительной машины. Конечно же, этот проект имел много поколений.

Я постараюсь вам рассказать о разработке архитектуры отечественной вычислительной техники и развеять существующее мнение о том, что Россия, наше техническое сообщество, наша вычислительная техника отстали, некоторые даже употребляют выражение «навсегда». Я хочу показать, и это мое глубокое убеждение, что за все это время мы в области архитектуры никогда не отставали от Запада. И даже во многом и намного опережали Запад. Сейчас настало время, когда та фундаментальная работа, которая была проделана в прошлом, стала активно влиять на развитие вычислительной техники во всем мире.

Начну с зари вычислительной техники. 1950-е годы, середина, я тогда был студентом физтеха, куда поступил в 51-м году, когда словосочетания «вычислительная техника» еще не существовало. Поступал я на специальность «машинная математика». Я думаю, хотя исследования на эту тему я не проводил, в России, в СССР наша группа студентов из 12–13 человек в области вычислительной техники была первой. Но я подозреваю, что и во всем мире в том 51-м году мы были первые. Когда я встречаюсь с западниками, никто меня не опровергает, но и, конечно, никто и не сознался, что на Западе такого курса тогда не было.

Вычислительной техникой я начал заниматься с самого начала. Машины тогда были, конечно, в основном, последовательными. Слова «последовательный» и «параллельный» за период развития вычислительной техники сильно изменили свой смысл. Тогда «последовательный» или «параллельный» означали, как обрабатывать арифметику, разряд за разрядом или слова целиком. Сейчас слово «параллельный» имеет совсем другой смысл.

Сергей Алексеевич был строгим сторонником параллельной работы. Под параллельной работой тогда понималось сложение или умножение со всеми разрядами сразу, и у нас были дискуссии внутри института. Тогда были машины, которые последовательно с барабана побитно считывали, начиная с младших разрядов по два числа, складывали их и по биту писали сумму на тот же барабан. Вы, наверное, и не знаете, что это такое. Бара-

бан – это предшественник дисков. Сергей Алексеевич тогда начал строить параллельную машину.

БЭСМ-1 и была первая параллельная машина. В ИТМиВТ она занимала несколько комнат. Перенос начинался в одной комнате, а заканчивался в другой (это такое было сложение). Умножение происходило столбиком и деление так же. Поэтому тогда весь фокус архитектурной работы состоял в том, как ускорить арифметическое устройство, так как основной частью всей вычислительной машины было арифметическое устройство, а все остальное – небольшие добавки. Представьте себе, что один бит сохраненной информации занимал 1 кубический дециметр. Такой блок работал на двух лампах больших размеров.

Первая моя работа (курсовая работа 1954 г.) была посвящена быстрому выполнению арифметических операций. Сложение с переносом было довольно понятно, это было как групповой перенос. Трудно было с умножением, делением и извлечением квадратного корня. Тогда будучи студентом, я разработал метод, который сейчас сильно развит, выполняется на всех машинах и называется carry-safe (carry-safe-adder, multiplier, divider, square root extractor).

Мы ее называли работой в 2-рядовом или многорядовом коде. Когда складываются два числа, то это делается поразрядно. Кроме того, должны осуществляться переносы, причем, в худшем случае, перенос может бежать по всем разрядам от младшего до старшего. Однако, оказывается, что если аргументы и результат представлять в двухрядном виде, то переносы можно запомнить в верхнем ряду, а их осуществление отложить на потом, например, 11 + 2 = 13 при поразрядном сложении дают:

0000 0000 0010
1011 + 0010 = 1101.

Таким образом, можно складывать такие двухрядные представления все время поразрядно, накапливая информацию о переносах в верхнем ряду, и только в самом конце сложить нижний ряд окончательного результата с верхним, используя сложение с переносом.

Умножение в 2-рядовом коде представляет собой серию сложений. Можно каждое сложение сделать с переносом, но это займет много времени. Можно поступить по-другому, использовать два ряда, т. е. сделать все сложения без переносов. Или же можно складывать по три числа сразу, причем держать переносы наверху.

Значительно сложнее было придумать беспереносное деление, но это тоже было придумано. По старшим разрядам угадывался приблизительно

ответ, а затем корректировался. Эта коррекция и будет вторым рядом в частном.

Все это было сделано в комплексе в 1954 г., а в 1955 г. был сделан доклад на физтеховской конференции. А первая западная публикация появилась в 1956 г., ее автором был Адлер. Так что формально, внешне мы тогда уже на год опережали Запад. Сергей Алексеевич очень активно принимал участие в этом, я ему рассказывал, и мы вместе обсуждали.

Теперь взглянем на продолжение, на все эти методы в общем. Вся современная арифметика основывается на двух идеях: это многорядовый код и использование другого основания счисления (в нем я не принимал участие), т. е. умножение не в двоичной системе, а скажем в четверичной, восьмеричной, что сокращает число сложений. Мой метод позволял быстрее складывать, а переход к большему основанию позволял еще и сократить вдвое число сложений. С тех пор прошло более 50 лет, а ничего нового не было предложено в этой области. Можно сказать, что это хорошо поставленная задача, решенная до конца. Конечно, было бы интересно подумать о предельности этого метода. По-моему, какие-то работы я встречал в МГУ, но они не получили развития. То есть за 50 лет никто ничего нового не предложил, хотя во всем мире гигантское количество ученых работало в этой области.

Я не хочу сказать, что разработка арифметических блоков превратилась в ремесленное поприще, поскольку каждые новые технологии предъявляют новые требования к элементной базе, и эти два метода постоянно приходится адаптировать к технологии, к числу разрядов – это искусство. Но один из этих теоретических методов, конечно, был придуман в России. Это был первый шаг. Другими словами, с арифметическими устройствами было все решено фактически в 1954–55 гг. Мы сделали нечто не улучшаемое. Это четко поставленная проблема, решенная до конца.

Архитектура – это одна сторона, технология – другая, и обе стороны крайне важны. Надо сказать, что архитектура постоянно была ведомой, технология развивалась и ставила новые задачи перед архитектурой. Не надо думать, что это чисто технологическая область, потому что архитектура машин – это фундаментальная область, как то, что я вам рассказывал про арифметику. С технологией это никак не связано, это фундаментальное решение хорошо поставленной проблемы, но, тем не менее, каждый новый уровень технологии требует дополнительных решений каких-то новых фундаментальных вопросов. Развитие архитектуры в значительной степени было всегда связано с использованием параллелизма, который ускоряет вычисления.

Следующий шаг в развитии архитектуры был сделан в России. Других таких коллективов, эквивалентных нашему коллективу, в России не было.

Мы шли впереди. Этот шаг был навязан технологией, так сказать, плохой ее стороной. В создании БЭСМ я не принимал участия, так как пришел студентом. Первая машина (М-40), в которой я участвовал, была ламповая. Она использовалась на полигонах в противоракетной обороне. При помощи этой машины было проведено первое успешное испытание противоракетной системы. Нам первым удалось сбивать ракеты, задолго до западников.

Чтобы вы представляли надежность этой ламповой машины, я расскажу, что однажды произошло. С Волги, из Капустина Яра запускали ракеты, а мы на полигоне на Балхаше сбивали эти ракеты. Представляете, что это гигантский комплекс, и целый день там проверяется готовность: 10-часовая, 5-часовая, 10-минутная готовность. Все прошло блестяще, все было готово. ЦУ дает пуск. Баллистическая ракета запущена. И вдруг, сразу же после запуска этой ракеты у нас взорвалась одна лампа в машине. Память тогда была сделана на ферритовых торах, а драйверы этой памяти на лампах ГУ. Взорвалась лампа. Ракета летит до нас 12 минут, т. е. у нас было 12 минут времени. У нас были подогретые лампы. Что тогда было хорошо – так это простейшая диагностика, сразу видно, где сломалось. Мы эту лампу выдернули, поставили новую лампу, проверили все. Не прошло 12 минут – у нас все заработало. Успешно сбили ракету, вывели комплект печати, и тут взорвалась вторая лампа. Если бы мы не вывели печать, весь пуск пропал бы, так как не было бы никакой информации.

В таких условиях мы работали: сильно страдали от ненадежности машин. Потому на следующей машине, это была полупроводниковая 5Э92б, которая выполняла умножение и деление, разработанное мной, мы уже приняли меры, чтобы этого больше не повторялось, потому что это очень дорого стоило. Мы сделали новую машину и тем самым решили вторую архитектурную задачу. Мы сделали машину, которая при одиночном сбое восстанавливается стопроцентно, за счет резервирования. 5Э92б – это комплекс, состоящий из большой и малой машин, которые работали на общей системе команд и с общей памятью. Заметьте, что это был 1964 год. Многопроцессорность уже существовала. Но все не так просто. У нас был корректирующий код, т. е. каждое слово имело дополнительные разряды, которые обнаруживают ошибку. И если ошибка была перемежающаяся, мы перезапускали задачу по несколько раз. Если состояние восстанавливалось, то выполнение боевой задачи продолжалось. Машины тогда представляли собой много шкафов. Чтобы испытывать эту систему, мы все строго просчитывали. Любой одиночный сбой восстанавливался со 100%-й вероятностью. Множественные ошибки, конечно, тоже восстанавливались, но не всегда.

В дальнейшем мы развили эти методы в следующих поколениях машин, которые уже были многопроцессорными. И уже восстанавливали не только перемежающиеся сбои. В случае полного отказа одного из процессоров сокращенное число процессоров восстанавливало сбой с помощью ПО, и счет продолжался. Много проблем было решено, и это сильно повысило надежность.

Эта функциональная надежность на машинах следующего поколении нам здорово пригодилась. Потому что следующее поколение мы делали на ECL-схемах средней интеграции, – «Эльбрус-1» и «Эльбрус-2». Зеленоград, к сожалению, сделал очень ненадежные схемы, так что наработка на отказ одного процессора составляла 300 часов. Здесь наша техника избыточности очень сильно пригодилась. За много лет, что эти машины простояли на противоракетном дежурстве, не было ни одного случая, чтобы число процессоров десятипроцессорной машины сокращалось меньше восьми. Мы так и договорились, что если остается восемь процессоров, то это не считается за отказ системы.

Сейчас машины надежные, но раньше, когда машина ломалась, то, до создания системы избыточности, имел место постоянный спор между программистами и инженерами. Если ответ был неправильный, инженер говорил, что это в программе ошибка, а программист говорил, что это машина сломалась. При наличии избыточности и аппаратного контроля все очень просто: если срабатывал аппаратный контроль, загоралась лампочка, и все понимали, что бегать будут инженеры, а программисты отдыхать. Если лампочка не загоралась, а результат был неправильный, то инженеры были спокойны: ошибка в программе. Это имело много положительных результатов.

Все достижения, которые имелись в предыдущем поколении машин, мы, конечно, переносили на следующие поколения. И быстродействующая арифметика, и функциональная надежность использовались в многопроцессорных комплексах, но особенно революционным сдвигами был отмечен первый «Эльбрус». Мы начали проектировать его первую модель в 1972 г., а первый тест через операционную систему первой машины прошел в 1978 г. В декабре 1978 г. мы распечатали то, что сейчас называется «Hello World!». А мы просто распечатали единичку, первая программа состояла из печати единички, но она прошла через всю операционную систему и через всю аппаратуру – и распечатала единичку.

В первом «Эльбрусе» тоже было много шкафов, потому что интеграция здесь была десять вентилей на кристалл, ТТЛ-вские схемы. В этой машине было два крупных фундаментальных нововведения. Первое – очень важное. Вы все хорошо знаете, что все современные машины, наверное, сейчас

можно сказать, к сожалению, имеют такую архитектуру, при которой программа, выполняемая на вычислительной машине, представляет собой последовательность операций, команд. Каждая команда состоит из операции и ее аргументов.

Для первых машин большего и не было нужно. Первая БЭСМ, как вы знаете, работала таким образом: она сначала считывала аргументы из памяти, которая по сравнению с арифметикой тогда была очень быстрой. Потом работала арифметика. На операцию требовалось несколько тактов, и арифметика выполнялась долго, а память ничего не делала. Затем результат записывался в память. Считывалась следующая команда, анализировалась, и т. д. Все выполнялось последовательно. В дальнейшем процессоры все время ускорялись, а память отставала. И разрыв между памятью и процессором постоянно рос, на уровне М-20 это уже было заметно, потому что когда обращаются к памяти, а процессор в это время простаивает, то это очень заметно. И тогда уже наложили считывание следующей команды и считывание данных на работу предыдущей команды.

Это был уже первый параллелизм, но и первое неудобство. Если результат данной операции используется следующей операцией, то нельзя выбирать заранее аргумент, и программист должен был позаботиться, чтобы этого не было. Машина точно выполняла указания программистов, и программист был ответствен за результаты, если машина выполняла команды так, как он написал, т. е. последовательно. Тогда гарантировался именно тот результат, который он уже ждал.

Уже в 5Э92б было использовано множество арифметических устройств. Они появились не столько потому, что мы хотели параллельно (в современном смысле) считать, сколько потому, что полупроводниковые дискретные интегральные компоненты тогда уже позволили использовать гораздо больше оборудования. В первой БЭСМ вся арифметика делалась на одном арифметическом устройстве. Сложение было последовательное. А теперь у нас было достаточно оборудования, чтобы умножение делать пирамидой. И мы поняли, что столбиком умножать – нехорошо. Это привело к тому, что умножитель стал специализированным устройством. Были еще арифметические устройства для умножения, деления, сдвигов. Имея много арифметических устройств, выполнять команды последовательно одну за другой не имело смысла. Это наводило на мысль, что здесь есть резервы.

Мы стали думать, нельзя ли последовательную программу выполнять параллельно. Начало этой разработки – 1972 год! Естественным желанием было распараллелить последовательную работу на много устройств, но это

оказалось совсем не просто. Если следующая операция использует результат предыдущей, то перестановка операций и распараллеливание невозможны. Например, есть такая программа: операция сложения, в которой первый аргумент считывается из памяти в первый регистр, второй – считывается из памяти во второй регистр, они складываются и результат записывается в память. Далее операция умножения считывает в третий регистр и в четвертый, умножает и записывает в память. Может так оказаться, что числа под первую операцию еще не готовы, еще не считались из памяти, а под вторую операцию уже готовы. Хочется переставить. Но представляете, что это такое – во время выполнения анализировать, можно ли переставлять. Это для машины гигантская работа, потому что все это надо делать в каждом такте.

Более того, оказалось что, кроме выяснения информационных зависимостей, нужно переименовывать регистры. Я могу привести такой пример. Программист написал такую программу: первый регистр сложить со вторым. Результат записать в память. Для программиста первый регистр уже освободился, и он дальше пишет команду умножения, опять считывая что-то в первый регистр. Умножить с чем-то и записать в память. Эти два фрагмента по смыслу независимы, но программист использовал один и тот же регистр, потому что он-то считает, что команды выполняются последовательно. Спасти ситуацию может условие, что второй фрагмент надо считывать не в первый регистр. Тогда все это независимо и можно переставлять. То есть на ходу надо было обнаруживать зависимости и переименовывать регистры.

Мы это все сделали. Мы разработали машину, которая сейчас называется суперскаляр, она заработала в 1978 г. Это был классический суперскаляр. Мы переставляли операции, и выполняли в пике две команды за такт. Сейчас выполняют четыре команды за такт. Но и сейчас только четыре, а если больше – начинает плыть тактовая частота. Первый суперскаляр у нас заработал в 1978 г., а первый суперскаляр за рубежом – «Пентиум» – заработал в 1994–95 гг. Вот насколько мы опередили. Это было решение фундаментального вопроса.

Четко поставленная задача – выполнять последовательную программу как можно более параллельно – и сейчас никаких других методов ускорения вычислений на Западе не используется. Эта идея до сих пор базовая, она единственная. Даже на микроуровне мы использовали технику, которую только сейчас во всех машинах используют. У нас система команд была безадресная, основанная на стеке. Такую систему команд очень сложно распараллеливать. Потому что стек – это существенно последовательное

понимание. Прежде чем применять суперскалярную технику, мы на лету перекодировывали стековую нотацию в трехадресную регистровую. И уж после этого применяли все суперскалярные методы.

Сейчас во всех машинах в «Интеле», в АМД тоже так делается. Несколько по другим причинам, конечно. Берется трехадресная операция, перекодируется на лету в микрокод, в микрокоманды, а затем применяется суперскаляр. Даже в этом плане мы опередили Запад технологически, на 14 лет опередили. Конечно, параметры этой работы другие. Сейчас сложнейшие системы команд, которые тогда невозможно было бы распараллелить. Если вы посмотрите на x86, то ее система команд очень сложная, и ее надо было распараллелить, как мы в свое время распараллелили «Эльбрус». Мы сами разработали систему команд. Это еще один пример, где мы существенно опередили.

Что касается суперскаляра, то не очень точно было бы говорить, что мы самые первые его сделали. Потому что когда мы рассказывали о наших работах, в том числе на IBM, профессор Хопкинс заявил, что они первые сделали суперскаляр, чуть-чуть раньше нас. Я ответил, что следил за литературой, и спросил, были ли публикации? Он сказал, что публикаций не было, коммерческого продукта тоже не было, так как это была закрытая работа. Тогда я пошутил, что у нас много закрытых работ было. Одно дело – разработать научную машину, экспериментальную, лабораторную, а другое дело сделать коммерческую машину. Это значительно более серьезный вопрос. Это была первая опережающая технология «Эльбруса-1».

Вторым нововведением было защищенное программирование. Развитие архитектуры машины, языка машины, его совершенствование, состоит из двух частей: совершенствование семантики машинного языка и совершенствование синтаксиса. Что такое семантика машинного языка? Я бы сказал, что это набор машинных данных, которыми оперирует язык машины, и соответствующий набор операций, т. е. базовый набор типов данных. Это четыре базовых типа: булевы, целые и вещественные и, к сожалению, пойнтеры (указатели), при этом пойнтеры бывают двух сортов – data-pointer и functional-pointer – указатель на данные и указатель на процедуру соответственно.

Если вы посмотрите на первые три типа: булевы, целые и вещественные, то на них больших нареканий сейчас нет. Можно придираться к тому, как там сделаны прерывания и т. п., но эти три типа в довольно хорошем состоянии. На наиболее сложную вещественную арифметику есть стандарт.

