Логики и стили программирования

Н. Н. Непейвода, Ижевск, Удмуртский государственный университет[1]

Доказательством варварства русских является то, что они буквально помешаны на мытье: они, по слухам, чуть не каждую неделю ходят в баню!
(Немецкий путешественник XVII века)
Нецивилизованность русских доказывается тем, что они не верят рекламе: чем больше рекламируется товар, тем хуже они его покупают!
(Американский социолог рубежа XX–XXI веков).

Programming style is a notion corresponding to scientific paradigm. It is often logical in essence because programming is activity which is nearest to God's (or Demiurg) creation.
In book [1] programming is analysed practically but from the point of view of advanced logic and analytical philosophy. Some relations between logic and programming styles are summarized here.
Furthermore there are some methodological principles which can be substantiated analyzing so complex and often obviously erroneous activity. Some of them are summarized here. We can stress e.g. that 'positive thinking' is the least in creativity. Thus Russian 'negative' mentality is in essence more progressive than Western one.

Современная логика и философия накопила громадный потенциал высокоуровневых понятий, концепций и результатов. Этот потенциал просится в дело. Здесь показано, как он был применен при создании книги [1] к анализу сложнейшей области — современного программирования. Задача анализа в данном случае была затруднена большим объемом информации, изложенной таким образом, что на первый план выступают случайные внешние особенности аппарата. Она облегчалась, во-первых, тем, что в программировании создаются настоящие искусственные объекты (см. [5]), *в принципе* подчиняющиеся лишь законам классической первоуровневой логики (в первую очередь закону противоречия), и во-вторых, тем, что теория, которая используется для обоснования принятых решений, очевидно и грубо неадекватна[2], а принципиальное методологическое невежество создателей заставляет их допускать очевидные концептуальные ошибки.

В книге [5] введено понятие *стиля программирования*.

Под *стилем программирования* понимается внутренне согласованная совокупность базовых конструкций программ и способов их композиции, обладающая общими фундаментальными особенностями, как логическими, так и алгоритмическими. Стиль включает также совокупность базовых концепций, связанных с этими программами.

Стиль программирования реализуется через *методологии* программирования, заключающиеся в совокупности соглашений о том, какие базовые концепции языков программирования и какие их сочетания считаются приемлемыми или неприемлемыми для данного стиля. Методология включает в себя, в частности, модель вычислителя для данного стиля.

Методология реализуется через методики, которые состоят из следующих компонент:

a) Поощрение (или прямое предписание) использования некоторых базовых концепций программирования.
b) Запрещение (или ограничение) применения некоторых других базовых концепций. Иногда запрещение либо ограничение может быть неявным,

[1] Данная работа частично поддержана грантом РГНФ 02-03-18307а

[2] Это не означает, что теория, на которую ссылаются, сама по себе плоха. Но часто хорошую теорию используют как заклинание для обоснования решений, никакого отношения к ней не имеющих. Масса примеров таких случаев приведено в книге [5].

через исключение нежелательных концепций из предписываемого языка или его диалекта.

c) Требования и рекомендации по оформлению и документированию программ.
d) Совокупность инструментальных и организационных средств, поддерживающих все вышеперечисленные требования и рекомендации.

Заметим, что в программировании происходит систематическая ‘возгонка' понятий. То, что в других местах называется орудием, здесь называется методом; то, что называется методом или методикой, называется методологией; то, что называется методологией, возводится в ранг парадигмы, и т. д. Здесь мы придерживаемся выражений, более адекватно передающих смысл используемых понятий.

Можно выделить, в частности, следующие стили программирования.

Четыре или пять стилей первого уровня:

a) Программирование от состояний;
b) Структурное программирование;
c) Сентенциальное программирование;
d) Программирование от событий;
e) Программирование от процессов и приоритетов;

С программистской точки зрения эти стили различаются по следующим характеристикам.

В программировании от состояний процесс представляется как смена состояний системы. Новое состояние возникает в результате действия, изменяющего старое состояние, а выбор этого действия зависит от проверки условий. Математической моделью программы служит конечный автомат. Инвариантом применимости данного стиля является следующая характеристика задачи:

Действия глобальны, условия локальны.

