ОБ ОПЫТЕ ЕЗДЫ НА ДВУХ КОНЯХ В ПРОГРАММИРОВАНИИ

А.Буда

София, Болгария

Продолжая "религиозную" программу нашего симпозиума, я бы тоже хотел сказать о том, во что я верю, а не о том, что я знаю.

Во-первых, я верю, что наш бог - Аль-Хорезми, - сидя в своем заоблачном раю и освободясь от мирской суеты, узнал о природе алгоритмов значительно больше, чем сейчас знаем мы. И наш вопрос: "Что же делать, столкнувшись с алгоритмически неразрешимой проблемой?"—мы могли бы адресовать ему - Аль-Хорезми. Но по этому поводу я хотел бы напомнить один старый анекдот.

Бедный человек долго молился богу, чтобы он помог ему выиграть крупную сумму в лотерею. Наконец бог откликнулся на молитвы и сказал: "Хорошо, я помогу тебе, молись сильнее". Человек обрадовался, стал еще больше молиться и ждать выигрыша. Но прошло несколько тиражей, а выигрыш все не выпадает. Тогда человек опять призвал бога и спросил, почему он не сдержал своего обещания. На что бог ответил смиренно: "Поверь, я очень стараюсь, но помоги и ты мне - купи хотя бы один лотерейный билет..."

Наблюдается странное явление. Несколько десятилетий тому назад, когда были обнаружены первые примеры алгоритмически неразрешимых проблем, феномен неразрешимости казался таким необычным, что математики с трудом им поверили. А в наши дни, у человека, оказавшегося в бурном потоке работ по теоретическому программированию, может создасться впечатление, что алгоритмическая проблема, возникающая (как нередко отмечают авторы работ) в практике программирования, гораздо чаще оказывается неразрешимой, чем разрешимой. Так действительно ли эти проблемы навеяны практикой программирования? Мне кажется, что во многих случаях это не так. Мы имеем склонность рассмат-

ривать внутренние проблемы математики, которые отражают не историю развития вычислительного дела, а скорее историю развития теории вычислимых функций.

Десять лет назад я стал участвовать в создании крупной системы автоматизации программирования (150К автокод-операторов, 50 человеко-лет) и почти одновременно начал работать над диссертацией по проблемам разрешимости в теории схем программ. В обоих случаях моим научным руководителем был профессор А.П.Ершов. Он сказал мне, что я буду ехать на двух конях одной науки - программирования. Конь теоретического программирования будет помогать коню системного программирования и обратно. Таким образом, я буду ехать вдвое быстрее. Однако уже через год-два я скорее почувствовал себя сидящим между двумя стульями, которые все больше расползались. Мы сделали систему программирования, я защитил диссертацию, но все это время меня не оставляло ощущение, что я одновременно занимаюсь двумя различными делами. Сейчас мне все труднее запрягать своих коней в одну упряжку - они пасутся на совершенно разных лугах. И все-таки кажется, что опыт езды сразу на двух конях помог мне лучше почувствовать дистанцию между теорией и практикой программирования и выработать для себя представление (скорее интуитивное, чем точное) о практичной вычислительной модели, то есть о такой абстрактной модели вычислений, в рамках которой можно ставить и эффективно решать достаточно разнообразные задачи системного программирования.

Под вычислительной моделью я буду понимать математическое устройство, для которого строго определены следующие понятия:

- структура данных (структура обрабатываемой информации);
- оператор (единичный акт обработки данных);
- управляющая структура (отношения типа "предшественник-последователь" на множестве операторов);
- вычисление и его результат (допустимая последовательность операторов и ее "значение").

Примерами вычислительных моделей могут служить машины Тьюринга, схемы программ и магазинные автоматы, в то время как уравнения, определяющие рекурсивные функции, и контекстно-свободные грамматики вряд ли можно назвать вычислительными моделями.

Следующие четыре требования являются, по моему мнению, необходимыми условиями для того, чтобы вычислительная модель могла найти практическое применение.

Требование I. УСТРОЙСТВО ДОЛЖНО УМЕТЬ ОБРАБАТЫВАТЬ НЕ ТОЛЬКО СТРОКИ, НО И ДЕРЕВЬЯ. Устройства, обрабатывающие строки, кажутся плохо приспособленными для представления реальных вычислений, которые обычно выполняются над структурированной памятью. В этом случае наиболее естественным подходом является представление заключительных и промежуточных результатов вычислений в виде функциональных термов (деревьев). Более того, понятие дерева в программировании является, по-видимому, таким же фундаментальным, как и понятие действительного числа в математическом анализе. Переход от теории вычислительных моделей, обрабатывающих строки, к теории вычислительных моделей, обрабатывающих деревья, в программировании можно сравнить с переходом от теории полугрупп к теории универсальных алгебр в алгебре. Мне кажется, что другое обобщение теории программирования, связанное с идеей учета структуры данных при помощи определяющих соотношений (от свободных полугрупп к полугруппам с соотношениями в алгебре) вряд ли будет иметь широкую сферу приложений.

Конечно, я не первый, кто отмечает важность понятия дерева в программировании. Существует довольно большое количество работ, в которых описываются различные устройства, обрабатывающие данные, представленные не только в виде деревьев, но и в виде более сложных структур. Проблема состоит в построении и глубоком изучении ряда вычислительных моделей, предназначенных для обработки деревьев и удовлетворяющих другим требованиям для практичных вычислительных моделей.