Все три типа имеют хорошее теоретическое обоснование и фундаментальное понимание. Булевы – это просто минимальная информация. Це-

лые – это потенциальная бесконечность. Вещественные – это потенциальная непрерывность. В конечном варианте, все они представляются с огрехами, разумеется: у целых не хватает разрядов, и поэтому появляется переполнение, у вещественных – потеря значимости и т. д. Это я бы отнес к разряду придирок. А вот с пойнтерами дело обстоит хуже. В принципе, по алгоритмическим понятиям, по языкам, пойнтеры должны указывать на объекты. А в настоящее время в машине они просто указывают на место в памяти. И от этого места памяти далее можно считывать все что угодно, хоть всю память. Это приводит к невероятно плохим последствиям.

Люди не осмыслили до конца все отрицательные последствия этого, потому что никто по-другому и не работал. Конечно, в интерпретационных языках, в отладочных режимах, все так и делается, но все уже привыкли, что за хорошую отладку надо платить потерей эффективности, а это не совсем верно. Можно сделать машины, которые будут очень быстро работать и которые будут великолепно защищены с полной семантикой, в том числе и пойнтеров. «Эльбрус-1» был такой машиной. Мы ввели в «Эльбрусе» так называемые теги. Никто до нас, да и после нас, к сожалению, не использовал теги, чтобы обеспечить защищенное программирование.

Когда Джек Деннис приезжал в Россию, мы с ним обсуждали историю, и я ему говорил, что теги впервые использовали в двух местах. На фирме «Барроуз» и в Лондонском университете. На фирме «Барроуз» это сделал Роберт Бартон. В Лондонском университете – Айлиф. Курьез заключается в том, что коммерческими машинами были B5000 и B5500. А в Лондонском университете это даже не было новой машиной. Они просто взяли ICL-вскую машину и перепрограммировали микрокоды. В лондонской машине теги действительно использовались для защищенного программирования – CodeWords и т. д, но это была не реальная машина, а игрушка.

В реализации Роберта Бартона теги для защиты не использовались, т. е. теги в непривилегированном режиме могли переделываться пользовательскими программами. Я спрашивал в «Барроузе», вот у нас в Советском Союзе люди компиляторы любят делать, если они сделают компилятор, а программа будет ошибочная, как быть? А они отшучивались, дескать, пускай приходят ночью и отлаживают. Тогда пакетный режим был основным. До 1978 г. поисковых работ в этой области в мире было много, почти в каждом университете. Мы были первые, кто реализовал типовую защиту (type-safety) в коммерческом варианте. И она заработала в 1978 г. Понятно, что все языки обладают type-safety, только эффективность целевой программы заставляет пренебрегать этим при реализации. Если посмотреть на C и C++, то по своему существу, по своему базовому понятию это type-

safety. А в реализации type-safety нет. Но есть пойнтеры, которые позволяют делать, что хочешь. Ради эффективности исполнения и дьяволу душу продают. За всю мою жизнь я понял, что неэффективные системы никогда нельзя предлагать. Все, что ты разрабатываешь, должно быть эффективным. Если что-то неэффективное, не надо даже и предлагать.

Мы сделали систему, которая поддерживает type-safety и проверяет указатели. Это невозможно сделать эффективно в ПО из-за динамических проверок. Динамически это можно сделать только при помощи аппаратуры, и для этого аппаратура должна знать, где пойнтеры, а где не пойнтеры. В «Эльбрусе» мы это и сделали. Мы отметили все пойнтеры тегами и тегировали пользовательские программы. А теги не были доступны простым пользовательским программам. Это была прерогатива самой аппаратуры.

К примеру, вы могли взять команду и сказать: я хочу проиндексировать. Тогда она смотрела, если индексируешь, то первый аргумент – дескриптор и должны быть соответствующие теги. Если дали не теги, а целое число, то возникнет прерывание. Второе число должно быть целое, тогда все правильно. Мы проверяли, что каждый дескриптор описывал объект, вот и получается обратная связь. Сама машинная программа работает как языковая высокоуровневая процедура, проверяя правильность поданных аргументов,

Архитектура с тегами позволяла очень точно и совершенно гарантированно обеспечивать правильную работу, правильную семантику пойнтеров. Защита и должна быть железобетонная, там щелей не должно быть, иначе это не защита. То есть, если вы создаете массив, то, вы, скажем, обращаетесь в операционную систему, и она вам выделяет площадь и, одновременно, дает дескриптор на массив так, что вы уже хозяин этого массива. Если вы его сами никому не отдадите, то в него никто обратиться не сможет. Ну и наоборот, если вам никто не дал никакого массива, то вы ни до чего доступа иметь не будете.

Представьте себе машину с регистрами. Регистры, как правило, это как корень дерева доступов любой программы. Регистры мы можем также себе представить как некоторый массив с некоторым дескриптором. Мы его назвали контекстным дескриптором. У работающей программы есть один дескриптор, причем реально его нет, но мы можем его себе представить, чтобы все рассматривать более однородно.

Есть контекстный дескриптор, и значит, все программы могут обращаться только в данный массив, а на самом деле, это массив регистров. Если в этом массиве нет ни одного другого дескриптора, то вы больше никуда не можете обратиться – аппаратура вам просто не разрешит. При этом понятия несанкционированного доступа нет, аппаратура вам укажет на ошиб-

ку. Если регистров у нас, скажем, тридцать два, а вы обратились с индексом пятьдесят шесть, то она просто скажет, что у вас ошибка, что этот массив короче и вы ошиблись.

Если в этих регистрах есть один дескриптор, то вы имеете доступ в соответствующий массив, т. е. кусок памяти, а если в этом массиве есть еще один дескриптор, то есть доступ и далее. Таким образом, от контекстного дескриптора путем волнового алгоритма формируется пространство, до которого вы имеете доступ из работающей в данный момент программы. Она имеет доступ только к тому информационному пространству, которое достижимо от этого дескриптора, но это и есть семантически правильное пространство. Потому что, когда запускается какая-то процедура, то ей передается, грубо говоря, массив параметров, и больше она ничего не имеет, ей доступен только этот массив параметров и, может быть, глобалы.

Это очень гибкая система. Что тебе дано, то ты и имеешь, а что тебе не дано – ты не имеешь. Например, из массива можно взять подмассив: скажем, из ста чисел выделить три подряд числа, образовать для них новый дескриптор и отдать его в новую процедуру или записать куда-то в глобалы, ибо область глобалов – это тоже дескриптор. Фактически каждая процедура зажата в контекстные рамки, она не может выйти из того пространства, в котором она работает.

И единственное, что можно сделать – это забить дескриптор. Тогда ты уже перестаешь иметь доступ к тому, что у тебя было раньше, но это, наверное, не все будут делать, да это и не очень нужно. Таким образом, теги позволяют сохранять статус-кво, т. е. контекст процедуры невозможно нарушить. Я вам сейчас рассказал только про data-pointer, указатель на данные. Но кроме него в таких системах существует то, что называется function-pointer (указатель на процедуру или функцию), т. е. указатель не на работающую процедуру, а на процедуру, которую можно запустить.

В принципе, указатель на процедуру – это пара: во-первых, код, который можно запустить, если мы пойдем через этот пойнтер, и, во-вторых, это тот самый глобальный контекст. Когда процедура запускается, у нее всегда есть какой-то глобальный контекст, как в языках с вложенной структурой. То есть если существует одна процедура, а внутри нее описана другая процедура, то когда вы эту вторую процедуру запускаете, то все локальные данные первой процедуры доступны. И дескриптор, описывающий локальные данные динамически охватывающей процедуры, составляет часть функционального дескриптора. Эти функциональные дескрипторы есть тегированная информация – они могут быть в моем контексте.

Когда работает процедура, функциональный дескриптор смотрит на контекст потенциально запускаемой процедуры. Команд, которые могут пойти по этому контексту, нет, так как этот дескриптор может быть использован только командой запуска процедуры, а остальные команды просто недопустимы. Ошибкой будет, если вы попытаетесь сделать что-то другое, например, если вы захотите изменить этот дескриптор. В таком случае все теги обнулятся и будут просто числа – весь фокус пропадет.

Функциональный дескриптор работает следующим образом: вот работает процедура со своим контекстом, вот вы взяли какой-то функциональный дескриптор, в нем другой контекст, который пока недоступен, вы берете этот дескриптор и запускаете процедуру. В это время текущий контекстный дескриптор погружается в стек и становится недоступным, а доступным становится этот новый контекстный дескриптор. Так как зона параметров – это перекрытие в стеке, то зона параметров тоже оказывается доступной. Таким образом, работает новая процедура, а старая процедура находится в стеке и ждет возврата.

Если вы хотите пополнить свой арсенал, например обратиться, чтобы сгенерировать новый массив. В вашем глобальном контексте есть метка операционной системы, обычный стандартный функциональный дескриптор, вы запускаете процедуру, и, как я уже сказал, когда новая процедура запускается, открывается ее контекст. У операционной системы гигантский контекст, в него входит вся виртуальная память и есть таблицы, по которым она видит, что роздано, а что не роздано.

Тогда в соответствии с запросом (запрос – это параметры обращения к операционной системе) операционная система видит, что эта процедура попросила пять слов. Она находит пустые пять слов, формирует дескриптор и возвращает его. То есть раньше процедура работала и этого массива не было, а теперь, после возврата, в ее контексте появился новый дескриптор, т. е. все описывается этими простыми фундаментальными алгоритмами, и дальше нет ничего нового. Это все было просто взято из языков программирования и реализовано в аппаратуре.

Но смотрите, какой эффект! Во-первых, при введении такой технологии, совершенно фантастически упрощается отладка программ. Наши пользователи, люди которые программировали противоракетную оборону, и не только они, на опыте убедились, что, в общем, отладка проходит на порядок быстрее. Я объясню почему.

Как сейчас работают обычные простые программисты на обычной машине? Пускают задачу, она может прерваться, но, как правило, не прерывается, даже если в программе есть ошибка, и доходит до конца, затем про-

граммист смотрит результаты, а они не совпадают с тем, что он ожидал, и начинается мучительный процесс нахождения места, где возникла эта ошибка. В нашей системе, благодаря мощному динамическому контролю тегов, машина прерывается в точке ошибки, т. е. прямо на месте ошибки, и если произошла ошибка, вся последовательность вызовов, которая привела к этой ошибке на этой процедуре, находится в стеке процедур.

В «Эльбрусе» были реализованы все диагностические процедуры, но почти никто ими не пользовался. Как только происходил сбой, осуществлялась проверка стека и находилась ошибка. Когда мы сдавали «Эльбрус-2», Леонид Райков, мощный человек, руководитель программистов НИЦЭВТа, участвовал в приемных испытаниях. В «Эльбрусе» аппаратура была советская, операционная система была советская, компиляторы были советские. А в это время была сделана ЕС-1066 на абсолютно той же интегральной базе, но скопированная с IBM-370/3033, скопированная с точностью до микрокодов, т. е. логика – это абсолютно 3033, операционная система IBM, компиляторы IBM, и «Эльбрус-2» сравнивали с ЕС-1066. То есть мы конкурировали не с НИЦЭВТом, мы конкурировали с IBM.

Нельзя сказать, что у нас программы были хорошо отлажены… Мы сдавали систему многопроцессорную и многопультовую, и когда во время испытаний возникала ошибка, то распечатывалась память. Это был просто дамп памяти, он был типизированный, так как благодаря тегам было видно: вот здесь тег, вот здесь ошибка, здесь данные, а здесь дескриптор. Наши ребята садились, смотрели, часа два анализировали, а потом шли исправлять ошибку. Леонид Райков был просто поражен, потому что в обычной системе на поиски ошибки часто уходит много месяцев.

Представьте себе, что есть гигантская программная система, и давайте представим, что в ней нет ошибок, что она великолепно отлажена, хотя это и противоречит обычным высказываниям о том, что в любой программе есть хотя бы одна ошибка. Но представим, что в ней нет ошибок, и вы подключаете к ней новую процедуру, про которую вообще ничего неизвестно.

Она может работать плохо, вы, конечно, можете остановиться после того, как она выполнилась, посмотреть ее результат и, если он неправильный, – принять меры. Но самое плохое, что она может даже выдавать правильный результат, но ее побочный эффект что-то портит. Поэтому вам нужно быть готовым отлаживать уже отлаженное, потому что если она память испортила, то какие-то другие процедуры перестанут работать, а они уже отлажены, и программный комплекс большой. И вот, кто-то подключает эту процедуру, а остальных авторов просто нет, например некоторые

давно уволились с работы, и непонятно что же делать. Так вот – у нас этого просто не может быть.

Вопрос Б.Г. Михайленко (ИВМиМГ СО РАН): Переход с линии БЭСМ-6, на линию IBM – это была ошибка?

Ответ: А это немного не то, о чем говорю я, это же хозяйственный вопрос. Я бы не хотел высказываться категорично. С точки зрения развития архитектуры это, безусловно, ошибка, но хозяйственно, может, это и правильно. Единственное, что я знаю, все думали, что хлынет матобеспечение вообще, а оно, матобеспечение, не хлынуло. Воровали же программное обеспечение, и версии операционной системы там не совпадали с версиями языков, и было много проблем, потому что все было несовместимо.

А вот с точки зрения существа дела, я помню, что до того как ЕС ЭВМ начали делать, была масса коллективов, которые двигались мощно вперед, хотя и был большой перебор разных архитектур, но этот силовой переход на IBM все остановил, всех загнали в ЕС.

К нам в ИТМиВТ, наверное, пару месяцев приходили министры и замы министров. Сначала они хотели, чтобы Сергей Алексеевич возглавил НИЦЭВТ, потом, чтобы хотя бы присоединился. Сергей Алексеевич стоял насмерть: мы будем разрабатывать свое. Если бы Сергей Алексеевич тогда не устоял, не было бы всех этих результатов и не было бы того, чем мы сейчас занимаемся, в том числе и вы.

Теперь я продолжу, предположим, что вы подключаете процедуру с ошибками. Несмотря на то, что она с ошибками, она все равно ограничена контекстом, она ничего не может испортить вне контекста, она может только выдать неправильный результат. Я могу поставить останов на выходах, а неверный выход она не может сделать, потому что все переходы динамические, они тоже тегированы, я могу остановиться в конце. Ну, теоретически, она может зациклиться, и это трудно доказать, что она не может зациклиться, но практически зацикливание легко обнаружить. Я могу остановиться перед выходом, посмотреть ее побочный эффект, и если он меня удовлетворяет, то я могу забыть про эту процедуру. А все остальное будет все время блестяще работать, и это сильно облегчает интеграцию крупных программных комплексов.

По крайней мере, две причины, которые привели к тому, что отладка на «Эльбрусе» на порядок быстрее. Я помню, как наша молодежь, пока мы не стали работать с западными компаниями, нас атаковала – все работают без тегов, а вы придумали какую-то новизну. Мы говорили, что это облегчает

отладку, но все смотрели косо, а когда стали работать на западных машинах, они пришли и стали говорить, что на них и работать невозможно после «Эльбруса».

Мы поставили машину «Эльбрус» в Арзамас. Сначала у них были ЕС-совместимые машины, потом им поставили «Эльбрус-1». Они стали работать на Фортране и обнаружили, что даже на Фортране программы лучше отлаживаются. Много ошибок находили даже на отлаженной программе.

Вы знаете, какой опыт мы получили. Мы много переносили программ на «Эльбрус» и накопили большой опыт. Ни одна хорошо отлаженная программа, перенесенная на «Эльбрус», не проходила так, чтобы мы не находили в ней ошибок. Вы знаете, есть тесты SPEC, которые на всех машинах исполнялись, которые через все компиляторы прошли. Мы в SPEC95 нашли 30 ошибок и послали их в комитет по тестам SPEC. Это гигантское улучшение производительности программистов. Я думаю это то, что нужно Microsoft, у них сейчас творческий кризис, они сейчас не могут отлаживать гигантские программы.

Расскажу такой эпизод, который имел место, когда мы сдавали «Эльбрус-2» Госкомиссии. В Госкомиссии работали представители всех организаций. Мы попросили предъявить все задачи. Каждая организация предъявила по одной задаче. Мы их пропустили, подготовились, показали. А потом одна организация говорит, что есть еще одна задача. А мы хотели с гордостью показать виртуальную память. Поэтому мы хотели взять задачу, пустить ее в первозданном виде, а потом немного ее виртуализировать, чтобы скорость на ней показать.

Представитель захотел, чтобы задачу пропустили у него на глазах. Мы спросили, нет ли там ошибок? Он ответил, что это маленькая программа, что он ее на всех машинах пропустил, что на ней он проверяет скорость на всех машинах. Это был кто-то из ИПМ, по-моему. Мы согласились с условием, что пропустим ее до первой ошибки. Если будет хоть одна ошибка, мы ее выкинем. Мы знали, что там должны быть ошибки. Пустили ее, дошли до ошибки, она у нас на барабане сохранилась или на диске. На следующее утро мы выдали как отлаженный вариант исходной программы, так и переделанный вариант под большую скорость. Вот до чего мы были уверены, что отладка невероятно сильно ускоряется.

Это технология, которая была разработана на первом «Эльбрусе». На втором, на третьем она улучшалась. Сейчас эта разработка на таком уровне, как ни удивительно, когда теговская архитектура не замедляет вычисления, а ускоряет их. Потому что в аппаратуре есть масса мест, незаконных с точ-

ки зрения защищенности, которые все равно аппаратчики вынуждены отлаживать, чтобы машина работала быстрей. При защищенном режиме есть много вариантов, которые незаконны, и их просто не нужно отлаживать. Я уверен, что защищенные системы, грамотно спроектированные, процентов на 20 ускоряют производительность.

Эта технология была разработана тоже в 1978 г. До настоящего момента на Западе ее нет. До начала 80-х годов на Западе было проведено много экспериментов с type-safety. В начале 80-х годов «Интелом» была разработана машина архитектуры 432, где была реализована технология type-safety, но очень неаккуратно, неправильно. Они побоялись, как и все, ввести теги, поэтому сделали традиционно.