Естественным способом программирования такой задачи на современных языках программирования является использование операторов **goto** либо объектов, обменивающихся информацией через общее поле памяти.

В структурном программировании, которому сейчас учат как монопольному первоуровневому стилю, *действия и условия локальны*. Управляющие действия образуют иерархическую структуру, а потоки передачи данных *в принципе* должны согласовываться с данной структурой. Этот стиль поддерживается структурами современных традиционных языков программирования (например, Pascal, C). Математической моделью программ являются здесь вычислимые функции.

В сентенциальном стиле (Рефал, Пролог[3]) *действия и условия глобальны*. Каждый шаг программы проверяет все поле зрения на соответствие образцу, находит тем самым применимое правило преобразования, и, согласно найденному правилу, преобразует все поле памяти.

В программировании от событий и от приоритетов *действия локальны, условия глобальны*. Условие состоит в том, что в системе произошло некоторое событие, лучшим обработчиком которого оказалось данное действие. Но в программировании от событий такое событие (например, щелчок мыши) снабжает процесс-обработчик важной информацией (например, о месте щелчка). А при программировании от приоритетов событие состоит в том, что все более приоритетные процессы ничего не могут сделать, и никакой позитивной информации активизируемому процессу не дает.

Большинство этих стилей с точки зрения управления соответствуют разным вариантам пропозициональных конструктивных логик. Программирование от состояний соответствует нильпотентной логике. Структурное программирование в первом

[3] Программирование на Прологе традиционно называют логическим. От логики здесь ничего не осталось, и термин вводит в заблуждение (и даже если бы осталось, логика здесь была лишь инструментом, и ставить конкретный инструмент впереди концепции то же самое, что телегу впереди лошади).

приближении соответствует фрагменту линейной логики с одной линейной импликацией в формуле. Сентенциальное программирование соответствует интуиционистской логике, ограниченной одной импликацией. Программирование от событий соответствует фрагменту линейной логики, в котором формулы содержат две импликации: внешняя из них интуиционистская, внутренняя находится в заключении внешней импликации и является линейной. Программирование от процессов и приоритетов пока что адекватного логического описания не имеет. Видимо, релевантные логики могут оказаться здесь полезными.

С точки зрения передачи данных программирование от состояний и структурное программирование соответствует классической логике предикатов, описывающей взаимоотношения между входными и выходными данными. Сентенциальное программирование соответствует исчислениям общего типа, поскольку данные до и после преобразования описываются как результат применения правила с метапеременными. В программировании от событий описание данных является открытой формулой классической логики предикатов, свободные переменные которой соответствуют передаваемой информации. В программировании от приоритетов глобальные данные описываются замкнутой формулой, удостоверяющей лишь факт истинности достаточно сложного условия (отсутствие более приоритетных процессов, годящихся для данной задачи).

Ни один стиль программирования не описывается чисто логически. Это связано прежде всего с наличием в языках программирования фиксированных типов данных, например, чисел. Наличие хотя бы одного фиксированного типа данных вызывает к жизни все затруднения теоремы Геделя о неполноте и парадокса изобретателя. Итак, первый нетривиальный вывод:

Нужно четко разделить композицию понятий и их конструирование. Работу с исходными типами данных нужно помещать в среду суровых ограничений на правила композиции, а там, где требуется создавать сложные структуры понятий, конкретные данные нужно полностью отбросить, имея лишь косвенные ссылки на них через используемые модули низшего уровня. Итак, вычисления и рассуждения концептуально противоречат друг другу.

Из этого следует и педагогический вывод: математика, преподаваемая на базе монополии анализа, годится для описания физических, но не годится для описания информационных процессов. Для информатиков намного важнее логика, алгебра, топология, лингвистика и философия. Для них важнее преобразовывать понятия, чем формулы.

На уровне понятий более высокого уровня возникают другие стили.

