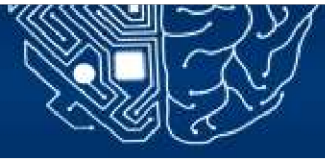
СПбГЭТУ «ЛЭТИ»

ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ

Парадигмы программирования. Логическая парадигма. Язык Пролог

Конспект лекции

СПбГЭТУ «ЛЭТИ», 2022 г.



1 ПАРАДИГМЫ ПРОГРАММИРОВАНИЯ. ЛОГИЧЕСКАЯ ПАРАДИГМА

1.1 Парадигма программирования

Парадигма программирования - это некоторый цельный набор идей и рекомендаций, определяющих стиль написания программ. Парадигма программирования представляет (и определяет) то, как программист видит выполнение программы.

Выделяют следующие основные парадигмы программирования (рисунок 1):

- Императивное программирование;
- Структурное программирование;
- Функциональное программирование;
- Логическое программирование;
- Объектно-ориентированное программирование.

Каждая из выделенных парадигм обладает своими индивидуальными особенностями, но их все можно разделить на две большие группы:

- **директивные** (directive), называемые также процедурными (procedural) или императивными (imperative), к которым относят императивное, структурное и объектно-ориентированное программирование;
- **декларативные** (declarative), к которым относят функциональное и логическое программирование.

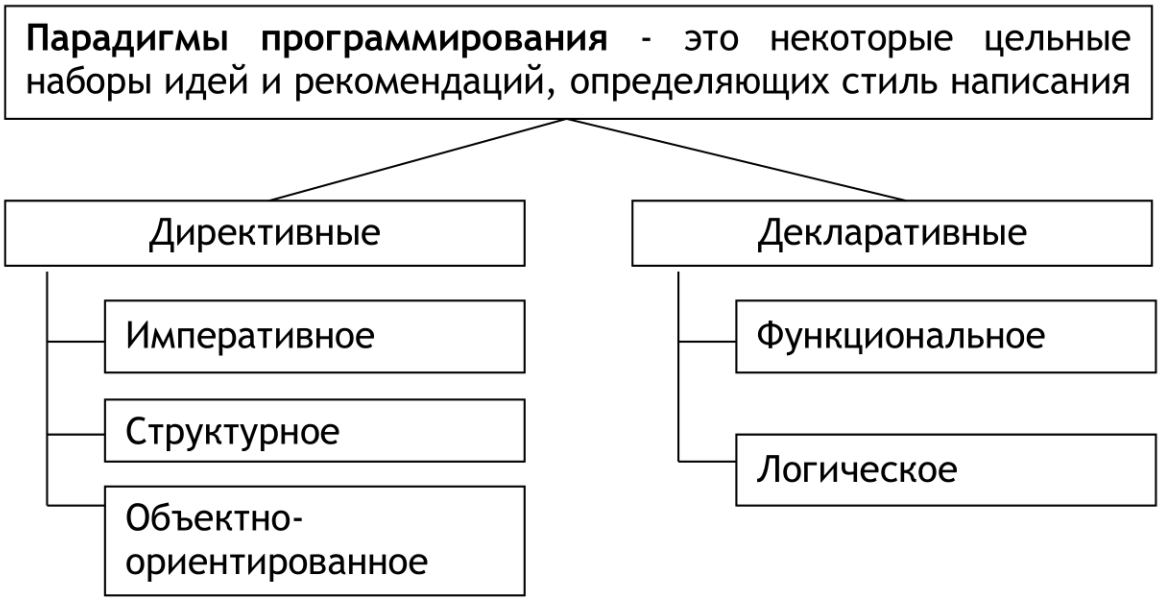


Рисунок 1 – Основные парадигмы программирования UTF-8 разработана

1.2 Современные парадигмы программирования

В настоящее время развитие информационных технологий идет очень динамично. Большое множество существующих языков программирования постоянно пополняется новыми языками, обновляются стандарты для существующих языков программирования. В основе каждого языка лежит одна, либо более парадигм программирования. Следует отметить, что современные языки программирования, базирующиеся на какой-либо парадигме программирования, часто обладают свойствами и механизмами управления, относящимися к другим парадигмам программирования. Как правило, целью сочетания нескольких парадигм в одном языке, являются удобство и упрощение решения требуемого множества задач с использованием данных языков.

Парадигму программирования можно определить как подход к разработке программ, который базируется на логически цельном множестве принципов. Каждая парадигма программирования обладает свойствами, которые определяют ее оптимальность для решения определенных классов задач. В статье приведено возможное краткое описание понятия парадигмы программирования. Детальное раскрытие сущности понятия парадигмы программирования выходит за рамки данной статьи.

Существует достаточно большое число работ, посвященных современным парадигмам и языкам программирования. Тем не менее, в данных работах вопрос классификации не рассматривается подробно. Однако классификация парадигм программирования должна способствовать в процессе изучения современных языков программирования обобщению их свойств, сравнению базовых приемов программирования и формированию группы языков для изучения в рамках предметной области их применения.

Часто предлагают разделять языки и парадигмы программирования на четыре основных типа:

- императивные,
- функциональные,
- логические,
- объектно-ориентированные.

Кроме перечисленных выше парадигм, выделяют дополнительные парадигмы программирования, к которым предлагается отнести:

– визуальное программирование,
– параллельное программирование,
– программирование ограничениями.