Барроуз тоже с этого начинал, а потом написал, что это ошибка, делать без тегов. Это похоже на то, что все дескрипторы собираются в отдельный сегмент. И это то, что type-safety аппаратура знает, что если в этом сегменте, значит, это дескриптор, но это дико ухудшает программирование, усложняет машину, и программирование становится не эффективным. В этой архитектуре было много других ошибок.

Я совершенно объективен, потому что когда появилась 432 машина, НИЦЭВТ купил машину «Интеллек». Начали прорабатывать, а в это время наверху в силе был Владимир Сергеевич Семенихин. Он очень хорошо относился к «Эльбрусам», и он меня позвал. Он сказал, Борис, что же они там берут какую-то интеловскую машину, когда вы тут «Эльбрус» разрабатываете?

Я познакомился с этой машиной и понял, что это неправильный подход. Но в то время был выпущен документ ЦК и Совмина, в котором они обратились к научной общественности о целесообразности копирования 432 машины. Была создана комиссия, я был ее членом. Все думали, что я буду защитником такого копирования и поддержу его. Я написал большой доклад против. А буквально через несколько месяцев уже весь мир понял, что никакой эффективности не было в той архитектуре. К сожалению, после этой работы все решили, что если уж лидеры не могут, то никто не сможет это сделать, и все бросили работы в этой области.

Тогда, в 1982 г., «Эльбрус-1» работал с положительным результатом. Это еще один пример, где мы значительно опережали Запад. То была выдающаяся связка двух технологий: суперскаляр и защищенное программирование. Суперскаляр – это, фактически, в области синтаксиса, в области скорости. А защищенное программирование, type-safety, я продолжаю быть последовательным – это не улучшаемое улучшение семантики компьютеров.

Уже тогда в 1985 г. мы поняли, что суперскаляр – это хорошо. Когда мы сдавали «Эльбрус-2» в 1985 г., НИЦЭВТ сдавал 66-ю машину. Между нами было соревнование. Технология одна и та же. Машина НИЦЭВТа была копией IBM, а «Эльбрус» был отечественной машиной. Они писали тесты для нас. Мы писали тесты для них. Мы их обыграли в два раза по скорости, потому что «Эльбрус-2» – суперскалярная машина.

Мы тогда обнаружили, что суперскаляр очень сложно аппаратно реализовать, и, честно говоря, все машины становятся все сложней и сложней. Развитие технологий делает их сложней, но надо, чтобы эта сложность разумно использовалась. А суперскаляр, по нашему понятию, и это понятие пришло к нам в 1985 г., слишком сложный и, самое главное, что он не дает возможности дальнейшего наращивания скорости. Почему? Потому что суперскаляр на ходу, прямо во время исполнения, должен распараллеливать исполнение последовательных команд. Он должен определять информационные зависимости, он должен знать, что переставлять, он должен выполнять переименование регистров. И это сложно. Мировая практика показала, что до ширины четырех команд это можно сделать, а дальше уже сам процесс преобразования из последовательности в параллельность не дает двигаться вперед. Нет возможности обработать за такт больше четырех команд – это предел. Это мы поняли в 1985 г., когда на Западе еще даже не начинали суперскаляр. Он только в 1995 г. появился.

Мы решили, что всю сложность преобразований должны делать компилятор и аппаратура вместе, т. е. параллелизм должен быть явный. Тогда мы начали работу над «Эльбрусом-3». Мы сделали машину с очень широким командным словом и никаких динамических механизмов, которые были в суперскаляре. Я не хочу сказать, что это была не улучшаемая архитектура, она улучшаемая, безусловно, но это был серьезный шаг по сравнению с суперскалярами.

Если вы посмотрите на любой алгоритм, то увидите, что это параллельное образование, там много видимого параллелизма. Но есть и невидимый параллелизм, через память. И что происходит? Пока аппаратура была не параллельная, а еще последовательная, компиляторы делали эффективную работу. Они точно планировали работу машины. Когда машины были простые, они просто последовательно выполняли операции, делали перестановку операций.

На М-40 мы считали быстродействие следующим образом: если операции одинаковой длины, то зависимость исходит от числа операций, если операции разной длины, а перестановка ни к чему не приводит, то просто учитывается статистика, сколько умножений, сколько сложений. В современных машинах

с подкачкой, когда память очень медленная, вам надо успевать подкачивать. И чтобы посчитать скорость, нужны довольно сильные модели.

Сейчас последовательная система команд – это просто тормоз. Потому что алгоритмы становятся все более параллельными, аппаратура высоко параллельная, и между ними эта последовательная прослойка – последовательная система команд. Компиляторы начинают упаковывать параллельную программу в эту последовательную систему команд, а это уже планирование вычислений – непростая работа. Причем планирование вычислений на машину, которую компиляторы хорошо не знают внутри. Потому что аппаратура динамически по-разному работает, и это совершенно неоднозначная работа.

Компиляторщики какие только эвристики ни придумывают, но что-то это все не очень надежно работает. *Inline*, *prefetch*, *software pipeline*, *peeling*, чего только не придумывают. Это все планирование той работы, про которую они не знают, как она будет выполняться в run-time. Компиляторы делают очень сложную работу, которая с точки зрения скорости не очень эффективна. Дальше аппаратуре опять предстоит сложная работа, так как последовательность команд она должна дешифровать обратно в параллелизм.

Это нужно было изменить: параллелизм, минуя последовательную командную нотацию, передать прямо аппаратуре. Мы сделали такую машину. Это «Эльбрус-3» с архитектурой широкого командного слова. В «Эльбрусе-3» пиковая ширина, т. е. за такт можем выполнять 22 команды. Это очень большая ширина. Разработанные нами компиляторы раскладывают параллельный граф в широкую команду. Результаты получились очень хорошие.

Но «Эльбрус-3» – это многострадальная машина, потому что мы ее изготовили в пресловутом 1991 году. К тому времени финансирование прекратилось, кроме того стали доступны западные машины. Представляете, что такое «Эльбрус-3»? Это очень допотопная технология по сравнению с западными технологиями того времени. Большой шкаф с водяным охлаждением, в шкафу «Эльбруса-3» приблизительно 15 млн транзисторов. В то время мы уже начали работать с Sun Microsystems, и к нам приезжал Скотт МакНили, президент Sun. Он привез чип – первый суперспарк. У меня фотография есть, она весь мир обошла. Мы с ним стоим, сзади «Эльбрус-3», а у него в руке этот чип. Там в шкафу – 15 млн транзисторов, а у него в одном кристалле – 6 млн транзисторов.

«Эльбрус-3» нужно было эксплуатировать с технологическим подэтажом для систем водяного охлаждения. Сейчас такие этажи утилизировали. А западные машины, несмотря на наши архитектурные достижения, побеждали чистой грубой силой, их технологии все перебивали. И «Эльбрус-3»

просто не было смысла продолжать отлаживать, несмотря на то, что при моделировании, насколько я помню на LinPack'е, «Эльбрус-3» был в два раза быстрее Cray YMP, самой быстрой американской машины в то время. Так «Эльбрус-3» и не был завершен, хотя это была постсуперскалярная архитектура – широкое командное слово.

Потом мы пытались найти инвесторов, но не нашли. Сейчас команда завершила разработку «Эльбруса-3» уже в виде кристалла. Машина показывает просто фантастическую скорость. На задачах типа SPEC и т. п. она в полтора–два раза работает быстрее, чем другая машина на той же тактовой частоте. Особенно на задачах, которые все более и более важными становятся: компьютерная графика, цикловые задачи, где параллелизма больше.

Было еще такое испытание. Все заказчики дали задачи. Эти задачи были исполнены на 300 МГц «Эльбрусе-3» и на современной машине с тактовой частотой 1,5 ГГц. «Эльбрус-3» по сравнению с этой машиной оказался в 1,4 раза быстрее во всех испытаниях. Потому что эта архитектура действительно оказалась выдающейся. Я думаю, что эта архитектура, конечно, не предел. Здесь можно и нужно усовершенствовать. Минус этой архитектуры в том, что она еще слишком статичная.

Нам надо передать информацию, что можно выполнять в параллель, а мы передаем информацию, что надо выполнять параллельно. Следовало дать возможность аппаратуре активнее вмешаться в процесс. Так что есть еще над чем работать.

Чтобы эту архитектуру внедрить в производство в середине 90-х годов, нам нужно было решить сложнейшую задачу. Потому что эта архитектура, явно не совместима ни с какими существующими архитектурами. А уже было гигантское количество пользовательских программ, двоичных программ. Это уже в то время, сейчас тем более. И предлагать архитектуру, которая не совместима с этим багажом, было просто сумасшествием, нас просто никто бы слушать не стал.

Поэтому нужно было решить проблему обеспечения на новой архитектуре совместимости со старой архитектурой. И мы ее решили – это технология двоичной компиляции. Мы разработали технологию, которая скрытым от пользователя способом исполняет интеловские программы. Для пользователя «Эльбрус-3» – это «Интел». Пользователь приносит интеловские программы, пускает, и они идут. Машина сама на ходу транслирует их и выполняет в эльбрусовском варианте. 100 %-я совместимость, включая операционную систему. На испытаниях принесли незнакомую операционную систему, раскрутили – работает. Это очень мощная и достаточно сложная система.

Двоичная компиляция как идеология известна уже давно. С ее помощью можно восстановить ассемблерный текст, можно коды, написанные для одной архитектуры, перенести на другую архитектуру. Это давно работает. Что у нас нового было? Проведем небольшой анализ.

Особенно заметных результатов добилась фирма Transitive. Это выдающаяся фирма в этом плане. Она переносит программы с любой архитектуры на любую. Когда Intel подружился с Apple, и Appple с PowerPC перешел на x86, то Transitive предложили свои услуги. И все программы, которые были написаны для PowerPC, на интеловской архитектуре также исполнялись, причем изменение архитектуры было незаметно для пользователя. Ну, конечно, помедленнее, но если портировать, то исполнение будет быстрее. Потом они предложили IBM перенести программы «Интел» на Power. И тоже перенесли, теперь эта фирма взялась за архитектуру Sun, и т. д. Эта фирма очень преуспевает, но она делает чисто программистскую работу, переносит программы с одной архитектуры на другую.

Двоичная компиляция имеет еще и другой смысл. Вы можете взять какой-нибудь двоичный код, и сильно его инструментировать. А инструментировать – это все равно переделывать. Если имеется в виду чисто программистская двоичная компиляция, то это делается, чтобы добавить какие-то функциональные возможности для пользователя, перенести с одной архитектуры на другую. Transitive так и говорит, если вам нужно скорость получить, то не пользуйтесь Transitive, лучше портируйте. Если вам нужна быстрая трансляция, то, пожалуйста, получите: минус скорость, плюс функциональные возможности.

Мы придерживались противоположного мнения. Наша идея заключалась в том, чтобы придумать новую архитектуру, которая будет намного быстрее старой. А для того чтобы сохранить совместимость, предложили взять двоичную компиляцию. Здесь, напротив, никаких заметных функциональных улучшений, для пользователя это должна быть старая машина, но скорость должна быть высокой. Ровно противоположно тому, что делает Transitive. Там функциональные возможности, но скорости нет. У нас скорость, но нуль новых функциональных возможностей. И это был наш подход. Это мы придумали. И реализовали в «Эльбрусе-3».

Я описал вам все наши технологии. Последний очень важный эпизод. В 90-е годы границы открылись. Мы потеряли возможность отладить «Эльбрус-3», но мы стали открыты, наши работы стали известны на Западе.

Первый контакт состоялся тогда благодаря Михаилу Горбачеву. Горбачев поехал с визитом в Калифорнию, а ответный визит в Россию нанес Т. Д. Роджерс, президент фирмы Cypress Semiconductor. Я ему рассказал про все

наши работы. Он написал про нас статью в американской прессе, и нашими работами заинтересовался Bill Joy, исполнительный директор Sun. Он приехал в Москву в конце 1990 года и захотел встретиться со мной. Мне устроили с ним встречу в ресторане «Прага», и мы 4 часа с ним разговаривали. Он был неподдельно восхищен нашими работами.

Он предложил, чтобы Дэйв Дицел занялся этой работой. Мы начали переписываться с Дэйвом. Где-то в середине 1991 года к нам приехали две фирмы: Sun Microsystems и Hewlett-Packard. HP в то время начинал разработку микропроцессора, который впоследствии стал называться Itanium. Это тоже, к сожалению, так же как 432 машина, была плохо реализованная хорошая идея. Из Sun к нам приехал Дэвид, и все десять дней мы сидели и обсуждали. Уезжая, Дицел сказал, что эта поездка ему на всю жизнь запомнится. И действительно, она перевернула всю его жизнь, мы стали с ним работать.

Три года Sun финансировала нас. Мы разработали вариант машины с широкой командой и с двоичной компиляцией, которая могла переводить Sun-вские двоичные команды в команды нового «Эльбруса». Дэйв хотел, чтобы фирма Sun пошла по нашему пути. Была невероятная война. Дэйв не смог их убедить и ушел из Sun, организовал фирму Transmeta, которая теперь всем хорошо известна. Он ничего другого не придумал, как делать машину с широкой командой и с двоичной компиляцией. Тогда он мне этого не сказал и, наверное, правильно сделал, мы могли разболтать все его секреты. Он некоторое время держал все в тайне. Transmeta сделала эту машину, и она некоторое время существовала. Но потом, конечно, не выдержала конкуренции с «Интелом».

Дэйв мне говорил, что если бы он знал, что будет так трудно конкурировать с «Интелом», то он бы не взялся за это дело. Мы поступили более разумно. Мы пытались снаружи конкурировать. Наша технология была выведена на западный рынок, и она произвела большое впечатление. Двоичная компиляция сейчас – это писк моды. Многие думают о двоичной компиляции, потому что суперскаляр – это предел. Дальше с суперскаляром уже идти невозможно. Только двоичная компиляция может спасти мир от застоя.

В принципе, мы пытались найти партнера. Думали, что мы сделаем что-то чуть более успешное, чем «Интел». Наиболее разумная политика, наоборот, придти в «Интел» и изнутри все изменить. Затрат меньше, но это очень непростой путь. Это работа в большой фирме. Большая ответственность у тех, кто руководит этой фирмой. Многомиллиардные затраты. Представьте, как можно повернуть этот гигантский корабль куда-то в другую сторону? Я бы не взялся за такую работу, чтобы организационно принимать такие решения. Но это было единственным вариантом, и я пошел по этому пути. А

сейчас и Дэйв Дицел перешел в Intel. Теперь он вице-президент Intel. И мы опять начали совместную работу. Объединились мы снова и с Владимиром Пентковским, и сейчас довольно большая группа в Москве занимается развитием архитектуры, уже интеловской.

Я думаю, есть ли в мире еще такой коллектив, как наш, созданный в ИТМиВТ, который 50 лет активно работает в области архитектуры как единое целое? Вряд ли. Мы продолжаем эту активную работу, и я продемонстрировал, что мы никогда не были позади западных разработок, не было ни одного такого года. Я надеюсь, что наши работы и далее будут хорошо восприняты.

Это все – результаты тех ранних лет, когда и Андрей Петрович все здесь начинал. Если бы не было того начала, то не было и того, что сейчас происходит, – соединения архитектуры машин и языковых средств. Андрей Петрович, как вы помните, начинал и языки, и теоретическое программирование, многие тогда критически к этому относились. Может быть, не все получилось, но это было правильно.

Я все время думаю, что во всей моей работе я не относился, как многие, к информационным технологиям как к ремеслу. Если ставится цель сделать что-нибудь, чтобы быстрее работало, здесь приляпать, там приляпать, – это всегда дает плохой результат. Все должно начинаться с фундаментальных вещей. Надо фундаментальные вещи улучшать. Если вы проанализируете то, что я сказал, то я всегда занимался так. Это, конечно, не просто, это требует больше времени, больше опыта. Я думаю, что когда Андрей Петрович говорил о теоретическом программировании, то это было как раз направлено в то же русло. Мы с честью проводим этот подход, и мы все хорошо помним это влияние Сибирской школы программирования. Хочу завершить тем, что те замечательные годы, которые были расцветом творческой мысли в наших коллективах, они незабываемы. Они дают хороший результат.

ВОПРОСЫ:

А. А. Берс (ИСИ СО РАН): К вопросу о параллелизме. Внутри связи процессор–память они продолжают быть все шире. А снаружи, наоборот все шире используются последовательные шины. Только ли потому, что несущие частоты передачи растут?

Ответ: Нет, я полагаю, что, наоборот, сейчас всюду шины выкидывают и переходят на коммутируемые многомерные связи типа Point-to-Point, аппаратура теперь это позволяет. Что касается меня, то мы никогда не использовали шины, как никогда не использовали микропрограммирование, там все замедляется.

Е. Н. Кашменский (ИСИ СО РАН): Архитектура широких VLIW в основном помогает в параллелизме внутри какой-то отдельной процедуры. Грубо говоря, между какими-то ветвлениями. Есть ли какие-то аппаратные архитектуры, которые помогают достичь большего параллелизма с участием ветвлений?

Ответ: Внутри процедуры ветвления очень хорошие. Там выполняются спекулятивные вычисления, закидывание load наиболее раннее время, там все отработано очень хорошо. А между процедурами... В первом «Эльбрусе» мы принимали некоторые меры, мы пытались наложить процедуру на процедуру. В «Эльбрусе-3» этого не было.

А. А. Берс: Надо это делать аппаратно или компилятором? То, что это не надо давать пользователю, это понятно, можно отдать планировщику, но тут много работы может быть сделано статически, поэтому, зачем ее перекладывать на машину?

Ответ: Это важный вопрос, но я не буду сейчас на этом останавливаться, это вопрос дальнейшего проектирования.

Е. Н. Кашменский: Какая характерная средняя ширина внутри одной процедуры?

Ответ: Мы в «Эльбрусе» заметили, что это сильно зависит от задач, но это очень широкий диапазон. Есть задачи, которые шире машины. Шире «Эльбруса». Например, циклы. Есть задачи, которые очень узкие. У нас компилятор планирует по-другому. Когда поток широкий, то компилятор не использует никаких спекуляций, там и так много вычислений. А когда нить узкая, то есть другое преимущество: много доступного оборудования, и мы начинаем очень агрессивно выполнять спекулятивные вычисления, и таким образом ускоряемся.

