С.П. Якимов, Е.М. Товбис

ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Красноярск
2017

Министерство образования и науки Российской Федерации
федерального государственного бюджетного образовательного учреждения
высшего образования «Сибирский государственный университет науки и
технологий имени академика М.Ф. Решетнева»

С.П. Якимов, Е.М. Товбис

ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Рекомендовано научно-методическим советом филиала СибГУ
в г. Лесосибирске в качестве конспекта лекций для студентов направления
09.03.01 Информатика и вычислительная техника очной, заочной и очно-
заочной форм обучения

Красноярск
2017

Логическое программирование: Конспект лекций для студентов направления 09.03.01 – «Информатика и вычислительная техника» очной, заочной и очно-заочной форм обучения / С.П. Якимов, Е.М. Товбис - Красноярск.: СибГУ, 2017.- 31с.

Рекомендуется студентам для получения базовых основ в области логического программирования и программирования на языке Turbo Prolog.

Содержание

Введение

В рамках курса «Логическое программирование» изучаются основные понятия логического программирования, синтаксис и семантика конструкций и основных встроенных предикатов языка Турбо Пролог. Изложенный материал ориентирован на студентов, имеющих определенные знания и умения в области информатики, но может быть интересен и для непрофессионалов, желающих получить понятие о логическом программировании.

Предлагаемый теоретический материал предполагается закреплять на лабораторных занятиях. Курс охватывает небольшое количество часов, в связи с чем многие вопросы затрагивает поверхностно. Для более глубокого изучения дисциплины рекомендуется самостоятельная работа по приведенной литературе.

Математическая логика давно используется при анализе компьютерных программ. Однако, непосредственное использование логики в качестве языка программирования, называемое логическим программированием, началось сравнительно недавно. Идейные корни логического программирования лежат в математической логике, где были созданы методы формального описания задач с помощью логических формул и методы формальных доказательств, позволяющие автоматически получать по этим описаниям решения с приемлемой степенью эффективности.

Пролог является языком логического программирования (***PROgramming in LOGic***), предназначенным для решения широкого класса задач, относящихся к проблемам искусственного интеллекта, таких как:

- быстрое прототипирование прикладных программ;
- экспертные системы и исследования в области искусственного интеллекта;
- естественно-языковые интерфейсы;
- символьное решение уравнений
- базы данных;
- разработка приложений, связанных с разработкой и использованием систем автоматизации проектирования, компиляторов и т.п.

Кроме того, Пролог использовался в качестве машинного языка в японском проекте вычислительных систем пятого поколения.

Датой рождения Пролога принято считать 1971 год, когда Алэн Колмэроэ (Марсельский университет) разработал его первую реализацию. Однако пик его популярности приходится на 80-е годы. И связано это с двумя обстоятельствами: во-первых, был обоснован базис этого языка и, во-вторых, в японском проекте вычислительных систем пятого поколения он был выбран в качестве базового для одной из центральных компонент– машины вывода. К сожалению, проект не решил декларированные в нем задачи и, в настоящее

время языки логического программирования часто рассматриваются как экзотический, во многом не отвечающий современным требованиям, инструментарий. В тоже время следует отметить перспективность многих подходов, использованных при разработке различных версий языка. Знакомство с логическими языками признано необходимым атрибутом специалиста в области систем программирования и входит практически во все образовательные стандарты по соответствующим направлениям и специальностям.

Пролог принципиально отличается от традиционных языков программирования в первую очередь тем, что в нем требуется описывать логическую модель предметной области в терминах объектов и отношений между ними без подробного описания алгоритма задачи. Программа на языке Пролог состоит из множества утверждений, каждое из которых является либо фактом (аксиомой) из заданной предметной области или правилом (теоремой), указывающим, как решение связано с заданными фактами или правилами. В настоящее время Пролог реализован для большинства типов микро и мини-ЭВМ. Наиболее известны реализации ***Arity-Prolog***, ***MProlog***, ***Turbo Prolog***. Все эти версии разрабатывались независимо и поэтому сильно отличаются друг от друга. Говорить о каком-либо стандарте языка затруднительно: можно изучать теоретические основы языка и какие-то его конкретные реализации.