В качестве основных групп языков программирования и парадигм выделяют следующие:

– императивные,
– объектно-ориентрованные,
– параллельное программирование,
– функциональные,
– логические,
– скриптовые.

Множество современных подходов к программированию определяют не характеристики языков программирования, а способы проектирования и организации программ и программных комплексов. Таким образом, логично выделить следующие парадигмы программирования верхнего уровня:

– парадигмы программирования уровня архитектуры программного обеспечения,
– парадигмы программирования уровня языка программирования.

Парадигмы программирования уровня архитектуры программного обеспечения можно охарактеризовать как парадигмы, определяющие способ построения компьютерной программы. Реализация данного способа построения может быть осуществлена посредством различных технологий, базирующихся на парадигмах программирования уровня языка программирования.

Парадигмы программирования уровня языка программирования можно разделить на две большие группы:

– *императивная* парадигма программирования (от англ. imperative – повеление; приказ), называемая также *директивной* (directive), к которой относят автоматное, структурное и объектно-ориентированное программирование;
– *декларативная* парадигма программирования (declarative), к которой относят функциональное и логическое программирование.

На рисунке 2 приведена иерархия парадигм программирования верхнего уровня.

В состав парадигм программирования уровня языка программирования, дополнительно к уже двум известным парадигмам входят *прочие парадигмы* программирования. К *прочим парадигмам* программирования следует отнести парадигмы программирования, которые вносят дополнительные идеи и подходы, сочетающиеся с видами программирования из обеих предыдущих парадигм, но не входящие ни в одну из них. В качестве примеров таких парадигм могут служить следующие:

- метапрограммирование,
- визуальное программирование.

Рисунок 2 – Иерархия парадигм программирования верхнего уровня

В случае программирования в соответствии с императивной парадигмой программист пошагово описывает алгоритм решения задачи.

При использовании декларативной парадигмы программист описывает логику решения задачи. Из декларативной парадигмы программирования можно выделить виды программирования, представленные на рисунке 3.

Безусловно, существуют и другие типы декларативного программирования. Перечисленные типы решено выделить, как наиболее значимые типы декларативного программирования.

Как уже было отмечено, императивное программирование представляет собой парадигму программирования, в рамках которой программа описывает шаги, которые необходимо выполнить для решения задачи. При этом шаги изменяют состояние программы в процессе решения задачи. Из императивной парадигмы программирования можно выделить виды программирования, представленные на рис. 4.

Подвидами структурного программирования являются:

- процедурное программирование,
- объектно-ориентированное программирование.

Из парадигм программирования уровня архитектуры программного обеспечения можно выделить в качестве основных виды программирования, представленные на рисунке 5.

