ПРОЛОГ. ПРОШЛОЕ. НАСТОЯЩЕЕ. БУДУЩЕЕ?

Лидовский В.

Семантика названия

Знакомство с языком программирования Пролог, как правило, начинается с раскрытия его имени: Пролог — это сокращение от Программирование Логики. Однако, это раскрытие вводит в заблуждение, которое почему-то до сих пор стараются поддерживать авторы многих публикаций на соответствующую тему. На самом деле, Пролог связан с логикой лишь исторически — на практике эта связь может быть обеспечена лишь на уровне волевого усилия программиста. Этим объясняется взлет с начала 80-х и закат в начале 90-х популярности этого языка. Действительно, освоить механизмы резолютивного вывода Пролога не просто, но освоив их, легко понимаешь, что вывод этот назвать логическим можно, только имея большое желание не видеть очевидного. Первые технологичные трансляторы с Пролога появились в начале 80-х и за десять лет программисты смогли разобраться, что это не предвещает ни революции в программировании, ни даже небольшого облегчения их работы. Но не все так просто... Трансляторы с Пролога продолжают появляться как коммерческие, например, Visual Prolog фирмы PDC, так и условно-бесплатные. По лицензии GNU распространяется Пролог Даниэла Диаса (Daniel Diaz), соответствующий стандарту ISO/IEC 13211-1. Все дальнейшие примеры тестировались на версии 1.1.2 этого транслятора, доступной с декабря 1999 года.

Пролог увенчал собой многочисленные и растянутые на века попытки формализовать процесс мышления. И в этом смысле, гора родила мышь... Но, как это часто бывает, неуспешное или неполное решение одной проблемы может решать другую.

Пролог и логика

Еще в Древней Греции были предприняты первые попытки свести работу ума к последовательности почти механических операций над точно заданными сущностями. Описание геометрии Евклидом (III век до н.э.), использовавшим подобный подход, оказалось настолько удачным, что его до сих пор используют в школах. В XVII веке немецкий философ и математик Лейбниц безуспешно пытался создать формальную систему, в

которую можно было бы вписать всю математику. В начале XX века, когда стало ясно, что без формализации естественных операции интеллекта, т.е. логики, математика может стать противоречивой, другой известнейший немецкий математик Гильберт предложил попытаться либо создать систему, формально описывающую всю математику, либо доказать, что это невозможно. Спустя 30 лет Гедель доказал свою знаменитую теорему и математики (философы, программисты, ...) узнали, что их не постигнет судьба географов — работа у них будет всегда.

Под формальной системой, упомянутой выше, подразумевается конечный ряд истинных утверждений и фактов, из которых при помощи конечного числа точно определенных операций можно было бы получить все прочие истинные утверждения и факты. Система Евклида как и классические системы логики, созданные в XIX и в начале XX века, состояли из трех компонент: языка, аксиом (априорно истинных утверждений) и правил вывода. Подобные системы предназначались для описания интеллектуальных операций, т.е. для того же, для чего можно использовать естественный язык. Но вследствие неоднозначностей и неточностей, присущих последнему, большинство из них использовало специальный искусственный язык. Ныне языком таких формальных логических систем является язык логики предикатов 1-го порядка. Например, фраза “У Артура есть компьютер” на этом языке описывается как

имеет(Артур, компьютер),

фраза “Компьютер есть у каждого” —

для всех X имеет(X, компьютер),

фраза “У Артура есть компьютер, но нет программ” —

имеет(Артур, компьютер) и не имеет(Артур, программы),

фраза “Если у кого-то есть программы, то Артур его ищет” —

существует X (имеет(X, программы) следовательно ищет(Артур, X)).

Математики для связок “и”, “или”, “не”, “следовательно”, “для всех”, “существует” обычно используют значки $\&$, $\vee$, $\Rightarrow$, $\neg$, $\forall$ и $\exists$, а программисты слова and, or, imp, not, all и exists соответственно. В предыдущих примерах “имеет” и “ищет” — это предикаты. Другими словами, предикаты — это функции, возвращающие логические значения. На язык предикатов 1-го порядка можно перевести большинство обычно используемых предложений естественного языка, но не все. Например, предложение “Какие связи имеются между Артуром и компьютером?” перевести нельзя. Дело в том, что предикаты 1-го порядка не могут использовать аргументы-предикаты.