Лекция 1 Основные понятия и конструкции

План лекции

1. *Введение*
2. *Основные понятия и определения*
3. *Утверждения и их виды*
4. *Способы описания утверждений в среде Турбо Пролог*

Логическая программа– это множество аксиом (утверждений, не требующих доказательств) и правил, задающих отношения между объектами. Вычислением логической программы является вывод следствий из программы. Основные конструкции логического программирования– *термы* и *утверждения*. Имеется три основных вида утверждений: *факты*, *правила* и *вопросы*.

Факт является простейшим видом утверждения и используется для констатации того, что выполняется некоторое отношение между объектами. Факты содержат информацию, которая в выбранной предметной области

Для обозначения факта используется запись вида «***Q.***»– *предикат*, вслед за которым ставится точка. Предикаты бывают 0-местные (без аргументов) или *n*- местные (с *n* аргументами). Предикат без аргументов обозначается одним именем. Предикат с аргументами обозначается именем, вслед за которым в скобках через запятые записываются все его аргументы. Аргумент предиката– синтаксическая конструкция, называемая *термом*. Определение предиката в обозначениях Бэкуса-Наура:

$\langle предикат \rangle ::= \langle идентификатор \rangle \left[(\langle терм \rangle, \langle терм \rangle)^{+} \right]$

Терм– это либо *константа*, либо *переменная*, либо *структура*.

Переменная– это либо знак «_» (подчеркивание, ASCII код #95), либо последовательность букв и/или цифр, начинающаяся с **прописной** латинской буквы. Переменная «_» называется *анонимной*. Все переменные в Прологе локальны: область их действия ограничивается одним утверждением.

Структура– более сложное, чем константа и переменная, образование, характеризуется именем (функтором) и компонентами– составными частями структуры. Каждый компонент структуры в свою очередь является термом.

По аналогии: факт, представленный в Пролог-программе одноместным предикатом:

man("Адам").

можно интерпретировать как утверждения: «Адам является мужчиной» (объект ***Адам*** обладает свойством ***man***).

Конечное множество фактов образуют простейший вид *Пролог-программы*.

Второй формой утверждения в логической программе являются *вопросы* или *целевые утверждения*. Для записи вопросов будем использовать следующую нотацию:

?– R_1, R_2, … R_n.

Здесь R_i– условия (*цели*), при выполнении которых целевое утверждение не отвергается. Вопрос– это средство извлечения информации из логической программы. С помощью вопроса выясняется, выполнено ли некоторое отношение (*цель*) или совокупность отношений между объектами (*целей*). Простой вопрос состоит из одной цели.

При одной и той же базе данных различные целевые утверждения соответствуют различным решаемым задачам. Поиск ответа на вопрос состоит в том, чтобы определить, является ли вопрос логическим следствием программы.

Логические следствия выводятся различными способами. Простейшее из них– *совпадение*: из наличия в базе данных факта «***P.***» следует истинность целевого утверждения «***?– P.***». По аналогии: из наличия «R_1.», «R_2.», … «R_n.» логически следует истинность «***?–*** $R_1, R_2, \ldots R_n$.». Возможные варианты ответов на вопрос: ***Yes***– в случае его истинности и ***No,*** если в базе данных отсутствуют данные, подтверждающие истинность целевого утверждения.

Так, к приведенному выше примеру можно задать, например, вопрос и получить ответ:

?– man("Адам").

Yes

Переменные служат для обозначения неопределенных термов. С помощью вопроса, содержащего переменную, выясняется, имеется ли такое значение переменной, при котором вопрос будет логическим следствием программы. Положительный ответ на такой вопрос включает в себя все допустимые значения переменной и сообщение ***Yes***. Понятие переменной тесно связано с понятием подстановки.

Подстановкой называется конечное (возможно, пустое) множество Ω пар вида $X_i = t_i$, где X_i– переменная и t_i– терм; $X_i \neq X_j$ при $i \neq j$ и X_i не входит в t_j при любых i и j.

Результат применения подстановки Ω к терму A есть терм, полученный заменой каждого вхождения переменной X в терм A на t для каждой пары вида $X = t \in \Omega$.

Введем еще один способ получения логического следствия – *обобщение*: целевое утверждение A истинно, если существует хотя бы одна такая подстановка Ω в A, что получившийся в результате терм является логическим следствием программы.

При выполнении подстановки Ω в целевое утверждение A переменные, входящие в A принимают конкретные значения или, как говорят, *конкретизируются*. Третье правило получения логического следствия– *конкретизация*: из утверждения A, содержащего переменные, всегда выводится утверждение, полученное в результате выполнения любой подстановки Ω в A.