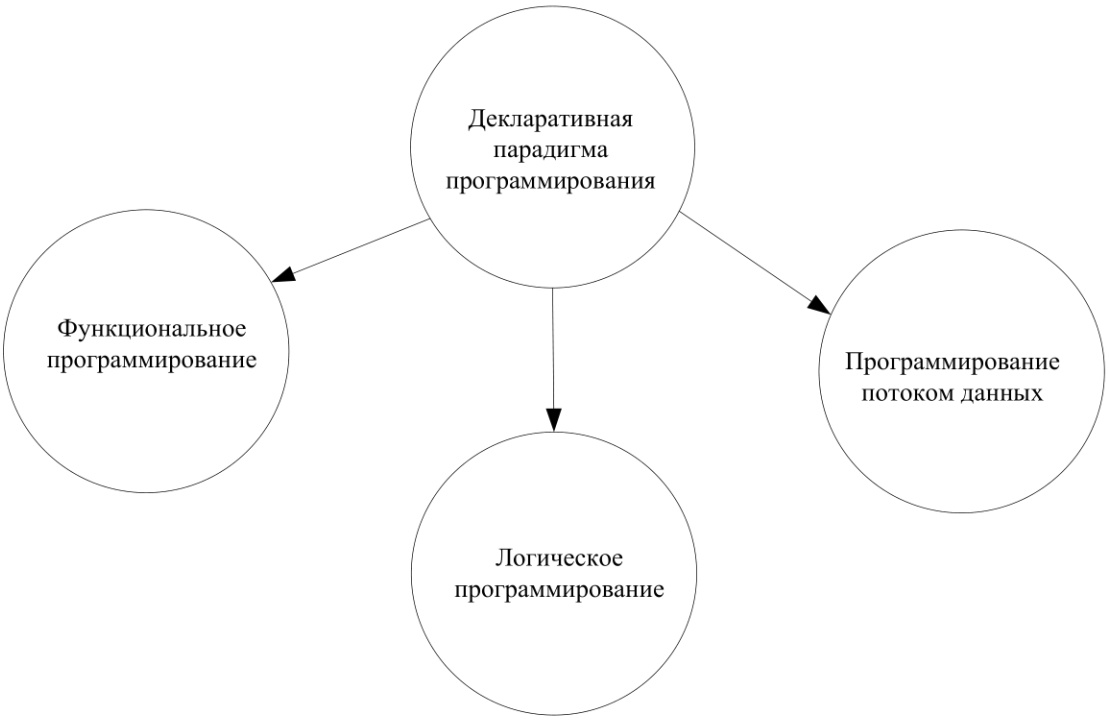


Рисунок 3 – Виды программирования декларативной парадигмы

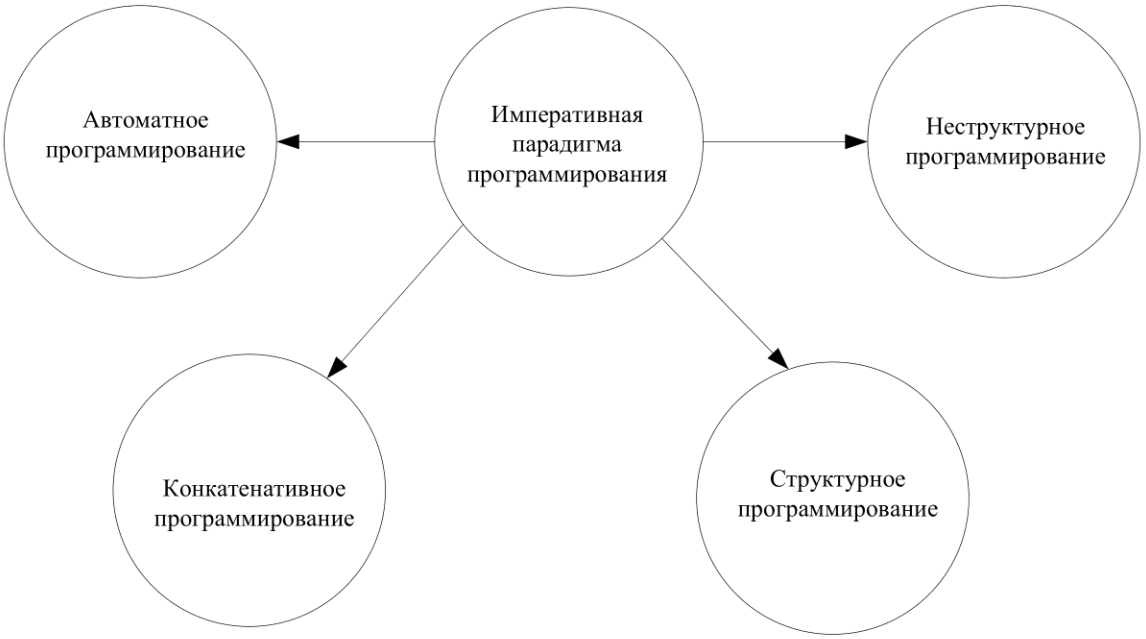


Рисунок 4 – Виды программирования императивной парадигмы

Рисунок 5 – Виды программирования парадигмы уровня архитектуры программного обеспечения

Главное различие между программами декларативного и императивного стиля заключается в следующем: программа декларативного стиля *заявляет* (декларирует), что должно быть достигнуто в качестве цели, а директивного - *предписывает*, как ее достичь.

Это различие можно пояснить на следующем примере. Предположим, вам надо пройти в городе из пункта А в пункт Б. *Декларативная программа* - это план города, в котором указаны оба пункта, плюс правила уличного движения. Руководствуясь этими правилами и планом города, курьер сам найдет путь от пункта А к пункту Б.

Директивная программа - это список команд примерно такого рода: от пункта А по ул. Садовой на север до площади Славы, оттуда по ул. Пушкина два квартала, потом повернуть направо и идти до Театрального переулка, по этому переулку налево по правой стороне до дома 20, который и есть пункт Б.

В директивной программе действия задаются явными командами, подготовленными ее составителем. Исполнитель же (компьютер) просто им следует. Хотя команды в различных языках директивного программирования и выглядят по-разному, все они сводятся либо к *присваиванию* какой-нибудь переменной некоторого значения, либо к *выбору* следующей команды, которая должна будет выполняться. Присваиванию может предшествовать выполнение ряда арифметических и иных операций, вычисляющих требуемое значение, а команды выбора реализуются в виде условных операторов и операторов повторения (циклов).

Декларативные программы не предписывают выполнять определенную последовательность действий, в них лишь дается разрешение совершать их. Исполнитель должен сам найти способ достижения поставленной перед ним составителем программы (программистом) цели, причем зачастую это можно сделать различными способами.

С появлением вычислительных машин и понятия программирования естественным образом возник вопрос о возможности автоматической генерации программ для решения определенных задач. Эта проблема связана с двумя сложностями: необходимостью формального описания условий задачи (формальной спецификации) и свойств возникающих при этом объектов, а также непосредственно алгоритм решения задачи или построения решающей программы. Стиль программирования, при котором решение задачи или алгоритм решения генерируется компьютером автоматически, а для решения необходимо только формальное описание,

называют **декларативным**, а соответствующие языки – **декларативными языками** программирования.

Одним из наиболее естественных способов описания программы является **математическая логика**, которая, с одной стороны, достаточно формализована, а с другой – допускает естественное формулирование многих свойств. Парадигма программирования, основанная на формальной логике, получила название **логического программирования**. В логическом программировании основное внимание уделяется описанию структуры прикладной задачи, а не выработке предписаний компьютеру, что ему следует делать. Программа рассматривается как набор логических фактов и правил вывода, а выполнение программы состоит в вычислении истинности (попытке доказательства) некоторого утверждения.