Предикаты 2-го порядка могут использовать аргументы-предикаты 1-го порядка и т. д. Таким образом, рассмотренная фраза естественно переводится на язык предикатов 2-го порядка. Хотя, очевидно, что чем выше порядок языка, тем ближе он приближается к естественному, на практике даже язык 1-го порядка нельзя использовать без ограничений. Это связано с непреодолимыми теоретическими ограничениями на возможность организации эффективного логического вывода.

В начале 30-х годов XX века Эрбраном был предложен метод резолюций — формальная система для логики предикатов 1-го порядка, которой в отличие от классических систем не требовались аксиомы и которая основывалась на единственной формальной операции. Естественно, эта система допускала несложную реализацию на компьютере, что и было предложено Робинсоном в 1965 году. Однако, как было доказано, вывод в такой системе — это процесс переборный и не всегда конечный. Переборные алгоритмы можно использовать только на очень маленьких исходных данных, в противном случае происходит "комбинаторный взрыв", когда количество вариантов для анализа начинает превосходить любые физические величины, например, количество атомов во вселенной. Бесконечный же вывод возможен, если попытаться доказать специальным образом сконструированное предположение, которое не следует из имеющихся данных.

С начала 70-х усилиями Колмероэ, Ковальского и Уоррена удалось для части языка предикатов 1-го порядка, называемого языком предложений Хорна, разработать метод SLD-резолюций и создать на его основе эффективную программу вывода. Язык предложений Хорна, дополненный некоторыми необходимыми для языка программирования средствами, а также средствами, специфичными для реализации метода SLD-резолюций, стал тем, что сегодня называется языком Пролог. В Прологе для записи связки "и" используется запятая, для "или" — точка с запятой, для "следовательно" — комбинация двоеточия и дефиса. В Прологе нет отрицания и нет возможности для записи кванторов, т. е. связок "для всех" и "существует". И даже для такого остатка языка логики предикатов не удалось реализовать механизм логического вывода. Вместо него был реализован так называемый процедурный вывод, которой в принципе не в состоянии всегда находить все логические следствия из заданных утверждений. Рассмотрим пример программы на Прологе; комментарии в нем начинаются со знака % и заканчиваются концом строки.

```
% Описание графа
path(d,b). %существует путь из пункта d в пункт b
path(b,f). %существует путь из b в f
```

```
% Описание того, как искать путь в произвольном графе
path(X,Y) :- path(X,Z), path(Z,Y). %если существует путь из X
  % в Z и существует путь из Z в Y, то существует путь из X в Y
path(X,Y) :- path(Y,X). %если существует путь из Y
  % в X, то существует путь из X в Y
```

Очевидно, из логических утверждений, составляющих программу следует, что, например, существуют пути: из d в b, из b в d, из b в f, из f в d. Пролог, однако, никогда не найдет пути из f в d или даже из f в b. На самом деле, даже для языка предложений Хорна, в котором допустимы только предикаты с не более чем одним аргументом, настоящий логический вывод эквивалентен синтаксическому анализу фраз контекстно-свободного языка, т.е. является хотя и всегда конечным, но переборным процессом.

Введение в Пролог нелогических средств таких как, например, “предикаты” отсечения или лжи, без которых практически обойтись очень трудно, вместе с тем, что предикаты в нем используются как подпрограммы, фактически означает, что попытка реализации формальной логической системы вывода на компьютере привела к реализации чего-то совершенно другого, с логикой почти не связанного.