Правило – это вид утверждения, использующийся для констатации того, что при выполнении некоторых условий, являющихся фактами или правилами, выполняется некоторое отношение между объектами.

Для обозначения правил используется запись вида «Q**:–** $P_1, P_2, \ldots, P_n$», которая интерпретируется, как «цель Q удовлетворяется, если удовлетворяются

подцели (условия) $P_1, P_2, \ldots, P_n, (n \geq 0)$». Предикат Q, стоящий слева от знака «**:—**», называется *заголовком правила*, а предикаты $P_1, P_2, \ldots, P_n$, входящие в условие, составляют *тело правила*. Как следует из определения, факт является частным случаем правила, при $n=0$.

Лекция 2 Механизмы унификации и возврата

План лекции

1. *Представление о достижении цели путем логического вывода из программы*
2. *Механизм унификации*
3. *Механизм возврата*

Центральное место в вычислительной модели логических программ составляет алгоритм *унификации*. Унификация является основой автоматической дедукции и логического вывода в задачах искусственного интеллекта. Введем необходимую для математического описания алгоритма терминологию.

Унификатором двух термов называется подстановка, которая делает термы одинаковыми. Если такая подстановка существует, то термы называют *унифицируемыми*.

Алгоритм унификации находит наиболее общий унификатор двух термов, если такой существует. Если термы не унифицируемы, алгоритм сообщает об *отказе*.

При согласовании целевого утверждения с базой данных делается попытка унификации этого утверждения с каким-либо фактом или правилом путем их сопоставления. В роли образцов при этом выступают термы, являющиеся аргументами сравниваемых предикатов, имена которых, естественно, должны совпадать. При сравнении образцов приняты следующие соглашения:

●**1.** Если оба сравниваемых образца– константы, для успешного сопоставления они должны совпадать:

?– man("Адам").

Yes

●**2.** Если один из сравниваемых образцов не конкретизированная переменная, т.е. переменная, которой не приписано никакое значение, то второй образец может быть любым термом. При этом если этот терм– константа или константная структура, то переменная, выступающая в роли первого образца, конкретизируется: ей приписывается значение, которое является вторым образцом. Если второй образец– другая не конкретизированная переменная, то между первой и второй переменными устанавливается прямая связь: в случае дальнейшего означивания одной из них вторая автоматически получает то же значение. Если же второй образец является структурой, содержащей другие переменные, между ними и исходной переменной устанавливается более сложная «функциональная» связь. Так, сравнение образцов ***homo_sapiens(man(Y))*** и ***homo_sapiens(X)*** приводит к сопоставлению переменной ***X*** и структуры ***man(Y)***. Дальнейшая конкретизация переменной ***Y***, например, в результате подстановки ***Y= “Адам”*** приведет к конкретизации ***X= man(“Адам”)***

●**3.** Если оба сравниваемых образца– структуры, для их успешного сопоставления должны совпадать имена этих структур, а также сопоставляться между собой все компоненты этих структур по приведенным выше правилам.

●**4.** Конкретизированные переменные при сопоставлении ведут себя точно так же, как и их значения– константы или константные структуры.

По-особому осуществляется сопоставление с образцом, содержащим анонимные переменные « _ ». Анонимные переменные не конкретизируются.

Например:

?– pair(_ , _).

Yes

В отличие от:

?– pair(X, X).

No

Редукция цели состоит в ее замене телом того правила из программы, чей заголовок совпадает с данной целью. На каждом этапе редукции имеется некоторая *резольвента* (конъюнкция подцелей), которые следует доказать. Выбираются такая цель в резольвенте и такое предложение в логической программе, что заголовок предложения унифицируем с целью. Вычисление продолжается с новой резольвентой, полученной из старой заменой выбранной цели на тело выбранного предложения и последующим применением наиболее общего унификатора. Вычисление завершается, если резольвента пуста.

Неформально процесс работы логической программы может быть описан следующим образом. Он начинается с исходного (возможно конъюнктивного) вопроса ***G*** и завершается одним из двух результатов: успешное завершение или отказ. Работа развивается с помощью *редукции цели*.

Этапы редукции можно отразить в *протоколе* логической программы, в который включаются результаты применения редукции и наиболее общий унификатор.

Рассмотрим протокол логической программы относительно вопроса:

?– father(S, “Павел I”).