Требование 2. УСТРОЙСТВА ДОЛЖНЫ БЫТЬ ДОСТАТОЧНО СВОБОДНЫМИ. Интуитивное представление об абсолютно свободном устройстве дает конечный автомат, а о сильно несвободном - стандартная схема программы. Убежденность в необходимости этого очень неформального требования возникла у меня после ряда безуспешных попыток понять причины неразрешимости некоторых алгоритмических проблем в простых, на первый взгляд, подклассах стандартных схем. Во-первых, оказалось, что во многих случаях множества допустимых вычислений устроены настолько

сложно, что вряд ли могут быть описаны с помощью современных методов. Во-вторых, я обнаружил, что в теории схем программ существует очень мало положительных результатов о разрешимости, которые не имеют прямых аналогов в более "прозрачных" и хорошо изученных структурах алгебры и логики. В-третьих, подклассы стандартных схем, допускающие интересные постановки проблем разрешимости, очень часто выглядят искусственными с точки зрения практических приложений. По-видимому, для того, чтобы "нащупать" границу между разрешимыми и неразрешимыми проблемами программирования, целесообразнее переходить от изучения более свободных моделей к менее свободным, вводя "несвободу" на каждом уровне иерархии строго контролируемым способом.

Требование свободы для устройства можно сравнить с требованием гладкости для функций в математическом анализе. Изучение негладких функций ведется с помощью более сложной техники, например, с помощью их приближений гладкими функциями. Вполне вероятно, что такой подход возникнет и в теоретическом программировании, но так как сегодня мы не обладаем точным знанием о "гладких функциях" программирования в достаточной мере, то их изучение должно быть одной из первоочередных задач.

Требование 3. ЭКВИВАЛЕНТНОСТЬ УСТРОЙСТВ ДОЛЖНА УЧИТЫВАТЬ ИХ СТРУКТУРУ. Другими словами, алгоритм, проверяющий эквивалентность устройств, должен учитывать не только заключительные результаты вычислений, но и некоторые истории их получения. Теоретическую мотивировку этого требования дает следующий неутешительный результат: для многих содержательных вычислительных моделей проблема функциональной эквивалентности является неразрешимой, если устройства рассматриваются как "черные ящики" (устройства называются функционально эквивалентными, если для произвольной пары равных входных данных они вычисляют равные результаты). Я неоднократно убеждался на практике, что программист при создании программы упорно следует своей первоначальной идее о ее "логической схеме" и значительно менее охотно изменяет управляющую структуру программы, чем информационную. Он склонен рассматривать не весь класс программ, функционально эквивалентных данной, а только те программы из этого класса, которые имеют "похожую" управ-

ляющую структуру. Это наблюдение можно считать практической мотивировкой требования 3.

Близкое по духу требование для стандартных схем программ впервые выдвинул А.П.Ершов, который ввел понятия истории вычисления и формальной эквивалентности стандартных схем. В случае, когда отношение эквивалентности учитывает управляющую структуру, кажется более естественным представлять истории вычислений в виде деревьев, вершины которых помечены предикатными термами, а денотационную семантику определять в терминах алгебраических систем, а не универсальных алгебр.

Требование 4. ЭКВИВАЛЕНТНОСТЬ УСТРОЙСТВ ДОЛЖНА ПРОВЕРЯТЬСЯ ЗА ПОЛИНОМИАЛЬНОЕ ВРЕМЯ. Это сильное требование является, как мне кажется, необходимым для существования достаточно широкого набора быстрых алгоритмов глубоких эквивалентных преобразований устройств, обосновывающих методы глобальной оптимизации, анализа и трансляции программ. На современном этапе развития вычислительной техники одним из необходимых условий для практической реализации алгоритма является существование для него верхней временной оценки сложности не больше (по порядку) n^3 или, иногда, n^4 . В ближайшие десятилетия трудно ожидать более чем двойного увеличения этой характеристики. Поэтому вряд ли можно считать практичными (даже с учетом перспективы) алгоритмы распознавания эквивалентности устройств с верхней временной оценкой сложности по порядку превышающей n^8 , где n - максимальный размер сравниваемых устройств.

Я не верю в то, что может быть построена универсальная вычислительная модель, в рамках которой будет развита содержательная алгебра программирования (единый математический аппарат, обосновывающий разнообразные методы преобразования программ, применяемые в практике). Однако я верю, что существует целый ряд практичных вычислительных моделей, которые в совокупности могли бы служить основой для создания достаточно широкой алгебры программирования.

Свое выступление я рассматриваю как еще один призыв к исследованию более практичных вычислительных моделей. Время мне кажется подходящим: практика программирования начинает не только методологически, но и непосредственно влиять на основ-

ные понятия и структуры математики. Ярким свидетельством этому является и научная программа нашего симпозиума.

Все мы, собравшиеся в этом зале, пришли на поклонение одному богу, и, узнав о его научном мировоззрении из великолепных лекций профессора Г.Земанека, можно предположить, что в эти дни он, в нетерпении поерзывая сразу на двух стульях, пытается нам сказать: "Ближе к практике, коллеги! Помогите же мне немного!"