Пролог, ООП и визуальное программирование

Сегодня почти идеальным языком программирования является C++, которым обеспечивается как высокая эффективность получаемых кодов, так и поддержка структур данных, максимально приближенных к моделируемым программами взаимосвязанным сущностям. C++ легко вписывается в любую систему визуального создания программ. Это обусловлено его объектно-ориентированной природой. С другой стороны, попытайтесь представить ход решения на C++ следующей задачи: рассчитать в заданной электрической схеме сопротивление между заданными точками. Электрическая схема естественно формализуется нумерацией точек соединения контактов и таблицей из трех столбцов, содержащих пары номеров непосредственно соединенных контактов и сопротивления между ними. Используя Пролог, автор решил эту задачу менее чем за час. В программе на C++ фактически пришлось бы реализовывать встроенные механизмы Пролога. Однако, используя Пролог, трудно создать приемлемый пользовательский интерфейс в силу того, что этот язык практически невозможно соединить с концепциями ООП. Работая с системой наподобие Visual Prolog, ощущаешь, что работаешь на самом деле

с двумя хотя и искусстно, но искусственно склеенными частями от разных механизмов. Единственная возможность естественным образом соединить механизмы вывода Пролога с современным интерфейсом пользователя — это создать на C++ класс Prolog, в простейшем случае, имеющий вид:

```
class Prolog {
  public:
    list<string> db;      //пролог-программа
    string goal;          //цель
    list<string> results; //список результатов
    void find();          //поиск решений
};.
```

Подобный подход позволит как использовать метод поиска решений, встроенный в Пролог, так и использовать средства C++, в частности, для организации ввода-вывода. Предложенный класс Prolog можно значительно усовершенствовать. Например, добавив в закрытую часть класса внутреннее, эффективное для поиска решений представление базы данных. Можно добавить средства для отслеживания работы функции поиска и прочее, и прочее...

Создавая подобный класс, можно использовать неплохие средства GNU Пролога или других реализаций этого языка для связи с программами на C++.

Пролог и базы данных

Есть серьезные основания утверждать, что Пролог на самом деле представляет из себя не много не мало, а идейный костяк СУБД будущего.

Действительно, ныне наиболее широкое распространение получили реляционные СУБД, управляющие данными, представимыми в виде таблиц. Существующие средства позволяют эффективно работать с данными, организованными таким образом, объемом десятков миллиардов записей. Стандартным языком для работы с табличными данными является с начала 80-х годов язык SQL. Для проверки дальнейших примеров использовалась система PostgreSQL* версии 6.4.2 1999 года, распространяемая на условиях BSD.

---

* Система PostgreSQL разрабатывается в Калифорнийском университете Беркли по заказу ряда оборонных

Однако, любую таблицу или множество таблиц можно описать на Прологе. Поэтому пролог-программы обычно и называют базами данных или, чтобы подчеркнуть их большую выразительную силу, базами знаний. Например, таблицу, содержащую города и расстояния по железной дороге между ними, можно описать так:

```
%     City1,    City2,   distance
paths('Москва', 'Брест', 1094).
paths('Москва', 'Абакан', 4229).
paths('Москва', 'Севастополь', 1542).
paths('Киев', 'Брест', 577).
```

В этой пролог-программе описана таблица по имени paths, состоящая из 4-х записей и 3-х полей. На SQL запрос о расстоянии между Москвой и Брестом к подобной таблице выглядит так:

```
SELECT distance
FROM paths
WHERE City1 = 'Москва'
AND City2 = 'Брест';.
```

На Прологе этот же запрос будет таким:

```
paths('Москва', 'Брест', X).
```

Оба запроса гарантируют, что если существует несколько путей между двумя пунктами, то обо всех из них будет сообщено в ответе. В этих запросах важно, чтобы на первом месте стояла Москва, а затем Брест, что очень неудобно. Если не знать, что ставить на первое место, то на SQL придется либо требовать удвоения объема базы данных, либо писать следующий запрос:

```
SELECT distance
FROM paths
WHERE City1 = 'Москва'
AND City2 = 'Брест'
```

и научных учреждений США с 1986 года с целью создания эффективной СУБД структуры более сложной, чем реляционные.

```
OR City2 = 'Москва'
AND City1 = 'Брест';.
```

Для того, чтобы избежать подобных неудобств при работе с Прологом, достаточно добавить к программе одно правило:

```
paths(X, Y) :- paths(Y, X).
```

В случаях, подобных поиску длин путей между Москвой и Киевом, SQL вообще ничем вразумительным помочь не сможет. Пролог же легко справляется с подобными задачами, т. е. поиском маршрутов в произвольном графе, заданном только путями, соединяющими соседние пункты, добавлением нескольких новых, хотя и не слишком тривиальных правил к своей базе данных.