Вопрос редуцируется с использованием предложения ***father(X, Y):- child(Y, X), man(X).*** Наиболее общий унификатор– {***X=S, Y=“Павел I”***}. Применение подстановки дает новую резольвенту ***child(“Павел I”, S), man(S)***. Это конъюнктивная цель. В качестве очередной цели можно выбрать одну из двух. Выбор ***child(“Павел I”, S)*** приводит к тому, что цель унифицируется с фактом ***child(“Павел I”, “Петр III”)***, и вычисление продолжается с заменой ***S*** на ***“Петр III”***. Новая резольвента ***man(“Петр III”)***, совпадает с фактом из базы данных. Больше резольвент, подлежащих согласованию нет. Протокол вычисления приводится ниже

father(S, “Павел I”)
child(“Павел I”, S), ***S= “Петр III”***
man(“Петр III”)
Yes
Yes

Приведенный протокол не является единственным. К такому же решению приводит и другой протокол:

father(S, “Павел I”)
man(S) ***S= “Петр III”***
child(“Павел I”, “Петр III”)
Yes
Yes

В представленной модели работы интерпретатора имеется два выбора: выбор цели для редукции и выбор предложения. Природа этих выборов совершенно различна. Выбор цели произволен, как видно из приведенных выше протоколов, для успешного вычисления несущественно какая цель выбрана.

Выбор же предложения для редукции является недетерминированным. Не каждый выбор приводит к успеху. Например, при ответе на вопрос

?– father(S, “Павел I”).

мы могли на одном из шагов сделать неверный выбор:

father(S, “Павел I”)
child(“Павел I”, S) ***S= “Екатерина II”***
man(“Екатерина II”)
No

Альтернативные выборы, которые может сделать интерпретатор при попытке доказать цель, неявно определяют дерево поиска. Задача интерпретатора попытаться найти в дереве поиска путь, соответствующий доказательству цели. Большинство версий Пролога осуществляет полный обход в глубину конкретного дерева поиска цели.

При этом вместо произвольного выбора цели выбирается самая левая цель, а недетерминированный выбор предложения заменяется последовательным поиском унифицируемого правила и *механизма возврата*.

Приведем подробный протокол ответа на вопрос

?– father(S, “Павел I”).

демонстрирующий работу механизма возврата:

grandfather(S, “Павел I”)
father(S, A)
child(A, S) ***A= "Петр III", S= “Елизавета Петровна”***
man(“Елизавета Петровна”)
No срабатывает механизм возврата к ближайшей альтернативе
child(A, S) ***A= "Алексей Петрович", S= “Петр I”***
man(“Петр I”),
Yes

child(“Петр III”, A= ”Алексей Петрович”)

No срабатывает механизм возврата к ближайшей альтернативе

child(A, S) ***A= "Анна Петровна", S= “Петр I”***

man(“Петр I”)

Yes

child(“Петр III”, A= ”Анна Петровна”)

Yes

Yes

Наличие механизма унификации наряду с механизмом возвратов является характерной особенностью Пролога.

Лекция 3 Предикаты и методы управления Пролог-программой

План лекции

1. *Использование встроенных предикатов «отсечение» и «отказ»*
2. *BAF-метод*
3. *CAF-метод*
4. *UDR-метод*

Часто для управления процессом выполнения Пролог-программ используются два встроенных предиката:

1) «***!***»*– отсечение*, предикат, всегда завершающийся успехом, локально (в пределах текущего правила) отключает механизм возврата. Действие его можно охарактеризовать фразой: «Все, что при попытке согласования данного правила выполнялось, единственно верно и никаких альтернатив этому нет». Введение этого предиката связано со стремлением получить более эффективные программы.

2) *fail– отказ*, тождественно ложный предикат, используется для включения механизма возврата. Действие его можно охарактеризовать фразой: «Все, что до сих пор выполнялось, привело к неудаче, необходимо вернуться к ближайшей альтернативе».