Функциональный стиль программирования, когда программа представляет собой функционал высокого уровня, преобразующий функции, представлен языками ЛИСП и ML. Если функции типизированы, то этот подход, сохраняя возможности понятий высших уровней исключительно компактно выразить сложную структуру, вдобавок крайне эффективен по использованию ресурсов. Но нынешние реализации функционального программирования не могут удержаться от использования рекурсий и нетипизируемых конструкций типа оператора вычисления произвольного выражения или оператора неподвижной точки. Это создало функциональному программированию репутацию крайне неэффективного стиля, подходящего лишь для прототипов программ. Функциональному стилю соответствуют интуиционистская логика предикатов (типизированный вариант) и комбинаторная логика (нетипизированный вариант).

Рассмотрение функционального стиля приводит к выводу о губительном влиянии на современную высокоуровневую практику двух факторов. Во-первых, это *иллюзия универсальности*. Пора рассматривать слова ‘универсальная система’ как нечто подобное философскому камню (как рениксу). Иллюзия универсальности заставляет вводить в систему возможности, губительным образом расширяющие ее. Во-вторых, это

игнорирование отрицательных результатов, или, в более общем виде, *позитивное мышление и квазирелигия прогресса*. Закрывание глаз на теоретические предупреждения никогда ни к чему хорошему не приводит, а их систематическое изгнание из учебных курсов в угоду душевному комфорту и ‘позитивности’ вредно сказывается на способностях к творческому мышлению, один из приемов которого: превратить минус в плюс[4].

Объектно-ориентированый стиль является четверть-шагом к функциональному с точки зрения теории. Объекты показали даже достаточно необразованным людям. Что для сложных систем работа с действиями эффективнее работы с данными. С точки зрения практики, это несколько разных стилей, один из которых берет начало скорее в психологии и искусственном интеллекте, и успешно применяется для массового производства программ, таких, где внешняя упаковка важнее сути. Здесь объекты в разных контекстах могут вести себя совершенно по-разному, так что они соответствуют ролям людей. В данном случае четко видно, как ссылка на неадекватную сути подхода теорию абстрактных типов данных заставляет выпячивать слабейшие стороны подхода и мешает развитию действительно сильнейших. Программистам бы здесь к психологам обратиться, а не к математикам!

Далее, этот стиль великолепно иллюстрирует, как применяется в современной практике принцип Оруэлла “Невежество — сила”. Даже там, где известны хорошие решения, принимаются случайные, а влияние монополистов мешает развитию альтернатив.

Вообще, наблюдение за состоянием дел в современном “искусственном интеллекте” и информатике приводит к следующему закону экологии искусственных систем:

Первая система, декларировавшая успех[5] *в данной экологической нише, захватывает ее и изгаживает таким образом, чтобы никакая другая в ней не могла выжить.*

Далее, общая беда всех стилей и методологий программирования — практическое игнорирование того, что программу не опишешь понятиями, которые заданы внутри ее самой. Давно уже известно, что для полного описания свойств программы необходимы призраки — понятия, в программу не входящие и зачастую просто вредные для вычислений и непредставимые в машине (например, ординалы), но необходимые для корректного описания системы. Типичным примером призрака является число шагов цикла общего вида. Масса примеров возникновения и использования призраков приведена в [1].

Несколько слов о роли глобализации. Глобализация усиливает монополизацию, в том числе и в интеллектуальной схеме. В связи с этим исключительно интересен пример сентенциального программирования. Этот стиль возник независимо в СССР и на Западе, и созданные его конкретизации оказались ортогональными. При множестве общих черт Рефал и Пролог несовместимы по механизму сопоставления поля зрения с образцом и механизму управления. Если Рефал великолепно приспособлен для &-параллелизма, когда параллельно запускается несколько процессов и результат получается обобщением всех полученных частных результатов, то Пролог столь же великолепно приспособлен к V-параллелизму, когда несколько процессов соревнуются в том, кто быстрее достигнет

---

[4] Впрочем, что Вы хотите, если в США принята социологическая программа по мягкому лишению вундеркиндов их способностей. Конечно же, при этом говорится масса хороших слов о том, что стремятся к их счастью, но одна фрейдистская оговорка подчеркивает ее суть: гении подрывают ценности потребительского общества, поскольку получают от творчества гораздо большее наслаждение, чем обычный человек от секса. Более того, это наслаждение недоступно большинству, и тем самым его не купишь за деньги!