Логическое программирование возникло в результате исследования группы французских ученых под руководством Колмерье в области анализа естественных языков. Впоследствии было обнаружено, что логическое программирование столь же эффективно в реализации других задач искусственного интеллекта, для чего оно в настоящий момент, главным образом, и используется. Но логическое программирование оказывается удобным и для реализации других сложных задач. Например, диспетчерская система лондонского аэропорта Хитроу была переписана на Прологе.

Рассмотрим основные особенности логической парадигмы программирования. Для этого будем использовать синтаксис одного из наиболее известных представителей данной парадигмы – языка Пролог.

Программы на языке Пролог носят декларативный характер, в них отсутствуют такие управляющие конструкции, как условные операторы, операторы цикла и операторы передачи управления. Программа на языке Пролог является, по существу, спецификацией решаемой задачи в терминах объектов предметной области, фактов и правил, описывающих отношения на множестве данных объектов. Встроенный механизм логического вывода, базирующийся на методе резолюций логики предикатов, на основе данной спецификации и инициирующего запроса-цели строит алгоритм процесса дедукции для доказательства данного целевого утверждения.

Следующие свойства языка Пролог делают его эффективным инструментом для создания небольших по объему исходного текста, но

хорошо структурированных и обладающих широкими возможностями экспертных систем:

- уровень языка достаточно высок, что проявляется при сравнении возможностей, которые достигаются одинаковыми по размеру исходного текста программами на императивном языке и на языке Пролог;
- на языке Пролог удается эффективно строить адекватные конструкции, предназначенные для обработки входной и выходной информации, представляемой на естественном языке;
- программист, использующий Пролог, избавлен от необходимости тщательно продумывать последовательность операторов, и его усилия целиком сосредоточены на описании закономерностей (правил и фактов) предметной области и на конструировании целевых запросов к данной базе знаний;
- программу на языке Пролог относительно легко модифицировать при необходимости внесения новой информации о предметной области, так как отдельные правила и факты не связаны между собой общими именами переменных.

Пора обсудить пример. Если вас попросят написать программу поиска минимума среди некоторого списка (одномерного массива) значений, то ваши размышления могут привести к следующему результату.

```
Min = Cells(1, 1)
For i = 2 To 10
  If Cells(i, 1) < Min Then Min = Cells(i, 1)
Next
Cells(11, 1) = Min
```

В данном случае вы видите VBA код, выполненный в Excel`е. Все значения располагаются в первом столбце в диапазоне строк от 1 до 10, ответ заносится в ячейку Cells(11, 1). Суть размышлений такова: возьмем первое значение из списка и назначим его текущим минимальным; далее будем сравнивать текущее минимальное значение последовательно со всеми значениями из списка и если какое-то из них будет меньше текущего минимального, то назначим уже его текущим минимальным; так будем

продолжать до тех пор, пока список не кончится; когда список кончится, то текущее минимальное назовем просто минимальным значением списка и зафиксируем его.

Мы последовательно и в деталях объяснили машине как решать задачу.

Давайте сравним этот подход с реализацией на Прологе:

```
min([H|T],MT):-min(T,MT),MT<=H,!.

min([H|_],H).
```

В переводе на русско-алгоритмический это будет звучать примерно так: если минимум в конце списка меньше первого элемента списка, то результатом будет минимум из конца списка, иначе, первый элемент и есть искомый результат.

По существу мы не рассказали машине как решать задачу, мы лишь передали как мы понимаем что есть минимум списка. Иначе говоря, мы описали отношение между объектами предметного поля. В этом описании (декларации) указаны признаки минимума, отличающие его от других значений (максимум, среднее арифметическое и т.п.). Кроме того, обратите внимание на то, что в лексике Пролога нет команды min. Вместо min можно написать всё, что угодно. К примеру, nim или mother и программа будет работать с таким же результатом. Программист волен давать предикатам любые имена, ограничиваясь лишь удобством последующего чтения и понимания существа программы.

В Прологе мы должны не пытаться описать поиск решения задачи, а пытаться описать постановку задачи. Чтобы сделать это руководствуйтесь правилом: не пытайтесь описать инструкции поиска решения, предположите, что вы уже нашли решение, а ваша задача только проверить, что решение найдено.

1.3 Области применения языка Пролог

Пролог слабо приспособлен для решения задач, связанных с обработкой графики, вычислениями или численными методами. Вместе с тем он с успехом может использоваться в компьютерной алгебре, которая в отличие от численных методов, занимается реализацией аналитических методов решения математических задач на компьютере и предполагает, что исходные данные, как и результаты решения, сформулированы в

аналитическом виде. В качестве основных областей применения Пролога можно отметить следующие направления:

- быстрая разработка прототипов прикладных программ;
- управление производственными процессами;
- создание динамических реляционных баз данных;
- автоматический перевод с одного языка на другой;
- создание естественно-языковых интерфейсов;
- реализация экспертных систем и оболочек экспертных систем;
- системы автоматизированного проектирования;
- создание пакетов символьных вычислений;
- доказательства теорем и интеллектуальные системы, в которых возможности языка Prolog по обеспечению дедуктивного вывода применяются для проверки различных теорий.

Prolog нашел применение и в ряде других областей, например, при решении задач составления сложных расписаний. Он используется в различных системах, но обычно не в качестве основного языка, а в качестве языка для разработки некоторой части системы. Достаточно часто Prolog используют для написания функций взаимодействия с базами данных.

Области, для которых Пролог не предназначен:

- большой объем арифметических вычислений (обработка аудио, видео и т.д.);
- написание драйверов.