***BAF- мето*д (Backtrack After Fail– возврат после отказа)**

Суть метода заключается в организации повторяющихся отказов некоторой цели и включении механизма возврата для повтора ранее выполненных целей. Пример использования метода:

```
car(“BMW–320”).
Car(“Toyota Carina”).
Car(“Жигули”).
Car(“Москвич–412”).
Car(“Запорожец”)
look_cars:–
car(X), write(X), nl,
fail.
```

Хотя выполнение запроса *look_cars* всегда будет приводить к отказу, в результате многократной унификации будут получены необходимые результаты:

```
?– look_cars
BMW–320
Toyota Carina
Жигули
Москвич–412
Запорожец
No
```

***CAF– метод* (Cut And Fail– отсечение и отказ)**

В некоторых случаях при выполнении программы полезно ограничить поиск по базе данных. При использовании этого приема для организации повторений можно, как и раньше использовать возврат по отказу *fail*, а с помощью отсечения «***!***» запретить его при выполнении определенных условий.

Внесем в предыдущую программу небольшое изменение:

car(“BMW–320”).
Car(“Toyota Carina”).
Car(“Жигули”):– !.
car(“Москвич–412”).
Car(“Запорожец”)
look_cars:–
car(X), write(X), nl,
fail.
Look_cars:–
write(“Москвич и Запорожец– это CARa за грехи наши !”).

До определенного момента программа работает так же, как и в предыдущей версии, но потом она поступает подобно уважающему себя автомобилисту: после Жигулей у нее срабатывает *отсечение альтернатив*:

?– look_cars
BMW–320
Toyota Carina
Жигули
Москвич и Запорожец– это CARa за грехи наши !
Yes

***UDR– метод* (User-Defined Repeat– повторения, управляемые пользователем)**

В отличие от предыдущих методов, в которых количество повторений ограничено общим количеством не опробованных альтернатив, здесь повторения могут выполняться произвольное число раз. Для этой цели очень часто используется следующее рекурсивное определение:

repeat.
Repeat:– repeat.

Следует отметить, что в языке Пролог *рекурсия* является одним из основных методов управления программой. В общем случае рекурсивное определение содержит следующие группы предикатов: 1) предикаты, определяющие нормальный выход из рекурсии, их успешное или безуспешное выполнение не приводит к рекурсивным обращениям; 2) предикаты, работающие на входе в рекурсию, в случае отказа осуществляется прекращение рекурсии; 3) непосредственно рекурсивное обращение; 4) предикаты, не оказывающие непосредственного влияния на рекурсию, в процессе рекурсивных обращений они «выталкиваются» в стек и выполняются лишь по завершении рекурсии. Следует обратить внимание на условие выхода из рекурсии. Рекурсивное правило всегда должно содержать этот предикат.

Рассмотрим использование рекурсивных определений на примере вычисления ***n!***:

factorial(N, 1):– N<=1, !.	Условие выхода из рекурсии
factorial(N, S):–	Рекурсивное правило
N1= N-1,	Предикат, успешное выполнение которого приводит к рекурсии, а отказ– к выходу
factorial(N1, S1),	Рекурсивное обращение
*S= S1*N.*	Предикат, выполняющийся на выходе из рекурсии

Лекция 4 Логика предикатов первого порядка – теоретическая основа логического программирования

План лекции

1.Предикат в логике высказываний. Свойства предикатов

2. Основные равносильности

Понятие предикатов

Рассмотрим предложения, зависящее от параметров: "X не равен Y", "X и Y - коллеги", "'B-G=:Y". Заменяя параметры конкретными значениями, получаем истинные или ложные высказывания. "2 не равно 3" - истина, "5-2=3" - истина, "Иван и Петр - коллеги" , "5 не равно 5" -ложь," 7-4=1"-ложь.

Предложения такого типа называют предикатом**.** Дадим точное определение данного понятия;

Предикатом $P(x_1,...,x_n)$ называется функция, аргументы которой принимают значения на некотором множестве M, а сама функция принимает два значения И и Л. ;

$$P(x_1,...,x_n): M_n \to \{И, Л\}.$$

Если P зависит от n аргументов, то P называют n-местным предикатом. Множество M определяется обычно математическим контекстом. Предикат обозначается обычно большими латинскими буквами. Над предикатом можно производить обычные логические операции,

Пример:

1. P(x) – «x делится на 2»

   Q(x) – «x делится на 3»

   P(x)∧Q(x) – «делится на 2 и 3»

2. Повторять ряд операторов, пока x и y не станут равными, либо прекратить вычисления после 100 повторений. P(x) – «x <> y»

   Q(x) – «n<=100»

   R(x) – «n>100»

a) P(x)∧Q(x) – « x <> y и n<=100»

b) P(x)∨ R(x) – « x <> y и n>100»

В программировании на языке Pascal a) используется в цикле While:

```
While (x<>y) and(n<=100) Do
Begin
<оператор цикла>
End;
```

a b) используется в цикле Repeat:

```
Repeat
<операторы цикла>
Until {x <>y) or (n>100);
```

Правила для кванторов

1. Перенос квантора через отрицание.