То, что базовая часть SQL может быть реализована на Прологе, подтверждается тем, что некоторые дистрибутивы Пролога в качестве примера включают подобную реализацию. Такие средства SQL, как индексы, сортировка, группировки и т. п. связаны либо с машинным представлением данных, либо с готовыми высокоуровневыми функциями. Машинное представление базы данных на Прологе можно также оптимизировать, используя индексы, а с добавлением высокоуровневых средств вообще никаких проблем кроме чисто технических быть не может.

Что же касается баз данных не реляционной структуры, например, сетевых, продукционных или иерархических, которые для многих задач являются оптимальными с точки зрения соответствия структуре данных этих задач, то с ними связана та же проблема, что и с базой данных, описанной на языке логики предикатов, а именно отсутствие формальных, эффективных (не переборных) методов извлечения из них той части информации, которая соответствует заданному запросу.

Итак, для баз данных, структуры более сложной, чем предлагаемая Прологом, не существует СУБД, способной эффективно выдавать ответы на запросы. С другой стороны, структура реляционных БД является подмножеством структуры программ на Прологе. Следовательно, альтернативы Прологу как СУБД будущего нет. Конечно же имеется в виду не сам Пролог, как транслятор, а лишь язык на основе предложений Хорна, используемый для внешнего представления баз данных и, возможно, как язык запросов.

Пролог и будущее

Перспективы собственно Пролога весьма туманны. Автор, например, в течении всей весны 2000 года не мог попасть на web-страницу (http://www.logic-programming.org/prolog_std.html), посвященную стандартизации этого языка. Хотя некоторые энтузиасты возможно будут использовать его еще долго.

На пути трансформации в полноценную СУБД Прологу нужно, как минимум, пройти через следующие изменения:

- все предикаты должны стать динамическими;
- должно быть добавлено удобное средство для сохранения базы данных в файл.

В современном Прологе предикаты делятся на статические и динамические. Первые содержатся в тексте программы, а вторые возникают в процессе ее выполнения. Хуже всего то, что статические таблицы и правила, составляющие предикат, нельзя изменять динамически. Это разделение связано исключительно с эффективностью: статические предикаты — быстрее.

Если вспомнить судьбу другого языка программирования, разработанного математиками, Алгола, то предположение о малой самостоятельной жизнеспособности у Пролога получит еще одно подтверждение. Алгол, как и Пролог двадцать лет спустя, в течение десятилетия 60-х играл роль передового языка программирования, который не воспринимается профессионалами. Затем он постепенно исчез. С другой стороны, сегодня трудно назвать язык программирования, который бы в той или иной степени не основывался на идеях, впервые реализованных Алголом...

Вывод

Имя Пролога нужно понимать не как аббревиатуру, а буквально.

Использованная литература

1. Danial Diaz *GNU PROLOG* /Edition 1.1, for GNU Prolog version 1.1.2, 1999 (http://www.gnu.org/software/prolog).
2. The PostgreSQL Development Team *PostgreSQL User's Guide* /Edited by Thomas Lockhart, 1998 (http://postgreSQL.org).

3. Жан-Луис Лорьер *Системы искусственного интеллекта* — М.: Мир, 1991.
4. Тейз А., Грибомон П., Луи Ж. и др. *Логический подход к искусственному интеллекту* — М.: Мир, 1990.
5. Игорь Швыркин *Пролог. Генезис* //Мир ПК, 5/2000, 48–53.

Опубликована в журналах “Компьюлог” №4, 2000, с.63–65, “Информационные технологии” №3, 2001, с.11–13.