2 ЯЗЫК ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ ПРОЛОГ. ОСНОВНЫЕ КОНСТРУКЦИИ

В математической логике теории задаются при помощи аксиом и правил вывода. То же самое принято и в языке Пролог.

В нем аксиомы – это факты,

Факты, принято представлять как правила с пустой "посылкой" (1).

с. (1)

Факты в программе – это данные, которые всегда истинны и не требуют доказательства. Данные в фактах могут быть использованы для логического

вывода. Факт может описывать некоторые свойства объекта или отношения между объектами. Факт – это такое свойство объекта или отношения между объектами, которое всегда истинно.

Примеры фактов:

%объект яблоко – красное

red (apple).

%пункты A и B связаны дорогой

road (a, b).

Правила

Под правилом обычно понимается следующая конструкция:

ЕСЛИ <антецедент> ТО <консеквент>

Здесь <антецедент> обозначает условие, которое активирует “зажигает” правило. Это условие может представлять собой логическое выражение с использованием логических операций “И”, “ИЛИ”, “НЕ”. Часть правила <консеквент> обозначает утверждение, которое выполняется, если антецедент истинен. Рассмотрим пример правила медицинской диагностики:

ЕСЛИ болит горло

И температура высокая

И белый налет на миндалинах

ТО вероятный диагноз ангина

На языке Пролог данное правило может быть записано следующим образом:

```
диагноз-ангина :- болит_горло, температура_высокая,
                  белый_налет_на_миндалинах.
```

Правила в Прологе ограничены по форме до так называемых "дизъюнктов Хорна" – утверждений вида $A \Leftarrow B_1 \& \ldots \& B_n$. В программе на Прологе такие утверждения записываются в виде (2).

A :- B_1, ..., B_n. (2)

Правило состоит из левой и правой частей, разделенных двумя символами :- , которые читаются как "при условии, что". Левая часть правила A называется заголовком правила. Правая часть правила вывода – последовательность B_i называется телом правила. Тело правила может

состоять из нескольких подцелей, перечисленных через запятую или точку с запятой. Запятой обозначается операция "логическое И", точкой с запятой – операция "логическое ИЛИ".

Головная цель правила истинна (доказана), если истинны (доказаны) все подцели в теле правила, перечисленные через запятую, или если истинна хотя бы одна группа подцелей в теле правила, среди перечисленных через точку с запятой.

Примеры правил:

```
% Объект, обозначенный переменной Animal, является ласточкой,
%если объект умеет летать и летает низко
swallow (Animal) :-  fly (Animal), flying_low (Animal).

%Объект, обозначенный переменной Animal, является орлом,
%если объект умеет летать, он хищник и летает высоко
eagle (Animal) :- fly (Animal), predator (Animal), flies_high (Animal).
```

Факт является частным случаем правила при n = 0, а правило без заголовка – это вопрос (запрос, цель). Правила, факты и вопросы имеют общее название – предложения.

В предложениях могут использоваться переменные. В конце предложения обязательно ставится точка.

Для приведенного выше примера правила медицинской диагностики с использованием составных термов, переменной P для обозначения пациента и переменной T для обозначения температуры пациента оно может быть записано:

```
диагноз(P,T,ангина) :- симптом(P,болит_горло), T>37.5,
                       симптом(P,белый_налет_на_миндалинах).
```

Из этого примера видно, что исходная конструкция правила ЕСЛИ … ТО заменена двумя символами :- и правило читается на языке Пролог в обратном направлении – справа налево.

Существуют два способа интерпретации правил.

Первый способ предполагает интерпретацию правила как логического высказывания с условием – правой частью правила и заключением –

головной частью правила. Двоеточие тире – это символ логического следования.

Введенное нами выше правило можно прочитать так: "Для всех объектов Animal, если объект умеет летать, он хищник и летает высоко, то Animal – является орлом". Такой способ прочтения правила называется декларативной (логической) интерпретацией.

Второй способ прочтения правила предполагает декомпозицию его правой части на компоненты, которым соответствуют действия, выполнение которых необходимо для получения результата – головы правила.

Каждый компонент правой части можно рассматривать как вызов процедуры с именем компонента и параметрами – аргументами компонента. Совместно компоненты правой части правила – это тело внешней процедуры, а голова правила – заголовок этой процедуры с формальными параметрами – аргументами головы правила.

Такое прочтения называется процедурной (императивной) интерпретацией правила. Процедурная интерпретация позволяет рассматривать выполнение логических программ как последовательность вызова вложенных процедур.

Запросы (вопросы, цели)

Для того чтобы запустить на выполнение программу, написанную на языке Пролог, к ней нужно обратиться с вопросом.

Вопрос может состоять только из одной цели или включать несколько подцелей. Выполнение программы заключается в доказательстве всех подцелей, входящих в вопрос. Программа считается успешно выполненной, если доказаны все подцели, из которых состоит вопрос.

Часто необходимо формулировать вопросы, относящиеся к некоторой группе объектов. Для построения таких вопросов используют логические переменные, которые служат для обозначения объектов, значения которых неизвестны. Вопрос с переменной :- A(X), означающий "Существует ли такое значение X, что A(X) истинно?", рассматривается как утверждение о существовании такого значения X=k, при котором A(k) истинно. В логике

предикатов подобные утверждения записываются с использованием кванторов существования.