Е. Н. Кашменский: Тут тоже есть барьер? Как в суперскаляре – четыре команды? Тут порядка 20 или около?

Ответ: Да, мы исследовали этот вопрос на существующем наборе. Предел у нас – 22. Мы знаем что по мере увеличения параллелизма, конечно, эффект начинает снижаться, но до 18–20 все-таки эффект ускорения стоит того. Выше на существующих методах программирования, а дальше будет все меньше и меньше. Это тонкий баланс: с одной стороны увеличивается скорость, а с другой стороны увеличиваются аппаратные ресурсы, которые все дешевеют и дешевеют. Сегодня за 5 % скорости жалко отдавать, а в будущем, может, и будет стоить.

А. А. Берс: Можно ли переключаться по тактикам? Кроме распараллеливания существует еще и мультипрограммная загрузка, а она требует переключения контекста. Спрашивается, что тут можно? Каковы перспективы того, что переключение контекста окажется уже не таким сложным как сейчас? Если ты из одного кольца защиты захочешь полезть в другое, то тебе придется делать громоздкую программу.

Ответ: Тут переключений контекста может быть два. В процедурах и в задачах целиком. Для задач сейчас идет переключение контекстов.

А. А. Берс: Да в задачах мы можем заваливать процессор под завязку. Это еще на М-20 делалось. Мультипрограммная набивка, из этого операционные системы вышли. И это делалось еще на тех машинах, А вот потом появились thread'ы, и это, вообще говоря, попытка перенести мультипрограммность внутрь задачи. Но на нее навешивается огромный контекст, особенно в процессорах «Интела». Что может по этому поводу улучшиться архитектурно?

Ответ: Здесь много чего можно сделать, но пока об этом рано вслух говорить.

А. А. Берс: В свое время были такие машины Nord-10, Nord-100. У них было по 16 комплектов регистров. Они просто перебрасывали указатели ОС. То же и в «Пентиуме» планируется, что «Пентиум» только снаружи 86-й, то это всем известно. Это хорошо?

Ответ: Это же просто увеличение оборудования, а надо хорошо его использовать. И так сейчас делают. Мой подход в том, что самое главное – это single-thread ускорять как внешнее. Труднее всего ускорить выполнение одной программы.

А. А. Берс: Или не складывать в single-thread то, что можно разнести.

Ответ: Конечно.

В. А. Непомнящий (ИСИ СО РАН): Скажите, пожалуйста, а можно ли «Эльбрус-3» уложить в обычную РС-шку по размеру?

Ответ: Конечно. Сейчас «Эльбрус-3» называется E2k, его изготовили на Тайване. Это просто один кристалл, и есть уже РС на «Эльбрус-3». Это уже сделано.

Почему такие работы можно делать только в «Интеле»? Это очень просто, «Интел» обладает совершенно гениальными фабриками. Вот сейчас чипы делаются уже на 45 нм, скоро будет 32 нм, затем 22 нм. Конечно, можно делать архитектуру отдельно, а потом посылать на тайваньский завод для реализации. Но Тайвань выдает способы, как проектировать на но-

вом процессоре, когда этот процессор абсолютно отлажен. А в «Интеле» отлажен процесс, готов проект самого процессора. То есть, если вы работаете без фабрик, снаружи, вы заведомо на три–четыре года отстаете. А это все, это смерть. Здесь отставать и на полгода нельзя. Поэтому единственный вариант для микропроцессоров и, может быть, для памяти, там, где есть очень большая массовость, большая серийность, нужно иметь свой завод. Причем если этот завод малосерийный, то конкурировать опять-таки будет невозможно. Когда массовый выпуск, то все дешевле намного. Это драма. Поэтому мы в «Интеле». Работу можно продолжать только в такой организации.

<…>: Про бинарную компиляцию. Увеличение производительности и увеличение функциональности. Что Вы имели в виду под увеличением функциональности?

Ответ: Под увеличением функциональности я имею в виду, например, перенос с PowerPC на x86. Вот это функциональность. Это то, что видно человеку, а не что-то там добавляется, для того чтобы получить новую возможность. А то, что мы делаем, мы говорим: «Вот тебе, это интеловская машина». Только она быстрее и там никакого улучшения функциональности нет. Кроме скорости.

Е. Н. Кашменский: Почему в чипе «Эльбруса» 300 МГц?

Ответ: Известно, что в custom-design каждый транзистор лудится вручную. А есть mask design. Несколько сот человек вручную делают маски. Сейчас 45 нм. А луч лазера, который маски проектирует, – 190 нм, если вы сделаете более высокочастотный, то он поглощается, им нельзя пользоваться. Поэтому маски делаются за дифракционным пределом. Маски искусственно искажают, чтобы они давали правильную проекцию. Сидят сотни людей, вручную правят маски. То есть это проектирование современного процессора. Это сотни, почти тысяча человек. Custom design, по сравнению с библиотеками (ASIC), улучшает тактовую частоту в шесть раз.

Теперь расскажу, как проектировался «Эльбрус-3». У нас не было денег даже библиотеки купить. Был контракт с «Аванти» (теперь Synopsis), мы разрабатывали библиотеки для них. Но для того чтобы использовать их в «Эльбрусе», нужно было все равно платить деньги. И мы пользовались даже не покупными библиотеками, а библиотеками open source, а это библиотеки очень низкого качества, раз в 6. Умножьте 300 на 6, это и будет 2 ГГц.

Е.Н. Кашменский: Если провести технологическую проработку Эльбруса, довести его до 2 ГГц, как эта архитектура в такой реализации будет конкурировать, например, с Intel Core Duo?

Ответ: Это сложный вопрос. Во внедрении микропроцессора много коммерческого. Мы сравнили, мы обыгрываем их на тестовых задачах. Я думаю, что он, конечно, не будет уступать.

Т. С. Васючкова (НГУ): А во сколько раз обыгрываете?

Ответ: На универсальных задачах SPEC по сравнению с 300 МГц в 1,5–2 раза. А на мультимедийных задачах 300 ГГц Эльбрус работает в 1,5 раза быстрее, чем современная 2,5 ГГц машина. Потому что эти задачи хорошо распараллеливаются. Специализированные западные машины, которые для графики сделаны, тоже показывают такую скорость. Но мы-то сделали универсальную машину.

Ф. А. Мурзин (ИСИ СО РАН): Некоторое высказывание хочу сделать. В свое время я познакомился с системами команд CRAY и «Эльбруса», чтобы увидеть, как что устроено. И впечатление такое, что там много похожего. В «Эльбрусах» осуществили мощнейший рывок, чувствовались идеи. Это из прошлого. А из настоящего такой вопрос. О процессоре CELL, что Вы можете сказать?

Ответ: Я не очень с ним знаком. Я просто знаю высказывания друзей. Вначале он производил сильное впечатление. Это не результат моего анализа. Это от людей, которые вокруг меня. Сейчас он уже не вызывает такого восторга. Сейчас на переднем крае фирма Nvidia, это серьезная фирма.

Т. С. Васючкова: В каком году сгорела лампа, когда испытывали противоракетную защиту?

Ответ: В 1959–1961 гг. на полигоне.

<...>: Вряд ли в этой аудитории знает кто-нибудь машины серии 5Э, но наша обороноспособность держалась очень сильно благодаря этим машинам. У «Эльбрус-3» судьба такая же, это будет чисто военная машина?

Ответ: Это чисто военная машина, потому что конкурировать на рынке невозможно. Сами понимаете 300 МГц с 4 ГГц, ну как может конкурировать? Но государство финансирует. В основном, чтобы быть независимым от западных разработок. Параллельно это дает возможность работать в области архитектуры.

III Ершовская лекция по информатике
20 апреля 2009 г., Академгородок
Конференц-зал ИВМиМГ СО РАН

ОБ ОСНОВАНИЯХ ИНФОРМАТИКИ

Андрей Александрович Берс
Институт систем информатики им. А.П. Ершова СО РАН
Новосибирск, Россия

Данная лекция – третья из нашего цикла, участвовать в котором, конечно, большая честь. Можно считать, что традиция состоялась и поддерживается, хотя и с вариациями, поскольку в этом году день рождения Андрея Петровича – 19 апреля – попал на воскресенье, да еще и на Пасху, пришлось сдвинуться на день.

Другая вариация – первые две лекции были без иллюстраций, более того, Ю. Л. Ершов, начиная первую лекцию, подчеркнул, что «в публичной лекции, лектор должен брать на себя ответственность и держать аудиторию как умеет, если не в напряжении, то, по крайней мере, стараться поддерживать интерес своей речью». Конечно, каждому хочется прослыть хорошим лектором, но, с другой стороны, я вспомнил, что когда Андрей Петрович готовился к докладу «Программирование – вторая грамотность» на открытии Всемирной конференции ЮНЕСКО и ИФИП «Применение ЭВМ в обучении» в Лозанне в 1981 г., то оформление слайдов он заказал профессионалу, главному художнику журнала «Химия и жизнь» М. Златковскому.

Я буду говорить об основаниях информатики, т. е. о той системе категорий, через которую все остальное в информатике может быть выражено. Говоря об основаниях, нельзя не сказать и об обоснованиях. Потому что если основания – это часть содержания информатики, то обоснования – это то, что лежит вне нее, но питает ее и связывает с применениями и внешними источниками. Кроме того, соответствующий рассказ невозможен без обращения к историческим корням.

Хочу еще добавить, о чем я ***не*** буду говорить. Я заведомо не буду говорить об организации и промышленном производстве программного обеспечения, и, во-вторых, я не буду говорить о приложениях информатики потому, что за те 60 лет, которые она существует, они настолько сильно распространились, что вполне достаточно будет пары примеров.

Первый пример. 30 лет назад, когда мы занимались электронной подготовкой изданий, это вызывало очень большое сопротивление, потому что

касалось перестройки полиграфической отрасли в целом. Тем не менее, все так и получилось: сегодня в типографиях нет наборных цехов. Все наборные действия переведены туда, куда им и полагается, – в издательства.

Второй пример. Сейчас у каждого в кармане лежит устройство, по скорости заметно превосходящее БЭСМ-6, которая была в ВЦ главной рабочей лошадкой, и еще 10 лет назад никому не приходило в голову начинать телефонный разговор с вопроса «ты где?», теперь это стандартная ситуация.

Я начну с определения, выделив три аспекта.

Первое. Информатика – это мощная конструктивная деятельность, в результате которой создаются неимоверно сложные системы, но таким способом, чтобы ими можно было легче пользоваться и как можно проще было с ними общаться.

Второе. Информатика – это наука, которая изучает законы хранения, обработки и передачи информации. С помощью информатики можно объяснить сложность и таких элементарных актов, как, например, «поднять левую руку». Каждый может это сделать, а вот написать программу, которая моделирует такую вещь, – занятие сложное. И это делается обычно с помощью некоторого количества языков.

Одной из важных свойств науки является различение различного при сохранении общности целого.

И наконец, третий компонент Информатики – это то, что она может и поэтому должна описывать информационно-деятельностную структуру мира нашей цивилизации и культуры, в том числе аккуратно описывать взаимодействия между ее субъектами – людьми и между людьми и компьютерами.

Я отделяю третье от второго потому, что мировоззрение или методология шире, чем наука. Во-первых, методология учитывает вопросы ремесла и искусства, а во-вторых, она включает исследующего субъекта в само рассмотрение, что наука, вообще говоря, запрещает делать. А в рамках методологии можно даже позволить себе двигаться в противоречиях. Дело в том, что формальные системы либо очень мелкие, как, например, теория групп, которая охватывает почти все, либо очень узкие, но глубокие приложения. Такая метафора: прежде чем копать свои собственные шесть соток на штык, надо добраться до своего огорода. Навигация оказывается движением по миру противоречий.

Хочу заметить, что многие люди приходят в информатику, получив техническое образование, большая часть приходит из математики, а третьими приходят те, кто набрался методологического опыта и знаний. Этот опыт показывает, что информатики сами начинают серьезнее относиться к про-

блемам мировоззрения и философии. Правда, чтобы чистый философ пошел в информатики, я не видел. Ему образования не хватает, как я полагаю. Таким образом, я представил «три источника и три составные части» информатики.

В дополнение хочу заметить, что ничто так быстро не развивается как информационные системы. Старшее поколение помнит ЭВМ, которые не влезли бы в этот зал. Первая машина, которую я увидел в Академгородке, занимала целое крыло первого этажа Института геологии, делала 20 000 операций в секунду и имела 4096 ячеек памяти, что примерно равно 16 килобайт. Если кому-нибудь предложить сегодня сделать систему, которая будет работать на такой памяти, то, наверное, все откажутся. Мы избаловались. Мы перешли от тысяч операций к миллиардам и от десятков тысяч элементов тоже к миллиардам. Потому что количество вентилей на чипе графического процессора уже приближается, а в некоторых случаях и превышает миллиард элементов, мы переходим от мегагерц к гигагерцам, и начинаем измерять производительность не в терафлопах, а в петафлопах, т. е. еще в 1000 раз больше.

Все системы информатики растут чрезвычайно быстро, и, кроме того, они чрезвычайно сложны. На мой взгляд, сложнее наших комплексов только биологические структуры, а все остальное – тоже не так просто – все гораздо проще.

А началась информатика в 1949-м году с создания вычислительных машин и программирования для них. В основания машин положены три принципа:

- Возможность делать преобразования между значениями, т. е. выполнять некоторые действия.
- Возможность хранить результаты счета, т. е. иметь память.
- Универсальность машин, что означает: сама машина только может циклически делать шаги по программе, а вся сложность функционирования программно-аппаратных систем закладывается в программу.

Лично для меня информатика началась как программирование с книги М. Уилкса, Д. Уиллера и С. Гилла «Составление программ для электронных счетных машин». Это первая книга, открыто изданная в СССР в 1953-м году, была написана в 1951-м про машину, которую запустили в 1950-м.

До этого было еще две книжки – описания машины БЭСМ и программирования на БЭСМ, но лично я смог их посмотреть только году в 1979-м, когда все это рассекретилось. Андрею Петровичу в этом смысле повезло

больше, поскольку он слушал первые в Союзе лекции по программированию, которые читал А. А. Ляпунов.

Maurice Vincent Wilkes
Род. 26 июня 1913 г

Машина ЭДСАК – первая, реально сделанная машина с хранимой вместе с данными программой. Построенная в США на два года раньше машина ЭНИАК имела коммутационные доски, как в счетно-аналитических табуляторах IBM, и коммутируемую внешнюю программу.

Главный конструктором ЭДСАК, построенной в университете Манчестера, как и автором упомянутой выше книги, был и поныне здравствующий Морис Уилкс.

Трое советских ученых были в 1996 г. удостоены медали «Пионер компьютерной техники» (Computer Pioneer, учреждена в 1981 году) – самой престижной награды всемирного компьютерного сообщества.

Сергей Алексеевич
ЛЕБЕДЕВ
1902–1974

Алексей Андреевич
ЛЯПУНОВ
1911–1973

Виктор Михайлович
ГЛУШКОВ
1923–1982

На лицевой стороне медали выполнен барельеф Чарльза Бэббиджа, а на оборотной выгравирована формула награждения:

«Лебедев, Сергей Алексеевич – разработал и построил первый советский компьютер и основал советскую компьютерную промышленность».

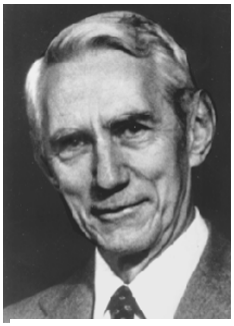
«Ляпунов, Алексей Андреевич – разработал теорию операторных методов для абстрактного программирования и основал советскую кибернетику и программирование».

«Глушков, Виктор Михайлович – основал первый в СССР Институт кибернетики на Украине, разработал теорию цифровых автоматов и компьютерной архитектуры, а также рекурсивный макроконвейерный процессор».

Аланом Тьюрингом было разработано понятие абстрактной дискретной вычислительной машины, получившей впоследствии название машины Тьюринга, способной имитировать (при наличии соответствующей программы) любую машину, действие которой заключается в переходе от одного дискретного состояния к другому. Отсюда, кстати, возник миф что машина и программа это одно и то же.

Alan Mathison Turing
1912–1954

Тезис Черча–Тьюринга, утверждающий невозможность существования формальной, чисто механической процедуры, которая позволяла бы решать, выводимо ли данное высказывание из некоторого набора математических аксиом, имел фундаментальное значение: Тьюринг и Черч вместе с Геделем похоронили надежды Д. Гильберта и его последователей, полагавших, что математику как самую формализованную часть человеческого знания можно представить в виде набора аксиом и теорем.



Claude Elwood
Shannon
1916–2001

Клод Элвуд Шеннон сумел ликвидировать разрыв между алгебраической теорией логики и ее практическим приложением. Свои идеи относительно связи между двоичным исчислением, булевой алгеброй и электрическими контактными схемами К. Шеннон развил в докторской диссертации, опубликованной в 1938 году. К. Шеннон, один из создателей математической теории информации, в значительной мере предопределил своими результатами

развитие общей теории дискретных автоматов, которые являются важными составляющими информатики.

John von Neumann
1903–1957

Джон фон Нейман (по рождению венгр – Neumann János Lajos). Фон Нейман построил замечательную систему аксиом теории множеств, такую же простую, как гильбертова система для евклидовой геометрии. Система аксиом фон Неймана занимает немногим более одной страницы печатного текста. В 1945 году ученым как США, так и Великобритании был разослан «Предварительный доклад о машине EDVAC», в котором описывались архитектура машины и ее логические свойства. Материалы отчета не публиковались в открытой печати до конца 1950-х годов. Известность фон Неймана как крупного ученого сыграла свою роль – изложенные им принципы и структура ЭВМ стали называться «фоннеймановскими». Читатели «Доклада» были склонны полагать, что все содержавшиеся в нем идеи, в частности решение хранить программы в памяти компьютера, исходили от самого фон Неймана. Однако их первыми авторами были создатели ЭНИАК Дж. Мокли и Дж. Эккерт, которые обсуждали вопрос о записываемых в памяти программах, по крайней мере, за полгода до появления фон Неймана в их рабочей группе. Многим неведомо было и то, что А. Тьюринг, описывая свою машину, еще в 1936 г. наделил ее внутренней памятью, и фон Нейман читал работу Тьюринга.