[5] Насколько этот успех реален, обычно второй вопрос, которым не задаются (по причине позитивного мышления), пока не станет слишком поздно.

цели. Ввиду «железного занавеса» обе концепции выжили, встали на ноги, актуальны до сих пор.

Если же рассмотреть, скажем, структурное программирование, то ввиду свободного обмена идеями и единого информационного пространства одна из его разновидностей прочно монополизировала экологическую нишу. В результате развитие параллельных компьютеров сдерживается прежде всего тем, что студентов с самого начала приучают программировать последовательно[6]. Более того, структурное программирование практически задавило совершенно другой, применимый в других условиях, стиль: программирование от состояний. **Goto** было объявлено вредным оператором. А оно просто плохо совместимо с оператором присваивания.

В последние годы программирование от состояний было возрождено и стало секретным оружием команды петербургских программистов, дважды ставших чемпионами мира (см. [3]).

Сейчас видно, что, так же, как абсурдно ставить задачу доказательства правильности программы после того, как программа написана, абсурдно решать задачу распараллеливания написанной на обычном языке программы. Очевидно, что логические системы, соответствующие структурному программированию, после некоторой модификации великолепно описывают модификацию этого стиля, параллельную с самого начала. Но по закону экологической ниши такая модификация уже не могла появиться, тем более, что успехи структурного программирования отнюдь не были мнимыми, и теория, на которую оно опирается, является (редчайшее исключение!) релевантной[7]. Тем самым глобализация сыграла здесь отрицательную роль. А сколько еще подобных случаев, невидимых невооруженным глазом[8]!

Наконец, заметим, что переход к следующему уровню понятий невозможен без запрещения многих методов, действовавших на предыдущем уровне. Но в программировании этому препятствует иллюзия универсальности, конкретизирующаяся в предрассудок совместимости, который заставляет при развитии системы тянуть за собой шлейф концептуально несовместимых устаревших понятий. Случаи, когда от этого предрассудка отказались, считаны.

И ко всему этому я присовокупляю, что пора логике перестать быть разделом математики либо философии, и встать и в педагогике, и в практике на свое место самостоятельной науки высшего порядка, связывающей точное и гуманитарное знание.

Литература

1. Н. Н. Непейвода, И. Н. Скопин. *Основания программирования*. Ижевск-Москва, РХД, 2003, 926 с.
2. Н. Н. Непейвода. Квазиискусственные объекты. Современная логика-2002. СПб, 2002, стр. 28-30.
3. А. А. Шалыто. SWITCH-технология. СПб: Наука. 1998, 626 с.

[6] Интересно, что первая «программа», вдохновившая Беббиджа на создание первой вычислительной машины, была параллельной. Во время Великой французской революции французские математики организовали работу по пересчету артиллерийских, навигационных, астрономических и геодезических таблиц, запрограммировав с помощью конечных разностей их приближенные вычисления таким образом, что их могла параллельно выполнять рота солдат, каждый из которых делал сложение либо вычитание и передавал результат следующим.

[7] Это подчеркивает еще одну опасность точных методов. Точный метод можно применить неточно и неадекватно, что, конечно, плохо. Его можно применить точно и адекватно, лишь ограничив область теми случаями, которые подходят под данную формализацию. После такого применения, как правило, сделанные ограничения забываются и все, что не подходит под формализм, оказывается якобы несуществующим.

[8] Автор не против свободы. Но ничто человеческое не является абсолютной ценностью. Скажем, чтобы добиться лучшего выражения своих чувств и мыслей, человек отказывается от свободы выражения, переходя от прозы к стихам. Так что пусть будет и обмен идеями, и трудности в этом обмене!