Символы кванторов существования в логическом программировании не используются, но подразумеваются.

Предикаты

Предикат – это функция, обозначающая отношение между объектами, являющимися ее аргументами, которая может принимать только значение истина (true) или ложь (false).

Факт или вопрос записывается с помощью имени некоторого предиката, за которым в скобках, через запятую, записываются его аргументы.

Например, факт

red (apple).

записан с использованием предиката red, имеющего один аргумент “apple”.

Факт

road (a, b).

записан с использованием предиката road, имеющего два аргумента “a” и “b”.

Арностью предиката называется количество его аргументов. Арность предиката road равна 2, что обозначается как road/2.

Переменные

Работа с переменными в Прологе достаточна своеобразна. Если в других, алгоритмических, языках программирования значение переменной, которое было ей присвоено, не изменяется до тех пор, пока не будет выполнено переприсваивание значения, то в Прологе переменная может получить некоторое значение в процессе поиска решения и потерять его, когда начнется поиск нового решения.

Имя переменной дается по следующим правилам: имя должно начинаться с заглавной латинской буквы или символа подчеркивания, после

которых могут следовать латинской буквы, цифры или символы подчеркивания.

Пример:

First_list

X

Person

City

Переменная, не имеющая значения, называется свободной, переменная, имеющая значение - конкретизированной.

В Прологе нет оператора присваивания, его роль выполняет оператор равенства =.

Если записать следующую цель:

…, X=3, …

то как эта цель будет рассматриваться, как сравнение или как присваивание, все зависит от того, получила ли какое-либо значение переменная X к моменту доказательства этой цели. Если переменная X имеет значение (например, равное 6), то оператор равенства = работает как оператор сравнения. Если же переменная X свободна (не имеет никакого значения), то оператор равенства = работает как оператор присваивания. При этом совершенно неважно, слева или справа от знака равенства находится имя переменной. Главное, чтобы она была свободной. С точки зрения программы на Прологе следующие две строки совершенно одинаковы:

…, X=3, …

…, 3=X, …

Самое важное, чтобы переменная X не имела значения. Из вышесказанного вытекает следующая особенность использования переменных в Прологе, нельзя записывать вот так:

…, X=X+3, …

В любом случае такая цель будет ошибочной. Действительно, если переменная X имеет, например, значение равное 10, то предыдущая цель сводится к доказательству цели:

…, 10=10+3, …

что, естественно, доказать нельзя.

Если же переменная X свободна, то нельзя к переменной, не имеющей никакого значения, прибавить 5, и присвоить эту неопределенность той же самой переменной. Как же тогда быть, если нужно изменить значение переменной? Ответ один - использовать новое имя, поскольку переменная, которая появляется в тексте программы впервые, считается свободной, и может быть конкретизирована некоторым значением:

…, Y=X+3, …

При этом опять же не важен порядок записи. Такая цель также будет правильной, и будет выполнять присваивание (конечно, если переменная X конкретизирована некоторым числом):

…, X+3=Y, …

Еще существуют специальные переменные, называемые анонимными. Их имя состоит только из знака подчеркивания. Анонимные переменные используются в случаях, когда значение переменной несущественно, но переменная должна быть использована.

Рассмотрим пример:

parent ("Сергей", "Петр").

parent ("Сергей", "Ирина").

parent ("Лиза", "Петр").

parent ("Лиза ", " Ирина").

Факты описывают родителей и их детей (первый аргумент - имя родителя, второй - имя ребенка). Теперь, если нужно узнать только имена родителей, но не нужны имена детей, к программе можно обратиться с внешним запросом, использовав анонимную переменную:

Goal: parent (Person, _).

Решение в данном случае будет избыточным, поскольку есть четыре факта.

Person=Сергей

Person=Сергей

Person=Лиза

Person=Лиза

4 Solutions

Сравните, если использовать запрос:

Goal: parent (Person, Child).

то результаты будут следующими:
Person=Сергей, Child=Петр
Person=Сергей, Child=Ирина
Person=Лиза, Child=Петр
Person=Лиза, Child=Ирина
4 Solutions

Основные секции программы

Как правило, программа на Прологе состоит из нескольких секций, ни одна из которых не является обязательной. Вот основные секции:

1. DOMAINS - секция описания доменов (типов). Секция применяется только, если в программе используются нестандартные домены.
2. PREDICATES - секция описания предикатов. Секция применяется, если в программе используются нестандартные предикаты.
3. CLAUSES - секция предложений. Именно в этой секции записываются предложения: факты и правила вывода.
4. GOAL - секция цели. В этой секции записывается внутренний запрос.

На первый взгляд, без секций DOMAINS, PREDICATES и GOAL действительно можно обойтись, но как написать программу без секции CLAUSES? Конечно, такая программа не обладает большим количеством возможностей, но принципиально такую программу написать можно. Например:

GOAL
write ("Введите Ваше имя: "), readln (Name), write ("Здравствуйте, ", Name, "!").

Это пример программы, состоящей только из секции GOAL. Используются только стандартные домены, следовательно, отпадает необходимость использовать секцию DOMAINS, нет нестандартных предикатов, следовательно, отпадает необходимость использовать секцию PREDICATES, и, наконец, все цели записаны непосредственно в секции GOAL, следовательно, нет необходимости использовать секцию CLAUSES.