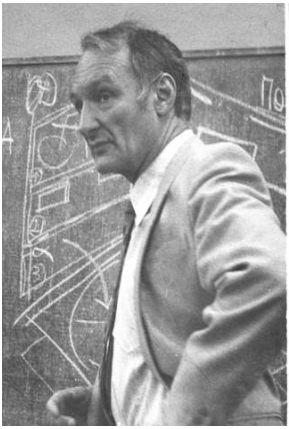
Grace Hopper
1906–1992

Грейс Хоппер сделала первый в мире транслятор для машины Марк-2 с языка Flow-o-Mathic. Когда при сбое работы, в 1970-м году в реле панели F был найден между контактами мотылек, она описала это в рабочем журнале как первый случай, когда был найден «баг», и таким образом возник термин «дебаггинг».

По ходу доклада я буду ссылаться на работы по теории и деятельности московского методологического кружка, который возник в 1952 г., пример-

но в то же время, когда появились компьютеры, во время дискуссии по проблемам логики на философском факультете МГУ, и назывался тогда московский логический кружок.

Его отцами-основателями являются А.А. Зиновьев, которого упоминал Ю.Л. Ершов в первой лекции, Г.П. Щедровицкий, Б.А. Грушин (много занимался социологией); Мераб Константинович Мамардашвилли, великий философ 20 века. К 1954 г. кружок преобразовался в методологический и лидером его стал Г.П. Щедровицкий.



Георгий Петрович Щедровицкий
1929–1994

Московский методологический кружок – это философско-методологическая и практическая школа интеллектуальной работы со своими оригинальными приемами и уровнем организации коллективного рефлектирующего творчества.

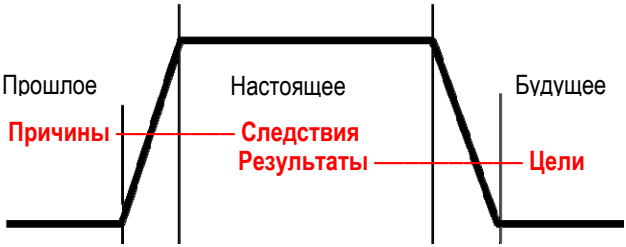
Кстати, когда мы строили систему БЕТА, мы использовали приемы московского методологического кружка: записывали дискуссии на магнитофон, транскрибировали их и использовали в работе стенограммы этих семинаров.

ММК превратился в методологическое широкое движение, замалчиваемое, к сожалению, «официальной» наукой. Еще когда была дискуссия по проблемам логики, то они там «дали жару» марксистско-ленинским профессорам, хотя Зиновьев был тогда аспирантом, а остальные – студентами. Они выступили на дискуссии так, что, как рассказывают, один из профессоров получил инфаркт.

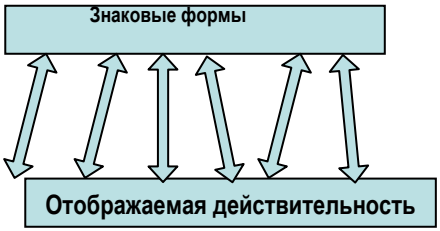
Что для нас, информатиков, существенно из того, что они сумели сделать впервые в мире?

Было показано, что формой существования цивилизации является деятельность, которая связывает ее в нечто целое. Что системы делятся на естественные и искусственные, и что они существенно различаются, потому что естественные системы описываются причинно-следственными связями – отвечают на вопрос «почему это происходит?». Искусственные же системы отвечают на вопрос «кому это нужно?», потому что они следуют целям, которые надо достигнуть, и значит, они подчиняются разным законам. Пример этой разницы можно найти у Стругацких в романе «За миллиард лет до конца света» – там это хорошо обыграно. Проиллюстрировать

это проще всего так: следствия всегда имеют причиной то, что происходило в прошлом, а результаты деятельности всегда определяются целями, которые вообще говоря, отнесены к будущему.



В ММК много работали в этом направлении и показали, что деятельность как форма существования цивилизации составляет структуру из конкретных деятельностей. Каждая конкретная деятельность задается целью, обязательно имеет начало и завершается. И, следовательно, конкретная деятельность – это искусственная система. После чего им удалось обосновать тезис, что всякая деятельность выполняется по программе. И если мы хотим изменить ход деятельности, то надо выйти на цели программы, а если вы хотите изменить искусственно существующую систему, то надо добраться до программы, которая ее породила и ей управляет, и вносить изменения в нее.



Следующая проблема, которую решили в ММК, это создание содержательно-генетической логики. Ими рассматриваются знаковые системы как обозначающие нечто существующее объективно в действительности. Так это делают и в математике, но математики считают, что если они описали некоторую знаковую форму (некоторую структуру знаков), то структура обозначаемой действительности изоморфна структуре знаковой формы. Оказалось, что на самом-то деле это не так, и что нужно, поскольку они не изоморфны, чтобы не потерять сути, рассматривать оба эти плана совместно. Это называется схемой двойного знания. ММК было обосновано также языковое мышление. Они показали, что мыслительная деятельность, которой все мы занимаемся, – это деятельность со знаковыми формами. Ее можно отличить от коммуникативной деятельности, от деятельности понимания, можно отделить от производственной деятельности и достаточно хорошо специфицировать. Все это было сведено в некоторое общее ком-

плексное, представление, которое называется «Системная мыследеятельность» (СМД). Именно этот комплекс как категорию я и буду иметь в виду, в качестве обоснования принимаемых свойств оснований информатики.

В 1964 г., после Томской конференции по логике Щедровицкий и К^{o} оказались в Академгородке и произвели на меня столь же неизгладимое впечатление, как и указанная ранее книга М. Уилкса. А к 1968 году мне удалось обосновать тезис, что то, что происходит в вычислительной машине, есть не просто модель какой-то внешней деятельности, а что это самая настоящая деятельность, которую машина осуществляет. Именно поэтому, мы и можем с ней взаимодействовать, можем поручить машине часть работы, так же как мы можем поручить ее лаборанту или научному сотруднику. В этом мы постоянно убеждаемся. Хочу еще раз подчеркнуть, что изложение методологии весьма конструктивно и поэтому вполне «съедобно» для господ программистов и информатиков.

Я попробую теперь провести некоторую периодизацию, начиная с 1950-х годов по десятилеткам. При этом в каждом из них я логически выделяю главное (ведущее, или просто модное) достижение, которое определяло содержание этого десятилетия.

50-е годы: Первое появление ЭВМ, составление программ для вычислительных машин и развитие компонентов вычислительных машин. Причем машины делались штучно. Машины были огромные и вообще от линкоров отличались только тем, что не имели собственных имен. А так вообще-то были столь же медленны, неповоротливы и т. д.

60-е годы: Это время, когда я приехал в Сибирское отделение. Оно характеризуется тем, что от составления программ для ЭВМ перешли к выражению программ, как говорил Андрей Петрович, «на ЯВУ» – на языках высокого уровня. Это подарило нам возможность описывать гораздо более сложные задачи и сделало программирование более человеческим. Появились почти все возможные структуры, которые мы сейчас считаем принадлежащими к основам информатике.

70-е годы: Следующие десять лет было посвящено, главным образом, обеспечению эффективности применения вычислительных машин. Оказалось, что участие человека в работе вычислительной машины вредно, он слишком медлительный. Поэтому были предложены различные архитектурные и программные средства, из которых затем образовались операционные системы. Например, чтобы процессор не простаивал, ожидая медленные устройства, была создана технология мультипрограммного заполнения машины.

80-е годы: Самым главным явилось освоение объектов. Объекты и ранее валялись под ногами, потому что элемент памяти, который осуществляет хранение, – это как раз и есть элементарный объект. До этого только сложные структуры действий сворачивались в процедуры. А в объекты можно не только сворачивать действия и какие-то процедуры, но и сворачивать данные в компактные формы, упрятывая внутрь множество деталей. В этом и состоял главный пафос. Заодно возникло и некоторое количество неприятностей, о которых я обязательно скажу, но чуть позже.

90-е годы: Посвящены сетям, появились интернет, асинхронная обработка, проблемы взаимодействия компьютеров – наиболее модные темы.

Сегодня, в следующее десятилетие, которое в следующем году закончится, у нас есть всемирная Сеть. В ней есть всемирные социальные службы. Развиваются огромные микропроцессорные кластеры. На них пытаются хорошо сделать параллельное решение сложных задач, чего до сих пор хорошо делать не умеют. Это же дает возможность эффективно реализовывать средства массовой информации, средства связи, глобальные средства навигации. И, в основном, большой рост приложений. Из аппаратуры ничего нового, после кластеров. Разве что кластеры теперь стали называться ядрами и размещаться на чипах. А так принципиального мало чего изменилось, по-моему.

Я попытался выразить это следующим образом (в конце 2000 г.):

Проблемы и задачи на III тысячелетие:

Субъектные взаимодействия,
Смешанное исполнение,
Эффективный параллелизм,
Эффективные *Вирт*-Машины
Речевое общение

Здесь давался прогноз, чем мы будем заниматься в этом тысячелетии, пока он оправдывается. Развивается спецификация взаимодействия субъектов между собой, в качестве нового способа описания проблем. В смешанном исполнении, когда трансляция и интерпретация переслаиваются так, как нам это надо, это хорошо подтверждается JIT-ерами. Есть достижения и в эффективности параллелизма. Все большую роль начинают играть виртуальные машины, которые я призываю вас называть Вирт-машинами, Никлаус Вирт этого заслуживает. А вот из диктовки машинам пока ничего не получается, хотя читать с листа они, как известно, уже научились.

На мой взгляд, в истории информатики можно выделить три самых главных рубежа – введение языков, освоение объектов и рассмотрение взаимодействия компьютеров между собой как самостоятельных и независимых субъектов.

Теперь поговорим о главных рубежах, по порядку и подробнее:

Про языки. Мы перешли на знаковые формы, на тексты программ. Появилась возможность писать содержательные обозначения, например какую-нибудь ячейку памяти, в которой хранится последнее вычисленное значение, можно обозначить: “здесь_хранится_ПоследнееВычисленноеЗначение”.

Появился, как у всяких текстов, синтаксис, но я хочу обратить ваше внимание на то, что у программных текстов имеются две семантики, и это было с самого начала. Сама семантика текстов, которая позволяла использующиеся вхождения соотносить с определяющими вхождениями, делать идентификацию, и позволяла организовывать вызовы процедур, организовывать движение по тексту. С другой стороны – семантика того, что будет выполняться, когда программу преобразуют и приведут к форме, пригодной для исполнения на машине. И вторых семантик изначально было несколько, например, императивная или дескриптивная и пр. И это надо иметь в виду.

Естественно, что появились все основные структуры, которые мы все прекрасно знаем. Описания, операции, выражения, операторы. Следование, переходы, ветвление, циклы, вызовы. Блоки, модули, файлы: напоминаю, чтобы не потерялось.

Учет национальных особенностей и адаптация языков программирования к региональным алфавитам. Это была целая проблема, естественно, что все программы, которые писались на Западе, писались по-английски. Против этого возражали активно только французы и весьма активно наш Отдел программирования. Мы этим делом занимались всерьез, причем воевать приходилось не только с американцами, но и с нашими, например, со многими москвичами. Во всяком случае, нам с Александром Федоровичем Раром, при полной поддержке Андрея Петровича, удалось добавить в стандартное описание международного языка Алгол-68 формальные правила, как по нему строить национальный вариант языка так, чтобы можно было, например, печатать программу, даже если у вас нет машинки с латинскими буквами, без каких-либо уступок. Теперь-то это все не так сложно, так как появился Unicode. А если у вас есть Unicode, вы можете писать хоть китайские иероглифы в качестве идентификаторов.

Самый важный момент, который здесь следует отметить, с языками появилось понятие данных. И появились понятия: тип данных и представление

данных. При этом среди типов выделились простые, составные, динамические типы.

Вот примеры простых типов: логические, битовые и байтовые. Вещественные и целые числа, указатели. Составные типы: массивы, структуры, строки. Динамические типы: списки, деревья. Динамические типы ввел Джон Маккарти в Лиспе.

Понятие типа эволюционировало. Изначально тип определялся как множество значений, которое данная величина может принимать. Через некоторое время перешли на алгебраический способ определения. Типом стало называться множество операций, которое можно производить с данной величиной. Так же стали описывать и типы объектов, когда ввели в обиход объекты.

Кроме хорошего, появилось некоторое количество трудностей. Коллизия обозначений: поскольку количество слов, которые мы используем, не так велико, они начинают совпадать. Тогда были придуманы такие конструкции, как блок и модуль. А если несколько человек делали части одной программы, оказывалось – то, что надо стыковать, называется по-разному. Вот там тоже возникали соответствующие проблемы.

Кроме того, как я уже говорил, происходит последовательное, постепенное, но постоянное, отчуждение исполнения программы от программиста, потому что между ними стоит транслятор и не какой-нибудь, а оптимизирующий компилятор, и то, что вы написали, и то, что будет исполняться, совсем друг на друга не похоже.

Вместе с процедурами появился побочный эффект. Когда вы выполняете некоторое действие, а потом оказывается, что где-то что-то изменилось, хотя вы-то, вообще говоря, этого даже и не имели в виду. Поскольку появилась вычислимая адресация данных через указатели, возникла проблема висячих указателей, которая делает программу неработоспособной. Автоматически стали порождаться величины и занимать память. Ее надо было освобождать. Появилось понятие мусора и появилось понятие сборки мусора.

Примером интерференции синтаксиса и семантики исполнения может служить «подстановка именем» в Алголе. Надо строку, представляющую фактический параметр, подставить на то место текста тела процедуры, которое занимает формальный параметр, причем сделать это во время исполнения. Но во время исполнения текста-то уже не существует. Поэтому возникает некая проблема, которую, конечно же, решили. Тем не менее, этот способ передачи параметров не прижился. Вместо него появилась подстановка по ссылке.

Возникли два режима работы с исходной программой: интерпретация и трансляция.

Очевидно, чистый интерпретатор, который ничего не делает с программой на языке высокого уровня, а прямо по ней двигается и ее исполняет, по-видимому, не существует – это было бы слишком медленно. Поэтому сначала идет некоторое промежуточное преобразование, а потом начинает исполняться то, что получилось, с учетом тех значений, которые находятся в памяти. Это и есть характеристическое свойство интерпретации, в то время как трансляция занимается преобразованием программ тогда, когда про значения, которые будут во время выполнения, еще ничего не известно.

Это отличие очень хорошо понимал и объяснил нам Андрей Петрович в работах по смешанным вычислениям. Это дало возможность переходить к смешанным исполнениям, когда трансляция и интерпретация могут делаться по очереди.

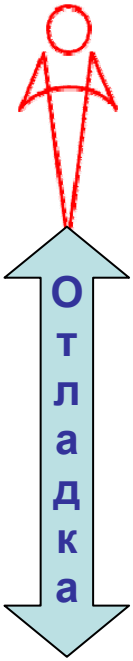
Кроме того, всегда была некоторая работа: либо на исходном уровне надо собрать куски и согласовать между собой, либо после некоторых преобразований надо рабочие подпрограммы состыковать.

В результате языковая парадигма дала нам в руки эффективные средства свертки действий в форме подпрограмм, процедур, модулей, появления стандартных библиотек и появления баз данных, в которых стандартным образом хранилась информация.

Я уже говорил, что 1970-е годы были посвящены в основном изменениям в архитектуре и эффективному использованию вычислительных процессоров. Значительно развилась и изменилась архитектура: появились многоуровневая память, многопроцессорные комплексы. Появились специальные и вспомогательные процесссоры, например, каналы ввода-вывода. В системе IBM/360-370 появилась совместимость для целого класса технически разных машин с одной архитектурой. Были построены всевозможные операционные системы, которые занимались планированием работы и выполнением ее без участия, более того, с отторжением пользователя.

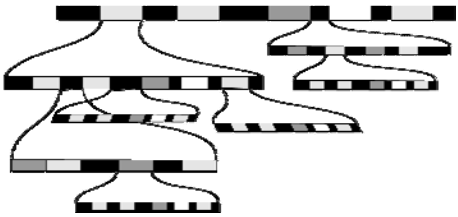
Один процессор старались набить «под завязку» – это мультипрограммный режим. Потом в программах стали выделять несколько процессов – мультипроцессный режим. Потом на процессорах их время стали делить на кусочки, квантовать время – это было разделение времени. Потом организовали сообщения. В результате получили системы коллективного пользования, где каждый терминал выглядит для пользователя так, как если бы он сидел за самой машиной в одиночку, как это и было в самом начале. К операционным же системам традиционно относятся системы хранения данных – файловые системы.

В результате все это выглядит, таким образом, как на схеме справа. Есть исходная программа. Она попадает в систему программирования. Здесь может быть и ряд таких программ. Они собираются, преобразуются, и создается некая исполнимая программа, которая поступает на вход операционной системы. В результате все программы разбиваются на некоторые кусочки, которые исполняются как целое и называются программными фрагментами.



Другими словами, до компьютера ведет слишком длинная дорога, и что делается в компьютере, когда ваша программа находится на каком-то шаге в языковом тексте, никто вам сказать не сможет. Это и есть отчуждение программиста от исполнения программы. Ошибку, которая находится в неизвестном месте на неизвестном слое, отловить довольно трудно. Это, как известно, называется отладкой. И за сложность архитектуры, и за высокий языковый уровень и за удобства ОС, за все приходится платить усложнением отладки.

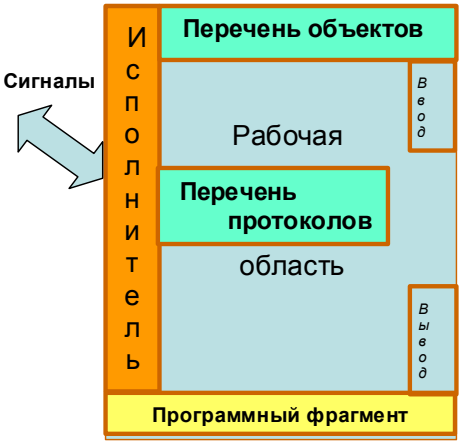
Больше всего описанная картина напоминает показанную иллюстрацию к известной с детства сказке Ганса-Христиана Андерсена. К сожалению, среди наших замечательных и прекрасных программисток настоящие принцессы встречаются чрезвычайно редко, а для всех остальных отладка продолжает оставаться весьма трудным делом.