Пусть А содержит свободную переменную х:

$\neg(\forall x)A(x) \equiv (\exists x)\neg A(x)$

$\neg(\exists x)A(x) \equiv (\forall x)\neg A(x)$

2. Вынос квантора за скобки. Пусть *A{:г.)* содержит свободную переменную *x, B* не содержит *x.*

$(\exists x)(A(x) \wedge B) \equiv (\exists x)A(x) \wedge B$

$(\forall x)(A(x) \wedge B) \equiv (\forall x)A(x) \wedge B$

$(\exists x)(A(x) \vee B) \equiv (\exists x)A(x) \vee B$

$(\forall x)(A(x) \vee B) \equiv (\forall x)A(x) \vee B$

•3. Перестановка одноименных кванторов.

$(\exists x)(\exists y)A(x,y) \equiv (\exists x)(\exists y)A(x,y)$

$(\forall x)(\forall y)A(x,y) \equiv (\forall x)(\forall y)A(x,y)$

Приведенные формулы

Формулу логики предикатов будем называть приведенной, если из логических символов имеются только $\wedge,\vee,\neg$, причем $\neg$ находится только перед символом предиката.

Теорема. *Для* любой формулы существует равносильная ей приведенная формула, причем множества свободных и связанных переменных в ней совпадают.

Приведенная формула называется нормальной, если она не содержит символов кванторов или все символы кванторов стоят впереди.

Теорема. Для любой формулы существует равносильная ей нормальная формула.

Выполнимость. Общезначимость

Формула *A* выполнима в данной интерпретации, если существует набор $<a_1,...a_n>$,где,

$a_i, \in$ М, значений свободных переменных $x_{i1},..,x_{in}$ формулы *A* такой,что $A|<a_1,...,a_n>$ =И

Формула А истинна в данной интерпретации**,** если она принимает значение И на. любом наборе $|<a_1,...,a_n>$, $a_i, \in$ М значений своих свободных переменных $x_{i1},..,x_{in}$.

Формула *A* общезначима или тождественно-истинна, если она истнна в каждой интерпретации.

$\forall xA(x) \supset A(y)$ (1)

$A(y) \supset \exists xA(x)$ (2)

$\exists x\forall yA(xy) \supset \forall y\exists xA(xy)$ (3)

Основные равносильности:

Для любых формул А,В,С справедливы следующие равносильности:

1. Коммутативность $\wedge$

$A \wedge B \equiv B \wedge A$

2. Идемпотентность $\wedge$

$A \wedge A \equiv A$

3. Ассоциативность $\vee$

$A \wedge (B \wedge C) \equiv (A \wedge B) \wedge C$

4. Коммутативность $\vee$

$A \vee B \equiv B \vee A$

5. Идемпотентиостъ $\vee$

$A \vee A \equiv A$

6. Ассоциативность $\vee$

$A \vee (B \vee C) \equiv (A \vee B) \vee C$

7. Дистрибутиввость $\vee$ относительно $\wedge$

$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

8. Дистрибутивность $\wedge$ относительно $\vee$

$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$

9. 1-й закон поглощения

$A \wedge (A \vee B) \equiv A$

10.2-й закон поглощения

$A \vee (A \wedge B) \equiv A$

И.Снятие двойного отрицанания

$\neg\neg A \equiv A$

12.1-й закон де Моргана

$\neg(A \wedge B) \equiv \neg A \vee \neg B$

13.2-й закон де Моргана

$\neg(A \vee B) \equiv \neg A \wedge \neg B$

14.1-я формула расщепления

$A \equiv (A \wedge B) \vee (A \wedge \neg B)$

15.2-я формула расщепления

$A \equiv (A \vee B) \wedge (A \vee \neg B)$

Лекция 5 Списки

План лекции

1. *Определение и структура списка*
2. *Описание списков в среде Турбо Пролог*
3. *Основные операции над списками*

Определение списка

Использование переменных в логических программах отличается от использования переменных в традиционных языках программирования. В логических программах переменная означает неопределенный, но единственный объект, а не некоторую область памяти.