Основные стандартные домены

Доменом в Прологе называют тип данных. В Прологе, как и других языках программирования, существует несколько стандартных доменов, перечислим их:

1. integer - целые числа.
2. real - вещественные числа.
3. string - строки (любая последовательность символов, заключенная в кавычки).
4. char - одиночный символ, заключенный в апострофы.
5. symbol - последовательность латинских букв, цифр и символов подчеркивания, начинающаяся с маленькой буквы или любая последовательность символов, заключенная в кавычки.

Для примера приведем программу, оформленную по всем правилам:

```
PREDICATES
bird (string)
has_wings (string)
can_fly (string)
can_swim (string)

CLAUSES
bird ("журавль").
has_wings ("синица").
has_wings ("пингвин").
can_fly ("синица").
can_swim ("пингвин").
bird (Object):- has_wings (Object), can_fly (Object).

GOAL
bird (Who).
```

Замечание: поскольку в программе не использовались нестандартные домены, не было необходимости использовать секцию описания доменов DOMAINS.

Интерпретация правил и правила вывода

Возможны два способа интерпретации правил. **Первый способ** основан на толковании правил, как логических высказываний. Двоеточие тире при этом используется для обозначения логического следования.

Пример правила:

нравятся_взаимно(X,Y) :- нравится(X,Y), нравится(Y,X).

Введенное нами правило следует понимать так: "Для всех X и Y, если X - нравится Y и Y нравится X, то X,Y - нравятся взаимно". Такой способ толкования обычно называют декларативной интерпретацией правил.

Второй способ толкования правил предполагает расчленение его на простые компоненты в соответствии с действиями, выполнение которых необходимо для получения результата. Последнее правило в таком случае следует понимать так: "Для ответа на вопрос нравятся ли X,Y взаимно друг другу, необходимо ответить на вопрос X нравится ли Y и вопрос Y нравится ли X". Каждое действие, связанное с ответом на вопрос, можно рассматривать как вызов процедуры, имя которой определяется функтором терма, а параметры - его аргументами.

Такое толкование называют процедурной интерпретацией правил. Процедурная интерпретация позволяет рассматривать выполнение логических программ с алгоритмической точки зрения, связывая его с последовательным вызовом процедур.

Для доказательства выводимости вопроса из программы, содержащей правила, воспользуемся обобщенным правилом вывода, называемым modus ponens. Это правило утверждает, что из B и A:- B следует A.

Прежде чем сформулировать это правило вывода в терминах, используемых при построении логических программ, уточним определение примера правила. Напомним, что переменные, встречающиеся в заголовке правила, могут повторяться в термах, образующих тело правила. Все эти переменные связаны квантором всеобщности, поэтому правило A':- B_1'

,B_2',..., B_n' может быть примером правила A:- B_1 , B_2 ,..., B_n , если существует подстановка Q , такая, что AQ = A' и B_iQ = B' для всех 1< i< n.

Теперь можно описать правило вывода следующим образом: из правила R = (A:- B_1 , B_2 ... B_n) и фактов B_1' , B_2' ,... B_n' выводится A' при условии, что A':- B_1' , B_2' ,..., B_n' является примером правила R.

Определим логическую программу. как конечное множество правил. В этом случае вопрос или цель G является логическим следствием программы P , если в P найдется правило с основным примером A :- B_1, B_2 ,..., B_n , где n > 0, таким, что B_1 , B_2 ,..., B_n являются логическими следствиями P и A является примером G.

Заметим, что цель G следует из программы P тогда и только тогда, когда G может быть выведена из P с помощью конечного числа применений обобщенного правила modus ponens.

В общем случае процедура получения ответа на вопрос отражает определение логического следования. В основе этой процедуры лежат многократно повторяемые действия поиска в программе правила, соответствующего текущей цели, и поиска подстановки для построения примера цели.

Выполнение логической программы

Логическая модель, построенная на логике предикатов первого порядка, предполагает в общем случае пропозициональный взгляд на знание, когда в качестве элемента знания рассматривается некоторое утверждение и, соответственно, элементарное знание представляется атомарной формулой языка логики предикатов. Процесс вывода строится как последовательность шагов получения целевой формулы с помощью *правил вывода*, к которым относятся следующие классические правила.

1. Правило Modus Ponens:

$$\frac{A,\ A \Rightarrow B}{B},$$

читается так: если A – выводима и A влечет B, то B – выводимая формула.

2. Цепное правило:

$$\frac{A \Rightarrow B,\ B \Rightarrow C}{A \Rightarrow C},$$

читается так: если формулы $A \Rightarrow B$ и $B \Rightarrow C$ выводимы, то выводима и формула $A \Rightarrow C$.

3. Правило подстановки: если формула A(x) выводима, то выводима и формула A(B), в которой все вхождения литерала x заменены формулой B.

4. Правило резолюций: если выводимы формулы двух дизьюнктов, имеющих контрарную пару, $A \vee c$ и $B \vee \neg c$, то выводима формула дизьюнкта $A \vee B$, полученного из данных двух удалением контрарной пары.

Можно построить процесс вывода с использованием только правил подстановки и резолюции, так как правило Modus Ponens и цепное правило можно рассматривать как частные случаи правила резолюций, интерпретируя формулу $A \Rightarrow B$ как дизьюнкт $\neg A \vee B$.