Начиная с самой первой машины, все программы исполняются более или менее одинаково. Берется программный фрагмент, который состоит из предписаний, записанных в некотором линейном порядке. Среди этих предписаний есть указания, если надо нарушать этот линейный порядок внутри программного фрагмента. Про каждое предписание можно сказать, что для его реализации существует другой программный фрагмент, и т. д. Получается некоторый стек таких вызовов. Этот стек придуман в 1957 году, он был даже запатентован, но срок патента уже истек. Сделал это Фридрих Бауер.

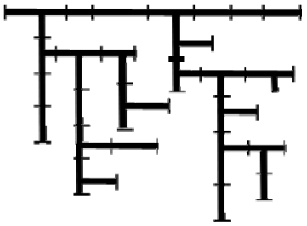
Чтобы программный фрагмент заработал, надо учесть очень многие вещи: систему команд, правильность расположения данных, к которым он обращается, и т. д., вплоть до того, что имел в виду программист, когда он писал эту программу, но в явном виде в нее не заложил. Ленинградский математик и программист Г. С. Цейтин называл это явление «призраками программирования».

Оказалось, все, что требуется, можно собрать следующим образом. Есть программный фрагмент, и есть операционная обстановка, в которой он выполняется. Есть активный элемент – Исполнитель, который может ходить по программному фрагменту, при этом он будет по очереди исполнять предписания. По очереди в том смысле, как это предписано структурой самого программного фрагмента. Для этого нам потребуется рабочая область памяти, где можно складывать промежуточные результаты. Там же можно выделить и области ввода и вывода. Хороший пример как это сделано аппаратно – кольцевой буфер архитектуры СПАРК, где выходная область соответствующего куска регистров является входной для следующего программного фрагмента. К этому надо добавить перечень объектов,



внешних по отношению к этому исполнению, и перечень внешних же подпрограмм и протоколов взаимодействия.

При этом мы описываем единичное исполнение (т. е. от начала и до конца каждого фрагмента). Поскольку программный фрагмент – это конкретная деятельность, то она обязательно заканчивается. Когда мы доходим до очередного предписания, мы запускаем реализующий это предписание программный фрагмент в другой операционной обстановке, которая может иметь свой перечень объектов, быть написана на другом языке и иметь исполнителя с другой системой команд. При этом движение по предписаниям порождает ***локальное*** время данного исполнения. Всякий раз, когда вы вызываете исполнение предписания, то оно, осуществляясь в другой операционной обстановке, порождает там свое время, разворачивающееся в ней самой, и к вашему времени не имеет никакого отношения. Поэтому все локальные имена оказываются многомерно упорядоченными, это позволяет легко переходить к параллельному исполнению.

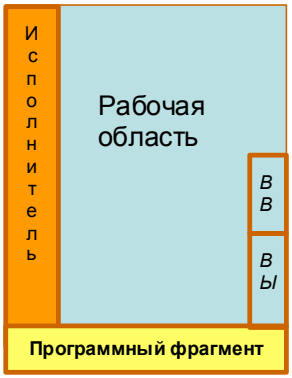


Многомерность локальных внутренних времен

Я подтверждаю еще раз тезис, что конкретная деятельность – это единичное исполнение программного фрагмента в заданной операционной обстановке, причем каждое предписание берет свои аргументы из рабочей области и помещает результат в рабочую область.

Операционная обстановка имеет форму буквы **Е**. Имеет смысл посмотреть некоторые частные формы, потому что им можно придать свою интересную интерпретацию.

Первая такая форма – L-форма. Это такая операционная обстановка, в которой нет внешних объектов. Это означает, что ей не разрешено оказывать эффект ни на какие существующие объекты, и во-вторых, в ней нет никаких протоколов, с помощью которых тоже можно испортить какие-нибудь другие объекты. Фактически предписания такой формы позволяют только переходы по программному фрагменту и вызов команды



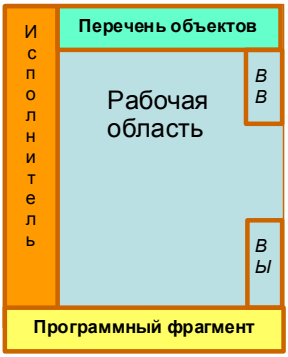
L-форма

исполнителя. Максимум, что может в результате произойти, это эффект в выходной части рабочей области, который и передается вызывающей операционной обстановке. Это означает, что стек операционных обстановок начинает сворачиваться обратно именно на программном фрагменте, который исполняется в L-форме операционной обстановки.

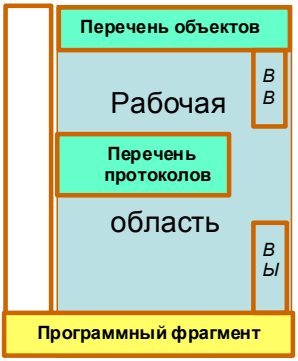
Вторая форма – C-форма. У нее нет протоколов, но есть некоторые вспомогательные объекты, с которыми надо работать. Мы не можем распространить свое влияние за пределы тех объектов, какие указаны в самой операционной обстановке.

Третья форма – особенная, в ней нет исполнителя. Для него предусмотрено место, но его еще нет. Ее вполне можно трактовать как результат работы системы программирования. Когда по программе определилась система программных фрагментов, то для каждого программного фрагмента система программирования может заранее заготовить требуемую для него обстановку с нужными перечнями объектов и протоколов и размером рабочей области.

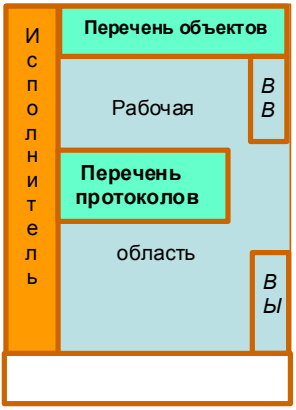
Можно сделать обстановку F-формы, подходящую сразу для нескольких фрагментов. Здесь есть запас объектов, с которыми можно работать. Есть библиотечные функции и протоколы, которые можно использовать. Есть исполнитель с хорошей системой команд. Есть достаточная рабочая область, только нет еще программного фрагмента. Такую обстановку я буду называть Вирт-машиной. Как только сюда подать программный фрагмент, исполнитель может начать делать свою работу. Результаты будут переданы через выводную область вызывающей операционной обстановки. Эффекты бу-



C-форма



Ξ-форма



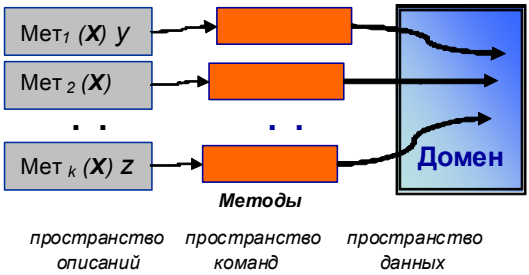
F-форма

дут зафиксированы изменениями в объектах, к которым разрешен доступ. На самом деле, такую F-форму можно подставить, как реализацию исполнителя.

Теперь о понятиях целостности и замкнутости. Операционная обстановка замкнута по управлению, поскольку на программный фрагмент управление поступает в самом начале, может как угодно двигаться в его пределах и обязательно выходит через его конец. Оно замкнуто по информации, потому что любое предписание программного фрагмента может пользоваться только данными из рабочей области или обращаться к объектам, которые для этого предусмотрены. А все, что происходит внутри самой рабочей области, схлопнется, как только окончится единичное исполнение. И, кроме того, имеет место замкнутость по времени, потому что в каждой обстановке порождается свое собственное локальное время.

В понятие ***операционной обстановки высокого уровня*** я фактически заложил всю ту информацию, которая должна окружать программный фрагмент, чтобы его можно было выполнять без сучка – без задоринки. Причем это удалось сделать таким образом, что когда вы переходите от обстановки к обстановке, вы не обязаны иметь одинаковых исполнителей, вы не обязаны писать программные фрагменты на одном языке, вы не обязаны ничего знать о структуре соответствующих объектов. Теперь самое время перейти к обсуждению объектов.

Я считаю, что после понятия подпрограммы, которое ввел Уилкс, объекты явились самым важным понятием. Почему? Потому что данные начали усложняться, а потом кто-то в Xerox-центре сообразил, что чем сложнее структура организации данных, тем больше специальных программных фрагментов надо использовать, чтобы корректно по ней двигаться. Вот эти фрагменты, адаптированные, чтобы корректно двигаться по сложной структуре объекта, и назвали методами, и решили, что их надо приписать к той структуре, с которой они будут работать. И никаким другом способом не разрешить пользоваться объектом. Это и назвали инкапсуляцией.



Логическую структуру объектов можно, исходя уже из сегодняшних соображений, показать таким образом. У каждого объек-

та есть свой собственный домен, где хранится его состояние, и где находятся его компоненты и подобъекты. Обращаться внутрь домена разрешено только программным фрагментам, которые являются методами этого объекта, а все остальные должны обращаться только в пространство команд к точкам входа в методы, используя формат описания для интерфейса соответствующего обращения к этому объекту.

При начальном появлении объектов было сделано некоторое количество очень неудачных, на мой взгляд, вещей. Объектная система Smalltalk была сделана на качественно более мощной рабочей станции, чем обычные локальные персональные машины, которые всеми использовались, а поэтому система могла быть достаточно эффективно реализована как интерпретирующая. А раз она была интерпретирующая, то она работала с объектами и с взаимодействиями объектов между собой, с учетом тех состояний, в которых эти объекты находились.

Поэтому там была задействована вот такая метафора: объект – это самостоятельная сущность, попроси его – он тебе все сделает. Это было проведено очень явно. И когда у нас появилась книга про Smalltalk-80, то еще неясно было, будем ли мы ее переводить, а если будем, то не я ли это буду делать, потому что это мне было страшно интересно, а с Алголом-68 я к тому времени уже развязался. И я говорил тогда Андрею Петровичу, что если я буду переводить эту книгу, то представляю себе, что слово «object» надо будет совершенно естественно переводить как «субъект» – целостную сущность, к которой можно обращаться и от которой можно что-нибудь получать.

В результате в объектно-ориентированных языках не очень явно делается упор на то, что объекты – это знаковые пассивные конструкции, которые сами ничего сделать не могут. Это просто такой договор, что когда придет исполнитель, то он будет работать только предписанным способом.

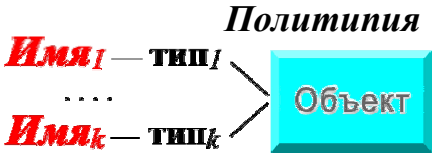
Далее, для облегчения описания объектов была придумана замечательная вещь, которая называлась наследование между классами объектов. Стоит напомнить соотношение между классами и объектами: объект – это единичный экземпляр, порожденный классом. Но между классами и объектами не было проведено, а в языках с плюсами и до сих пор не проводится четкого разграничения, и что класс – это информация по созданию объектов, информация о наследовании, и информация о том, что некоторые одинаковые обращения к объектам для разных типов объектов будут отрабатываться по-разному (полиморфизм). Не подчеркивается, что для эффективной реализации объектов вся эта «классовая» информация должна быть снята статически, т. е. использована до начала исполнения программы.

К тому моменту, когда объект создается в соответствии с системой наследования, определенной между классами, а не между объектами, надо хорошо себе представлять, что ни один объект ничего другому объекту не наследует. Наследование – это связи классов, говорящие о том, что если в некотором классе создается некий объект, то он получит в качестве своего типа определенный перечень методов. Во-первых, те, которые тут в классе описаны, а во-вторых, те, которые придут по наследованию. Из всех классов, которые для этого были предусмотрены. А в момент создания объекта этот список для него фиксируется, с предписанным полиморфизмом выбором того, что нужно подставить в качестве каждого метода.

Есть хороший простой пример полиморфизма. Если вы хотите поехать на Алтай, надо выехать на М52 и поехать на юг. Так вот, поехать на юг *прямо по* карте, то есть по объекту «карта», по компасу – это один метод. А поехать на юг *прямо по* дороге – другой. Потому что когда вы выедете под синий мост и поедете «прямо на юг» по компасу, то я могу точно сказать, в каком кювете вы окажетесь ровно через пару секунд. А оказывается, что ехать ***прямо по…***, можно и по кривой дороге. Это и есть тот самый полиморфизм, который здесь хорошо виден.

Кроме того, например, оператор создать объект класса «new» принадлежит классу, а вовсе не принадлежит объекту. Более того, можно даже показать, что это протокол, и он связан с некоторым подпространством, к которому объект отнесен.

Еще одного понятия вообще не было. Когда описывают, то говорят «объект имеет некоторый тип». Тип – это набор методов, которыми можно обращаться к этому объекту. Тип приписан объекту, но ведь обращаются к объекту все равно через его обозначение, то есть – через имя. И тип вполне можно *отнести к имени*. А тогда получится естественная ситуация, что если вы через это имя обращаетесь к объекту, то есть пытаетесь что-то сделать в его домене, то у вас этот объект выглядит как объект некоторого типа, а через другое имя вы можете на этот же самый домен посмотреть как на домен объекта другого типа. Это можно назвать политипией



Например, когда вы работаете с текстом с помощью текстового редактора, вы должны учитывать структуру текста, но когда вы после этого хотите передать этот текст куда-то, то вы его берете целиком, и он уже у вас перестает быть составным объектом, он упаковывается, вы эту упаковку и должны передать. Хороший пример можно найти в области семейных отношений: если у Коли есть сестра Катя и шурин Вася, тип *сестра* и тип *жена* для Кати, вообще-то, отличаются методами доступа.

Вот теперь я введу некий весьма важный принцип. Он называется принципом информационной замкнутости и требует, чтобы при обращении к объекту с помощью некоторого метода могло измениться только состояние в домене этого объекта и ни в каком другом объекте. Это довольно сильное требование, потому что, а что это значит, в каком это – никаком другом? Это аналогично тому, как понимать, когда на вопрос: «Вам воды с сиропом или без?» отвечают – «без». А без какого сиропа?

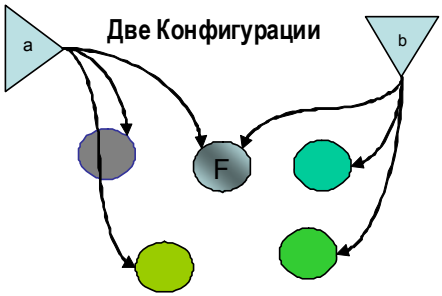
С другой стороны, это локализующее требование, которое обычно предъявляется в практике, и оно включает то, что никто, кроме метода объекта не может залезть в домен. Но это еще и означает, что когда вы исполняете метод, а метод – это программный фрагмент, то, значит, там есть операционная обстановка, в которой есть свой исполнитель, и там может быть вызов какого-нибудь другого программного фрагмента – так вот этот программный фрагмент должен быть только среди тех, которые отнесены к этому объекту. Вы не можете вызвать метод из другого объекта, потому что иначе у вас может измениться тот другой объект, а принцип информационной замкнутости это запрещает.

Нам, конечно, интересно работать не только с простыми объектами, но и с составными, у которых есть подобъекты. Если мы последим за сложившейся практикой, как мы работаем с составными объектами, – если нам нужно что-то сделать в подобъекте объекта, ну, например, в строке матрицы, то мы сначала получаем доступ к строке, а потом работаем с ней как с объектом типа вектор. Но, если мы захотим завести подобъекты в объекте, то принцип информационной замкнутости немедленно скажет нам, что подобъект объекта не может быть объектом и обратиться к нему как к объекту нельзя.

А теперь, представьте себе: у вас была куча деталей, т. е. объектов, а потом вы из них собрали видеокамеру. Видеокамера начинает работать, а детали как отдельные сущности перестали существовать. Они спрятались внутрь. И соответствующий подобъект может быть тоже достаточно сложным, например, строка матрицы - это вектор, у которого есть свои элементы. С этим надо что-то делать.

Проблема решается, если принять, что объект создается и находится не неизвестно где, а в каком-то подпространстве. Например, в качестве подпространства можно предъявить домен объекта-хозяина. Тогда, если мы возьмем матрицу, то у нас занимаемый ею кусок памяти – это ее домен, все строки, столбцы и элементы должны находиться внутри нее, их мы и будем относить как объекты к этому подпространству домена матрицы. А сама матрица лежит в каком-то другом блоке памяти, т. е. она является объектом другого подпространства, и коллизия исчезла.

Тогда ограничение будет уже таким: подобъект объекта не является объектом в том подпространстве, в котором объект является объектом. Здорово? Но нет, оказывается, что мы еще не достигли цели, потому что мы не можем одновременно объявить строки и столбцы матрицы объектами даже внутри домена матрицы. Потому что они пересекаются, и это нарушает принцип информационной замкнутости.



Оказывается, это просто потому, что у нас средств не хватает. А средство, которое нам нужно, – это умение группировать объекты так, чтобы они не теряли статуса своей объектности. Понятие, которое в данном случае нам нужно, – это конфигурация.

Вот я его и введу. Это такое множество объектов, при котором есть еще один объект, «голова конфигурации». Через голову обеспечивается навигация к любым составляющим объектам. Значит, конфигурация – это множество объектов, объединенных средствами навигации из головного объекта, эти объекты могут находиться в разных подпространствах, они могут быть связаны, например, указателями друг с другом, а могут быть несвязанными. Конфигурация сама приписана тому подпространству, где находится ее головной объект. И тип в конфигурации – это как раз набор операций навигации. Больше в ней ничего нет.

То, что объект предписан какой-либо конфигурации, не мешает ему быть предписанным другой конфигурации. И поэтому, если мы знаем, что никакие два объекта не могут иметь общих подобъектов, потому что их домены не пересекаются, то один и тот же объект вполне может входить в две конфигурации.