Переменные служат для обозначения неопределенных термов. С помощью вопроса, содержащего переменную, выясняется, имеется ли такое значение переменной, при котором вопрос будет логическим следствием программы. Положительный ответ на такой вопрос включает в себя все допустимые значения переменной и сообщение ***Yes***. Понятие переменной тесно связано с понятием подстановки.

Списки бывают полезны при создании баз знаний и баз данных, словарей, экспертных систем. Списки используются при разработке компиляторов и обработке текстов на естественном языке.

Список – это простая структура данных, широко используемая в нечисловом программировании. По определению список является бинарной (имеющей два аргумента) рекурсивной структурой. Первый аргумент списка– *элемент*. второй рекурсивно определяется как остаток списка. Списки напоминают массивы, используемые в традиционных языках программирования. Однако, в отличие от массива, для списка не нужно задавать его размер (количество элементов). Границы списка определяются лишь размером оперативной памяти. Отличие состоит еще и в том, что к элементам списка нельзя обращаться по индексу.

Для обозначения списка используются квадратные скобки, в которых через запятую перечисляются элементы списка. Например, список целых чисел задается следующим образом:

[1,2,3,4,5]

Списки, состоящие из символов и строк, соответственно:

[‘a’,’b’,’c’]

[“Красный”,“Зеленый”,“Синий”]

Для ограничения рекурсивного определения вводится понятие *пустого списка*, который обозначается [ ]. Исторически общепринятый функтор для списка обозначается «•». Чтобы не перегружать обращение к символу «точка», будем использовать специальную запись. Терм *•(X, Y)*, будет обозначаться [*X*│*Y*]. Элементы списка имеют специальные названия: *X* называется *головой,* а *Y*– *хвостом* списка.

Пролог допускает несколько эквивалентных способов записи списков: запись списков с помощью функтора «•», эквивалентную запись с помощью

квадратных скобок. В такой записи список представляется в виде последовательности элементов, ограниченной квадратными скобками и разделяемыми запятыми.

Порождение списков

Существуют различные способы порождения списков, один из которых, описанный выше, заключается в перечислении его элементов. Для порождения списка также может использоваться встроенный предикат *findall:*

Findall(VarName, Predicate(VarName), ParamList),

где на первом месте стоит переменная, указывающая на аргумент предиката, который стоит на втором месте, и все значения которого будут собраны в результирующий список ParamList.

Основной метод работы со списками

Пролог позволяет отделять от списка первый элемент и отдельно его обрабатывать. Данный метод работает вне зависимости от длины списка до тех пор, пока список не будет исчерпан. Этот метод доступа к голове списка называется методом разделения списка на голову и хвост и является основным методом работы со списком. У списка можно взять несколько голов сразу, для этого нужно перечислить их через запятую:

[X,Y,Z|Tail]

Основные операции над списками

Поскольку списки являются более богатой структурой данных, чем обычные константы, с ними связан набор полезных отношений. В Прологе есть только одна встроенная операция для работы со списками – это деление списка на голову и хвост. Все остальные операции над списками строятся на основе рекурсивного применения этой операции в правилах.

Фундаментальной операцией является определение вхождения отдельного элемента в список:

member(X, [*X*│ _]*).*

member(X,[_│ *Y*]*):– member(X, Y).*

Дополним правило member несколькими фактами:

list([*1, 3, 5, 7, 9*]*).*

list([*2, 4, 6, 8, 10*]*).*

и продемонстрируем, каким образом можно использовать списки на примере конъюнктивного вопроса:

?– list(X), member(8, X).

X= [*2, 4, 6, 8, 10*]

Yes

concat_lists([ ]*, X, X).*

concat_lists([*A*│*B*]*, X,* [*A*│*Z*]*):-*

concat_lists(B, X, Z).

Следующий предикат– сложение списков:

Возможные обращения и варианты ответов:

?– concat_lists([*1, 5, 3*]*,* [*2, 4, 7*]*, L).*

concat_lists([1, 5, 3], [*2, 4, 7*], *L)*

L= [*1*│Z_1]

Z_1= [*5*│Z_2]

concat_lists([5, 3], [*2, 4, 7*], Z_1*)*
concat_lists([3], [*2, 4, 7*], Z_2*)*
concat_lists([ ], [*2, 4, 7*], Z_3*)*
Yes
Yes
Yes
Yes

Z_2= [*3* | Z_3]
Z_3= [*2, 4 ,7*]
Z_2= [*3, 2, 4 ,7*]
Z_1= [*5, 3, 2, 4 ,7*]
L= [*1, 5, 3, 2, 4 ,7*]

После выхода из рекурсии будет получен ответ *L*= [*1, 5, 3, 2, 4 ,7*].