Механизм вывода Пролога как раз и построен с использованием данной модели и порождает, по существу, процесс поиска в глубину.

Логическая программа начинает выполняться после того, как она вместе с вопросом подается на вход интерпретатора. Интерпретатор представляет собой программу, способную строить логические выводы, как правило, начиная с заданного вопроса, т.е. методом "сверху вниз". Если интерпретатору удается доказать выводимость вопроса из логической программы, то он выводит сообщение об удачном окончании и значения переменных, для которых получено решение. В противном случае он выводит сообщение о неудаче.

Основой построения интерпретатора является процедурная интерпретация правил. Учитывая это обстоятельство, вместо термина логический вывод введем термин вычисление. Для того, чтобы определить этот термин, нам требуется ввести ряд новых понятий.

В общем случае вопрос представляет собой конъюнкцию термов. Доказательство выводимости из программы конъюнкции сводится к доказательству выводимости каждого элемента коньюнкции отдельно. В свою очередь доказательство выводимости терма из программы предусматривает поиск в программе правила, в заголовке которого

используется тот же функтор, что и в рассматриваемом терме, а затем выполнение унификации. Если унификация проходит успешно, то можно переходить к доказательству выводимости тела правила, которое также представляет собой конъюнкцию термов.

Назовем конъюнкцию целей, которую следует доказать на рассматриваемом шаге выполнения программы резольвентой. Вначале резольвента совпадает с вопросом. Если после удачной унификации очередной цели, эту цель заменить телом выбранного правила, то получим резольвенту для очередного шага выполнения программы. Операция, связанная с заменой цели G телом того примера правила, из программы P , заголовок которого совпадает с данной целью, называют *редукцией*. При этом заменяемая цель при редукции называется снятой, а добавляемые цели - производными.

Напомним, что переменные в каждом предложении связаны квантором всеобщности. Это означает, что значения переменных определены в области действия квантора и должны теряться при выходе из этой области. Такие переменные в программировании называют локальными. Чтобы избежать путаницы при использовании переменных с одинаковыми именами в разных правилах, условимся переименовывать переменные всякий раз, когда предложение выбирается для выполнения редукции. Среди новых имен не должно быть имен, ранее использованных при вычислении.

Иллюстрацией данного механизма управления также служит следующий фрагмент программы экспертной системы, в котором в диалоге с пользователем определяется, что задуманное им животное – зебра. Для положительного заключения сначала проверяется, относится ли животное к травоядным и не является ли хищником. Для проверки того, что животное травоядное, на следующем уровне проверяется принадлежность к млекопитающим. И на последнем уровне данная принадлежность проверяется по наличию фактов "Имеет волосы" или "Может давать молоко" в базе данных или из ответов пользователя.

```
animal_is("зебра") :- it_is("травоядное"),
                      not(it_is("хищник")),
                      positive("Имеет","темные полосы").
```

```
it_is("травоядное") :- it_is("млекопитающее"),
                       positive("Имеет","копыта").
it_is("травоядное") :- it_is("млекопитающее"),
                       positive("Может","есть траву").
it_is("хищник") :- positive("Может","есть мясо").
it_is("хищник") :- positive("Имеет","зубы"),
                       positive("Имеет","клыки").
it_is("млекопитающее") :-
                       positive("Имеет","волосы").
it_is("млекопитающее") :-
                       positive("Может","давать молоко").
```

Имеются три неопределенных ситуации, требующие уточнения.

Первая ситуация возникает вследствие неопределенности выбора из резольвенты очередной цели для редукции. Чтобы внести определенность в работу интерпретатора обычно используют правило выбора, согласно которому в качестве очередной цели берется всегда самая левая цель резольвенты. Это правило часто называют *стандартной стратегией выбора цели*.

Вторая ситуация возникает оттого, что в определении не оговорены порядок поиска в программе P правила C_i , функтор в заголовке которого совпадает с функтором цели G_i . Эта неопределенность обычно снимается за счет того, что поиск осуществляется путем последовательного просмотра правил в порядке их написания в программе. Такой способ поиска называют *стандартной стратегией поиска*.

Третья ситуация обусловлена тем, что в программе могут содержаться несколько правил с одинаковыми функторами в заголовках, которые совпадают с функтором цели. Такие правила называют_альтернативными. Если обозначить совокупность альтернативных правил $C_1,C_2,...,$ C_n,то невозможность, например унификации цели G с заголовком правила C_i не означает еще невозможность доказательства вывода G из программы. Заключение о неудаче можно сделать только на основании невозможности унификации цели G ни с одним альтернативным правилом. Таким образом, при работе с альтернативными правилами необходимо предусмотреть

действия, обеспечивающие повторные попытки унификации текущей цели с альтернативными правилами. Такие действия, обеспечивающие повторения называют возвратом или бэктрекингом.

Если же унификация цели G_i оказывается невозможной для последнего альтернативного правила C_m, то необходимо вернуться к цели рассмотренной на предыдущем шаге вычисления G_{i-1} и попытаться доказать ее выводимость, используя другое альтернативное правило (i-1)-го шага.

В общем случае возврат может произойти к любой из целей G_k, $k<i$, из которой с помощью нескольких последовательных редукций была получена цель G . Выполнение возврата к предшествующей цели предполагает уничтожение значений переменных, полученных в результате унификации при выводе цели G_i из G_k .