На примере из теории социальных ролей: каждый из нас является с одной стороны сотрудником института, входит в эту конфигурацию и, с другой стороны, является членом своей семьи, семьи своих детей, круга своих знакомых. Мы можем насчитать некоторое количество конфигураций, которым мы приписаны. И через которые нас можно доставать.

Может получиться такая ситуация: есть некоторый объект **F**, конфигурация **a** имеет доступ к нему, она считает, что это объект, а раз это объект, то он не может самостоятельно изменять свое состояние, он пассивен. Но она может обнаружить, что он меняет свое состояние, при этом она не обязана знать, что объект **F** входит в другую конфигурацию **b**, через которую это состояние изменялось. Тогда получается, что либо **F** не надо считать объектом, а надо считать субъектом, который может самостоятельно менять свое состояние, либо надо говорить о некоторой *наведенной* активности на этот объект.

Нужно различать два сорта конфигураций: те, к объектам которых нельзя добраться иначе, чем через голову конфигурации, назовем их *отдельностями*, и те, которые сцеплены с другими конфигурациями. И с ними работать надо по-разному. При этом из матрицы нельзя убрать строку, а из конфигурации, из списка, можно убрать и можно добавить объект-элемент. Такие динамические типы данных, как списки, конечно, являются конфигурациями объектов.

Вернемся к описанию подобъектов матрицы, сформулировав следующее: мы хотим, чтобы в матрице были как подобъекты и столбцы, и строки, а для метода прогонки еще были и диагонали – все они, конечно, пересекаются. Это легко доказываемое утверждение, и каждый может его проделать как легкое домашнее упражнение, т. е. любой подобъект объекта может быть представлен как конфигурация других объектов, которые являются подобъектами во внутреннем подпространстве домена хозяина.

В данном случае все подобъекты матрицы, кроме элементов, будут конфигурациями. Значит, внутренними объектами можно считать только элементы, а уже из них составлять строки, столбцы, диагонали, миноры, разреженные миноры, и вообще все, что вам захочется. Если вспомнить описание семантики массива в Алголе-68, то там это хорошим английским языком и описано.

Я многократно упоминал слова «значение» и «состояние». Теперь я хочу продемонстрировать важное различие между ними. Значения появились от функций. Функция по определению перерабатывает значения своих аргументов в значения своего результата. А состояние – это то, что сохраняется в объекте, если его не трогать. Но когда вы хотите записать некоторое

значение в объект с помощью оператора присваивания, происходит некоторое невидимое для вас преобразование, которое состоит в том, что значение превращается в состояние. После чего – дальше работают аксиомы – если правильным образом прочитать это состояние, то на выходе операции чтения получится то же самое значение.

Не мешает сделать простую вещь: вы записываете в регистр комплексное число, а в другой регистр – другое комплексное число, при этом вы ничего не знаете о том, как оно устроено, следовательно, регистр был простым объектом, к которому можно применять соответствующий набор операций. А вот для того чтобы описать программные фрагменты, реализующие эти операции, вам придется прочитать из этого регистра структуру с двумя полями, а именно с полем вещественное и с полем мнимое, рассматривая этот же самый регистр как объект составного типа. Читать вы оттуда будете соответствующими методами поля структуры. Если вы захотите, например, складывать вещественные числа, вы должны писать в регистр, как в простой вещественный тип, а читать как из составного, представляя его же в виде структуры с четырьмя полями: порядок, мантисса и два знака.

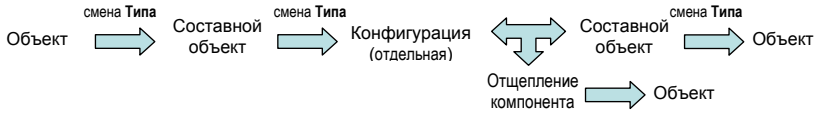
Это и есть применение той самой политипии, о которой я говорил, она все время работает. В то же время, значение вырабатывается функцией, выдается и его можно один раз использовать, но этот один раз может быть: либо вы передаете это значение, полученное в результате как аргумент по суперпозиции функции, если у вас записана составная функция. Либо вы можете спрятать его в некоторый объект (в качестве состояния). Если вы его спрятали, то вы сможете потом читать его столько раз, сколько вам необходимо. Если вы его передали по связи суперпозиции, то только ровно один раз, и поэтому значения однократны и не имеют внутренней структуры, и, следовательно, они вообще не копируемы, потому что то, что мы сохраняем, это не значение, а состояние объекта.

Самым показательным примером является музыка, она прозвучала и все. Ее, конечно можно записать, но дело в том, что запись – это не то же самое, что само звучание. Это уже некоторая статика, например физическая запись, либо если у нас есть исполнитель, который в состоянии превратить эту запись в текстовую запись, – ноты, а это называется транскрибированием, но тогда уже начинаются работы через состояние.

Вот задачка. Попробуйте на объектно-ориентированном языке программирования описать такую деятельность: вы хотите разжечь камин, у вас есть полено и вам надо нащипать лучины. Ключевым здесь является, конечно, вопрос, а где в полене лучинки? Или у вас есть прибор и его надо разобрать. Сейчас я покажу, как любое изделие разбирается. Есть целост-

ный объект – простой, как кирпич, рассмотрим его. Если я его беру, чтобы его чинить, я должен начать на него глядеть как на составной объект, значит, я должен выделить в нем имеющиеся существенные части, например, винты. Вот он – составной, ну и что? А то, что потом я должен буду сменить тип еще раз и начать рассматривать этот составной объект как отдельную конфигурацию. Как только я рассмотрю его как конфигурацию, то уже могу сказать, что она собрана из разных объектов. Вот и я начну эти объекты отделять.

Спецификация разборки изделия



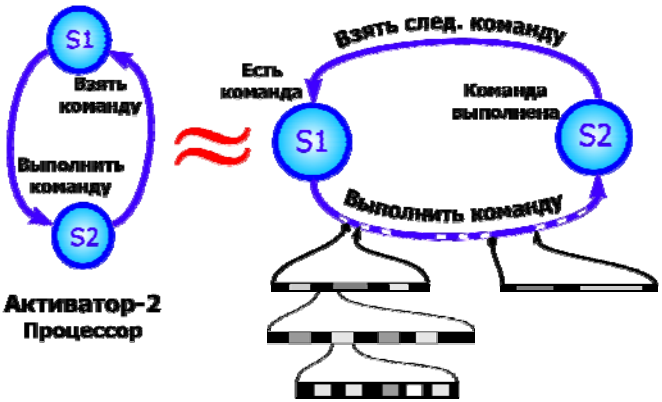
А для исходной задачи я начну рассматривать полено, как отдельную конфигурацию лучинок. Далее, как вы поняли, все проще простого: отщипну лучинку от полена, то, что останется, – это объект типа чурбак. А то, что я отщепил, я буду называть объектом типа лучинка. Если мне понадобится и дальше расщеплять, я буду рассматривать чурбак как конфигурацию, отщиплю еще раз, и когда нащипал нужное количество лучинок, то остаточный чурбак положу в запас. Все. Вот вам знаковая схема-спецификация разборки какого-либо изделия по информационно замкнутым конструкциям.

И если мы хотим, например, переслать фильм по электронной почте или через ftp, то что происходит? Это кино лежит в каком-то файле типа avi размером в пару гигабайт. Для того чтобы этот файл передать по линии связи, его надо рассмотреть как объект совсем другого вида, нарезать его на маленькие кусочки, которые воспримет система приема-передачи, а именно фреймы для передачи по линии. А потом на другом конце это все должны собрать, превратить это все опять в файл avi, и если все прошло хорошо, то можно наслаждаться кино, которое вам прислали в подарок.

Вся работа программы делается исполнителем над пассивными структурами. И я хочу обратить внимание на то, что, какие бы сложные алгоритмы ни применялись, и какие бы объекты вы ни нагородили, и какие бы вы ни придумали методы сами они «с места не сдвинутся». Пока не придет активный исполнитель и не будет ходить по этим методам в операционных обстановках. Активность, которая нам для этого нужна, невозможно вывес-

ти логическими преобразованиями из пассивных объектов, ее придется вводить аксиоматически как новую сущность.

Эта новая сущность в самом простом случае может быть устроена так: она должна иметь два состояния и независимо от того, обращаются к ней или нет, переходить из одного состояния в другое, вот такой «тик-так», который каждый раз порождает шаг времени. То же самое в операционной обстановке, когда исполняется программный фрагмент, исполнение каждого предписания дает нам один такт времени. Почему один? А потому, что для единичного исполнения предписание имеет размерность точки, т. е. нулевую, а любая единица измерения в нулевой степени дает единицу, в смысле «штука».



Если у нас есть такой активатор, то мы можем обратиться в фирму «Интел» и она нам расскажет, как построить процессор, который тоже можно будет считать имеющим два состояния: S1 «взять команду», и S2 «выполнить команду», а выполнив команду пойти и взять следующую команду.

Теперь я перерисую все это в горизонтальном виде и сменю обозначения. Нормальный математический ход:

S1 – есть команда,

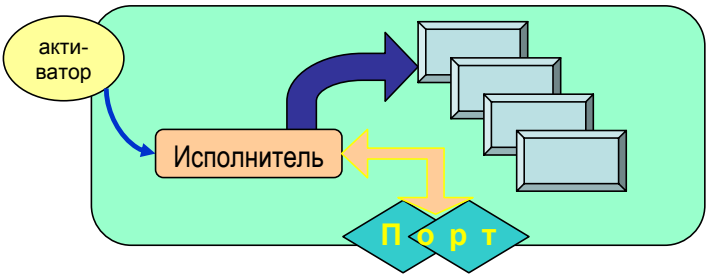
S2 – команда выполнена.

Тогда стрелка перехода из S2 в S1 – это взять следующую команду, а стрелка из S1 в S2 – выполнить команду.

Посмотрите, что получилось: каждая стрелка имеет начало и конец, а значит, она начинается и заканчивается, следовательно, может трактоваться как конкретная деятельность. Чтобы взять команду, на самом деле, потребуется выполнить последовательность шагов: надо обратиться по некото-

рому адресу в память прочитать состояние этой ячейки памяти как значение и передать его по шине в регистр команд. Затем сменить тип регистра команд и разобраться в его состоянии по частям, чтобы понять, что в этом регистре команд сейчас находится, запустить соответствующие регистровые передачи и так далее, и так далее.

Поскольку у нас есть некоторый отрезок, который разбивается по частям на шаги, то каждый шаг можно назвать предписанием. Если это предписание – команда исполнителя, пусть исполнитель его и делает, а иначе есть программный фрагмент-реализатор этого предписания. Запустим его единичное исполнение в нужной операционной обстановке, ну и так далее, все уже ранее неоднократно рассказывалось.



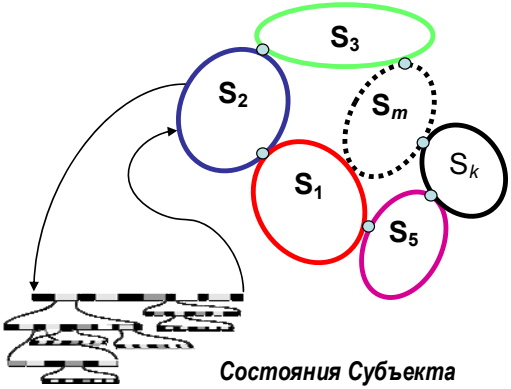
Это нам дает возможность представить, как устроен субъект, независимо от того, что мы не знаем, как он устроен. Исходим из того, что есть элементарный активатор, и потом с его помощью построим процессор. Ибо, что такое процессор? Это некоторый набор блоков, каждый из которых есть некоторый объект, активатор обеспечивает хождение по ним. У нас есть процессор-исполнитель, и можно написать сложную программу и задать требуемую структуру данных. Затем процессор начнет исполнение и осуществлять предписанное поведение и функционирование. Вот мы и сделали, то ради чего все затевалось, т. е. выполнили соответствующую цель в некоторой искусственной системе – построили субъект.

Когда мы знаем, как построить субъект, тут можно строить некоторые примеры, и если «переходить на личности», то понятно, что с каждым из нас надо связать не один субъект, а несколько. Мы-то сейчас разговариваем только о той части индивида, которая занимается мыслительной деятельностью. Но ведь в середине нас есть еще субъект, который называется сердце и который стучит независимо от того, можем ли мы в данный момент мыслить или нет.

«Стоять» это тоже деятельность, кто не верит, может встать и постоять немножко на одной ножке, пока не почувствует, особенно если у нее туфли

на высоком каблуке, что это деятельность. Лежать может даже бревно, а вот стоять – это активная деятельность, от этого даже устают. Между прочим, часовые, которые стоят час в карауле в президентском полку, говорят, что это изматывает безумно. Я вот постоял целый час и устал, вот сел и стал чувствовать себя лучше. Такие конструкции можно наращивать, но, конечно при условии, что субъекты информационно замкнуты, что, по-видимому, и имеет быть, а взаимодействуют путем обмена сообщениями.

Например, каждый из нас знает, что он думает, и предполагает, что те, с кем общается, тоже думают. Предполагает, хотя бы из вежливости или для своего собственного удобства. Но, слава Богу, когда я говорю, то никто не знает, что я при этом думаю, и не узнают никогда, и я не узнаю что вы про меня в этот момент думаете и, может, это тоже хорошо.



Вообще говоря, состояние субъекта очень хорошо описывать вот так – как систему взаимосвязанных циклов. Другими словами, состоянием для субъекта является элементарный цикл, по которому он в этом состояния крутится. Например, пусть s_1 – это цикл, который называется «бездействие системы». Потом приходит какой-то сигнал и система переключается на другой цикл, когда возникает еще какая-то ситуация она переходит на третий цикл, затем переходит на еще какой-то другой цикл, вот такой огромный путь, вот такой аттрактор. Ясно, что таким путем можно построить все, что угодно. Конструкцию любой сложности, даже хаотический аттрактор.

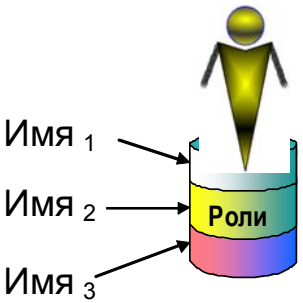
При этом любой кусочек каждого цикла можно рассматривать как имеющий начало и конец, а значит, он может быть корневым программным фрагментом, а значит, на него можно запустить стек исполнения его соответствующих предписаний.

Дело только в том, что если по отношению к программам и программным фрагментам мы говорим, что их каждое единичное исполнение начинается и заканчивается, то все программисты прекрасно знают, что если программа зациклилась, значит, она перестала выполнять свою работу.

Из этого стоит сделать четкий вывод, что операционная система программой не является, а компьютер программе не равносилен. И компьютер, и ОС как раз тем и занимаются, что они бегают по циклу. И если вы в этой системе циклов не предусмотрите в каком-нибудь месте предписания, посмотреть наружу, «а не нужен ли я кому-нибудь?», то вы получите аутичного субъекта, до которого не достучитесь. Он не будет реагировать на окружающую обстановку если в нем, т. е. в этой «не программе» этого не запрограммировано.

Имеет смысл сопоставить, на самом-то деле, все наши объектные рассмотрения, и все наши описания деятельности как отражение того, что, в естественной информационной деятельности субъекты информационно непроницаемы. И, слава Богу, никто не может залезть и посмотреть, что думает другой человек, его надо спросить, и он, может быть, что-то и ответит. Но это не значит, что он именно так и думает, – это он хочет, что бы мы думали, что он так думает. (!)

В знаковом мире мы стали это моделировать, заведя понятие объекта. Мы сказали, что это мы будем соблюдать условия инкапсуляции и информационной замкнутости. Несмотря то, что в домен *можно* залезть и делать там что угодно, мы этого делать не будем: в целом получится слишком сложно и ненадежно, поэтому ограничим себя и будет нам счастье.



Политипия
для Субъектов

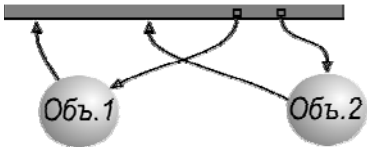
Все, что связано с типами и с политипией, прекрасно можно перенести на субъектов, собственно говоря, я это и сделал, когда сказал, что каждый из нас выступает в разных ролях. В роли сотрудника, в роли друга, в роли красавицы, на которую все оглядываются, когда она идет по улице, в роли матери или супруги. И та сущность, которая, вроде бы, и есть Личность, она помещена в систему стаканов, и каждый стакан – отдельная роль.

Однако заметим, что множество разнообразных, сколь угодно функционально богатых объектов само по себе является недостаточным. Действительно, все эти объекты фактически отвечают за то, что *можно* будет с их

помощью сделать. Но из них не выводится, что *нужно* сделать для решения какой-либо конкретной задачи.

Для этого потребуются специально нацеленный на эту задачу внешний по отношению к этому множеству объектов программный фрагмент и единичное исполнение этого фрагмента как корневого, опирающееся на выводимую из него правильную совокупность единичных исполнений его предписаний в подходящих обстановках, обеспечивающих доступ к требуемому подмножеству объектов.

Базисный протокол



Рассмотрим следующую ситуацию: есть два объекта (или субъекта), каждый из которых находится в некотором состоянии, а я хочу, чтобы они своими состояниями обменялись. Что есть у объектов? Есть домены, и есть методы. Из принципа информационной замкнутости следует, что ни один из методов первого объекта ничего не может узнать о состоянии второго, а методы второго объекта ничего не могут поделать с первым объектом. Значит, как и в предыдущем рассуждении, нам нужен независимый программный фрагмент, который я буду называть протоколом, а когда мы этот протокол запустим, то он будет исполняться в собственной операционной обстановке. Он сможет обратиться к объекту 1 и спросить его: «Скажи, как твое состояние?». Если такой метод предусмотрен в этом типе объектов, то тот ответит, например, «спасибо, хорошо». Протокол запишет ответ в рабочую область своей операционной обстановки. Затем протокол может спросить у объекта 2 «как поживаешь?» а тот, например, ответит «так себе», и этот ответ тоже записывается протоколом. Далее протокол зашлет узнанное от объекта 1 состояние объекту 2, и наоборот. Взаимодействие произошло, объекты обменялись состояниями, но для обеспечения этого потребовался внешний программный фрагмент – базовый протокол.

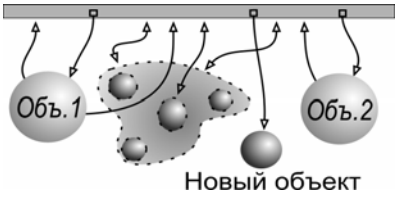
Чистый протокол

Но это, вообще-то, еще не все, даже если у нас взаимодействуют только два объекта. Например, чтобы позвонить по телефону, исполнитель будет в операторе связи. Во взаимодействие будут также вовлечены некоторые другие объекты из оборудования станции, о которых, кстати, абонентам, ничего не нужно знать, хотя во время работы протокола они меняют свое

состояние, обеспечивая соединение одного абонента со вторым, связь устанавливается и начинается разговор, два телефона обмениваются информацией.

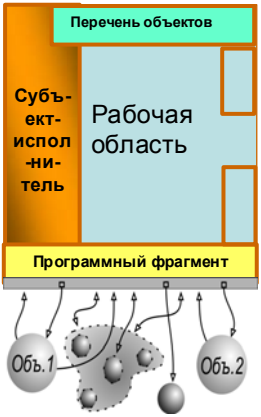
Принцип информационной замкнутости может быть расширен на протоколы следующим образом: по завершении взаимодействия могут измениться состояния только пользовавшихся протоколом объектов (субъектов), а все остальные, временно занятые объекты должны освободиться и остаться неизменными. Тогда протокол называется чистым.

«Экологически нечистый» протокол



Мобильные телефоны работают совсем по-другому – экологически не чисто, а именно всякий раз, когда вы разговариваете, среда засоряется новым объектом – счетом за ваш разговор.

Таким образом, протоколы бывают чистые и нечистые, их надо разделить, и с нечистыми протоколами надо держать ухо востро. Хорошо, что большинство протоколов, которые обеспечивают взаимодействие, сводятся к одному из более простых типов – базовому или чистому.



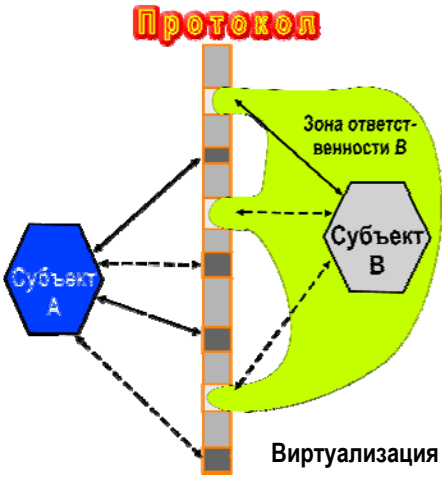
Ведь если мы пользуемся протоколом, то хозяином взаимодействия является протокол, и он диктует, что надо делать и в каком порядке.

Если мы пользуемся протоколом, то мы должны выполнять предписания, которые приходят из этого протокола. Если мы их не выполняем, то исполнение протокола рвется. Поскольку протокол – это программный фрагмент, а исполнение программного фрагмента порождает массу других программных фрагментов, реализующих его предписания, то протокол многослоен. Когда он на каком-то слое рвется, то, как правило, есть еще много «подстилающих» слоев, которые помогают его поддержать.

Что мы имеем в нормальном человеческом общении? Массу возможностей, сохранить взаимодействие. «Извините, я Вас не расслышал», «повторите, пожалуйста», и так далее. До ситуации полного разрыва, как это можно видеть, там, где это явно выражено, – в дипломатической практике, там ведь даже протокольные отделы есть, дело доходит не скоро.

То же самое мы видим и в протоколах передачи данных, в компьютерах и связи, особенно там, где помех много, – там используются весьма изощренные протоколы.

Если вы пользуетесь протоколом, вы должны его соблюдать, протокол – это корневой протокол, от него естественно идут всевозможные слои, иногда мы проваливаемся, восстанавливаемся. Это составляет нашу культуру взаимодействия. Есть субъект-исполнитель, который обеспечивает коммуникацию по максимуму. В пределе субъективизации протоколов получится в частности, как говорят арабы, «иншалла», что означает, что «аллах, всеведущий и всемогущий, соизволил захотеть и позволил нам…», например, прочитать сегодняшнюю Ершовскую лекцию.



Виртуализация протокола

И когда я напишу это слово, Его, с прописной буквы, то станет ясным, что я имею в виду, что то самое понятие Бога, которое, собственно и создано людьми для их удобства и для поддержания целостности понимания мира, у нас и возникает в результате возведения субъективизации протоколов в абсолют. Но на самом деле нам не обязательно протоколы субъективировать, мы можем считать, что протокол виртуален и что два субъекта его соблюдают, тем, что каждый их них в нужном порядке делает свою часть работы.

Четыре принципа, которые всегда исполняются в программе, кроме последнего, и которые я призываю вас тоже исполнять. Это можно делать на любом языке, при любых средствах программирования, просто их надо осторожно использовать. Я называю эти принципы «***Священные коровы программирования***»:

1. **Всякое единичное исполнение завершается, т. е. никакая часть программы не зацикливается**, если мы этого не предположим, мы не можем сложную работу разложить на простые.
2. **Корректность связей должна обеспечиваться системой**, висячие указатели не допустимы; за связями, которые прокладывают-

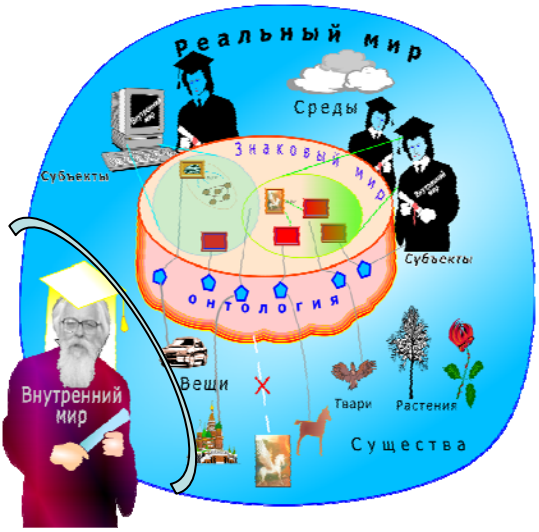
ся динамически, кто-то должен присматривать, чтобы они не испортились.

3. **Нельзя одновременно из разных источников вносить изменения в одно и то же место –** это общеизвестно.

И эти три принципа, ни в какой работающей программе реально не нарушаются – это принципы абсолютного запрета, и вот четвертый принцип,

4. **Не пользуйтесь информационно не замкнутыми объектами и обстановками** и у вас не будет побочных эффектов, т. е. будет делаться то, что вы хотите, и не будет происходить то, о чем вы не знаете.

А это значит, что у вас не будет отладки, кроме отладки синтаксической правильности, – вот к чему это приведет.



В заключение представление о том, как информационно устроен мир. Каждый субъект приходит и говорит: «Мир начинается с меня», потому что если бы меня не было, то и мира бы не было, а в том, что мир реален, в этом каждый спокойно может убедиться, Кстати, для этого в русском языке есть хорошее слово «наткнуться» – наткнуться на чужую волю, наткнуться на стену.

В мире есть вещи, которые не меняются, если их не трогать, есть существа, которые меняются сами по себе, и среди них – субъекты. В 1950-е годы появился еще один класс субъектов, который назван компьютер, А еще в мире есть среды. Если вещи занимают отдельное место в пространстве, то среды могут сосуществовать в пространстве, и если взять учебник по уравнениям математической физики, то можно хорошо увидеть, как там выводятся уравнения состояния среды.

Поступаем следующим образом: возьмем, и выделим в среде некоторый параллелепипед. Затем напишем для него условия равновесия, граничные условия непрерывности и, например, баланса потоков. А потом стянем его

к нулю и получим дифференциальное уравнение, описывающее нам поле. Таким образом, мы всегда работаем через объекты.

В отличие от того, как у нас в последнее время стало применяться слово «информационная технология», я утверждаю еще с 1985 года, что технология – это программа для пользователя, это то, что пользователь должен сделать сам. А остальное сделается машинами. С этой точки зрения искусственная система, при обычном взаимодействии с нею, когда вы ее не должны переделывать и когда вы не можете добраться до ее программы, ведет себя как естественная.

Это все равно, что наливать чай из чайника. Напишите программу «как налить воду в чашку»: надо взять чашку, взять чайник с водой и надо наклонить чайник над чашкой. А что дальше? Какой будет следующий шаг в программе? А дальше нужно будет следить за уровнем воды в чашке, потому что вода из чайника будет литься сама. И будет вот такое сочетание естественного процесса, «вода дырочку найдет», и искусственного программируемого слежения за уровнем воды, чтобы она не переливалась через край чашки – они естественным образом сосуществуют в деятельности.

И наконец я хочу привести пример ситуаций, когда соблюдение информационной замкнутости является ***жизненно*** необходимым. В фильме «Семнадцать мгновений весны» фрау Заурих дружит с инженером Больценом, и она считает, что это очень милый хороший человек: заботится о ней, водит ее гулять, он ее подкармливает ветчиной, беспокоится, когда она нездорова. У фрау Заурих убили на войне сына, и она ни за что не стала бы водиться с фашистом. Но она не догадывается, что Больцен на самом деле – группенфюрер СС Макс Отто Штирлиц. Почему? Потому что инженер Больцен информационно замкнут.

Но ведь группенфюрер СС Штирлиц это, как известно «только узкому кругу в руководстве советской разведки», а заодно и всем нам – советский разведчик Максим Исаев. А тот, кто смотрел фильм «Бриллианты для диктатуры пролетариата», знает, что Максим Исаев – это сын старого профессора и зовут его, вообще говоря, – Всеволод Владимиров.

Сейчас пасха. Вот вам пасхальное яичко в качестве подарка, а вот еще и пара старых лозунгов.

Каждый системный программист мечтает быть Пользователем. (лето 1972 г.)	*Машина поручает человеку работу, которую тот не смог поручить машине.* (осень 1984 г.)

Спасибо за внимание.

Я готов ответить как за свои слова, так и на ваши вопросы.

ВОПРОСЫ:

А. М. Казанцев: Андрей Александрович, вы несколько раз упоминали выражение «языки высокого уровня», а уровень высоты когда закончится? Что значит предельно высокий уровень? Это первый вопрос.

Второй вопрос. Вы когда-то бросили фразу, я ее слышал уже от других, от Поттосина, но он на вас ссылался: «Программисты, совершенствуя трансляторы, уничтожат самих себя». Я не ручаюсь за точность.

Ответ: Да, была такая максима: «Цель программирования, как науки есть уничтожение программистов, как класса».

А. М. Казанцев: А каким образом, если на содержательном уровне, т. е. конструктивном, это будет уничтожено? Что должен будет сделать программист?

Ответ: На первый вопрос: уровень языка – понятие относительное. Потому что если вы создадите хороший программно-аппаратный комплекс, который, имея некоторую систему будущей машины, изображает из себя другую машину, так же как Максим Исаев изображал из себя Штирлица, и причем это хорошо защищено, то у вас получается вирт-машина. А эти вирт-машины могут быть вложены. Но когда вы проектируете сложную систему, вы что делаете? Вы сворачиваете все под адекватную вирт-машину. Подбираете адекватный набор объектов для предметной области и подбираете для них методы. Вы пишете структуру классов, которая обеспечивает упрощение через наследование, описания этих объектов.

У вас получается целый набор объектов и типов, У них есть методы. Кто вам мешает сделать вирт-машину, у которой командами будут являться методы используемых объектов? Или соответствующих групп объектов? Никто не мешает! Кто вам мешает построить вирт-машину, командами которой будут протоколы? Теперь смотрите, что у вас есть. У вас есть сово-

купность объектов и способов обращений к ним, которые являются командами для некоторой машины. Вся сложность заложена внутри нее.

А что вы ей говорите? А вы ей говорите, как джинну: «Построй мне дворец». Она говорит: «Будет исполнено». Вы получите тот дворец, который джинн умеет строить и считает нужным строить, а если вы не хотите, а хотите чего-то конкретного, то у вас получится та ситуация, которая возникает когда вы в F-ке той самой. У вас есть все возможности? А приходит программный фрагмент и говорит, а мне надо только вот это. И путается у вас под ногами. Значит, если вы знаете, чего вы хотите, то вы должны дать спецификацию. И на самом деле, уничтожение программирования состоит в том, что переходим от программ к спецификациям. Когда вы вызываете джинна и говорите, построй мне дворец, то вы больше ничего не говорите. Но вы, конечно, можете дать ему спецификацию: вот тут башню, вот тут спальню, а вот тут бассейн. И он не будет против, но если вы попробуете джинну сказать что ты должен это делать вот так, а это – так, то он вас пошлет к царю Соломону или еще куда подальше.

А. М. Казанцев: Андрей, можно перебью. Не было компьютеров. Люди же между собой общались, и программистов тогда не было. Они понимали друг друга. Когда появились компьютеры, то математик, ставивший задачу программисту, описывал ее на русском языке, на естественном языке. Спрашивается, почему это нельзя сделать по отношению к компьютерам?

Ответ: А почему нельзя? По-моему так совершенно спокойно. Дело в том, что может оказаться, что это экономически нецелесообразно на данном этапе. Или что-нибудь в этом духе. Или что, вообще, никто, так сказать, не подходил.

Это не принципиальный вопрос, Саша, и поэтому на него не интересно отвечать. Уровень общения с компьютером можно повышать до ситуации, когда машина начинает понимать тексты на естественном языке так же, как когда человек читает книгу, читает текст и понимает его. Он на самом деле, осуществляет ту же деятельность, которую предусмотрел автор. Или, на нее похожую. Потому что, как известно, смысл заложен не в книге, которую вы читаете, а в самом читателе, который ее читает. Причем каждый, как говорит великий могучий правдивый и свободный русский язык, понимает в меру своей испорченности. Так что я думаю, что я ответил вашу группу вопросов. Еще есть вопросы?

М. А. Бульонков (ИСИ СО РАН): В начале Вы сказали, что сейчас многие проекты начинают делать с нуля. Я вообще себе сейчас с трудом пред-

ставляю, чтобы где-то кто-то подошел к голой машине и начал что-то делать. Сейчас, даже когда действительно создается новая машина, уже имеется такой багаж, который позволяет учесть готовое. Не бывает сейчас работ с нуля.

Ответ: Тем не менее, во многих программистских коллективах, начиная новую систему, вместо того, чтобы собрать то, что уже есть, и приспособить его к себе, на самом деле, начинают делать с нуля. Другое дело, что языки проэволюционировали таким образом, что, если вы рассмотрите какой-нибудь общеизвестный модный и хорошо используемый язык типа C++ и представите себе такую полку с языком, то сбоку будет стоять такая узенькая книжечка – описание C++. А дальше будут стоять толстые тома библиотек для работы с математикой, библиотек для работы с процессорами, с проектированием, со статистикой и для чего-то еще.

И вы, конечно, можете набирать эти вещи, я только думаю, что тут проблема состоит в том, что собирать что-либо из частей можно при условии, что от этой сборки свойства частей не меняются и подключение некоторой новой части не портит то, что было раньше. Если соблюдается информационная замкнутость, а я собственно, ради этого и приводил соответствующий пример. Если вы к телевизору поднесете магнит, близко к экрану, то у вас испортится изображение.

М. А. Бульонков: Я понимаю, но у меня вопрос был не про то. Я не говорю того, что знание сосредоточено в языке C++, не пытаюсь утверждать, что это какой-то хороший язык. Я говорю, что та самая операционная обстановка, за которую вы пропагандируете, она далеко выходит за рамки языка в обычной трактовке.

Ответ: Я это же, только что, тебе ответил. Именно с этим я и согласен. Ценность языка определяется вовсе не языком, а совокупностью литературы, которая на этом языке написана. Если вы хотите пользоваться всем этим накопленным. А если еще и на другом языке, так вообще хорошо.

М. А. Бульонков: С одной стороны, это действительно так. Но если посмотреть на производительность вычислительных машин с точки зрения того, что они делают для человека, то она увеличилась совершенно непропорционально тому, насколько увеличилась декларируемая мощь. Тексты мы продолжаем набирать с той же скоростью, что и раньше. Опечаток столько же.

Но вопрос не про это. Ваша картинка, которую вы показали в конце, виртуальные машины, вообще говоря, не позволяет говорить о том, что нас

может в принципе интересовать та машина, которая находится в самом низу. И я в этом смысле прекрасно знаю, что происходит в терминах той виртуальной машины, для которой я составляю программы. Но мне никто не мешает, и слава Богу, что я не знаю, что там происходит на уровне кэша процессора и переключения.

Ответ: А в чем вопрос? Почему Вы говорите, не так? Я же говорю, точно то же самое.

М. А. Бульонков: Правильно! Но вы предлагали, как мне показалось, некоторое решение, которое позволяет контролировать.

Ответ: Нет! Я предлагал совсем другое. Я считаю хорошим путем, чтобы для данной проблемной области подобрать адекватную систему объектов, после чего собрать виртуальную машину, командами которой будут обращения к этим объектам. После чего забыть абсолютно про все, что меня совсем не касается. В результате протоколы взаимодействия, которые я буду на самом деле писать, ведь кроме протоколов больше ничего и не надо делать. Так они будут ассемблерного уровня по отношению к этой машине высокого уровня. И значит, они для меня будут короткие, прозрачные и простые. Вот и этого я и добиваюсь. А если они друг на друга, не будут влиять в силу информационной замкнутости, то я ровно этого и хочу.

ЕРШОВСКИЕ ЛЕКЦИИ

(Памяти академика А.П. Ершова)

Рукопись поступила в редакцию 26.05.09
Редактор И.А. Крайнева

Подписано в печать 03.07.09
Формат бумаги 60 × 84 1/16 Объем 5.1 уч.-изд.л., 5.6 п.л.
Тираж 300 экз.

Центр оперативной печати «Оригинал 2»
г.Бердск, ул. О. Кошевого, 6, оф. 2, тел. (383-41) 2-12-42