Лекция 6 Структура программы на языке Turbo Prolog

План лекции

1. *Основные разделы программы на языке Пролог*
2. *Типы данных в Прологе*
3. *Отладка программ в среде Турбо Пролог*

Программа состоит из следующих разделов:

[***CONSTANTS***

⟨*раздел объявления констант*⟩]

[[***GLOBAL***] ***DOMAINS***

⟨*раздел объявления структур и типов данных*⟩]

[[***GLOBAL***] ***DATABASE***– [⟨*идентификатор*⟩]

⟨*раздел описания фактов внутренней базы данных*⟩]

[[***GLOBAL***] ***PREDICATES***

⟨*раздел описания структуры фактов и правил*⟩]

[***CLAUSES***

⟨*раздел объявления фактов и правил*⟩]

[***GOAL***

⟨*целевое утверждение*⟩]

Turbo Prolog работает со следующими стандартными типами данных:

п.п.	Тип данных	**Описание**
.	*bt_selector*	указатель индексации внешней базы данных (***B***+дерево ссылок);
.	*char*	одиночный символ (1 байт);
.	*db_selector*	указатель (буфер) внешней базы данных
.	*file*	указатель (буфер) файла
.	*integer*	целое число со знаком
.	*real*	действительное число
.	*ref*	ссылочное число базы данных
.	*reg*	набор регистров микропроцессора
.	*string*	последовательность символов (до 64 К)

0.	*symbol*	строка, занесенная во внутреннюю таблицу символов Пролога

Секция *DOMAINS*– предназначена для определения нестандартных типов данных, структур, термов, указателей файлов и баз данных. Например:

DOMAINS

int= integer объявление стандартного типа под другим именем;

*list_int= integer** объявление списка, состоящего из целых чисел;

*list_string= string** объявление списка, состоящего из строк;

pred= man(string) объявление структуры;

file= f1; f2 объявление указателей файлов ***f1*** и ***f2***;

db_selector= dba1; dba2 объявление указателей баз данных ***dba1*** и ***dba2***;

bt_selector= bt1; bt2 объявление указателей индексации базы данных.

В секции *DATABASE* объявляются факты, которые предполагается менять (добавлять, удалять, редактировать) в процессе работы программы. Все остальные факты и правила объявляются в секции *PREDICATES*. Все факты и правила описываются в разделе *CLAUSES*. Раздел *GOAL* обязателен только в том случае, если предполагается создание загрузочного модуля.

Отладка программ в среде Turbo Prolog позволяет обнаруживать ошибки появляющиеся на стадии выполнении программы. Трассировка бывает пошаговая [F7] и поблочная [F8]. Во время отладки программы возможно наблюдения за изменяющимися значениями переменных и предикатов. Во время работы в среде Turbo Prolog возможно использование встроенной системы помощи, вызываемой клавишей [F1]. Возможен также быстрый вызов справки по ключевым словам нажатием сочетания клавиш [Ctrl]+[F1].

Лекция 7 Программирование встроенных баз данных

План лекции

1. *Основные понятия теории баз данных*
2. *Структура динамической базы данных в Прологе*
3. *Предикаты работы с динамической базой данных*

Программы баз данных (БД) на Турбо Прологе есть частный случай систем управления базами данных (СУБД). Данные в БД представляют собой набор фактов, записанных в доступной для компьютера форме.

Существуют три хорошо известных модели организации БД: *иерархическая, сетевая* и *реляционная*. Логические программы можно рассматривать как мощное расширение реляционной модели БД. Дополнительные возможности возникают здесь за счет применения правил и наличия более гибких средств представления и структурирования данных (структуры, списки и т.д.).

В Турбо Прологе имеются средства для работы с двумя разновидностями БД: внутренней или динамической (ДБД) и внешней (ВБД).

ДБД состоит из фактов, которые существуют в ОЗУ только во время работы программы. Однако их можно сохранить в файле на диске. Такая организация ДБД делает работу с ней более эффективной, но ее размер ограничен возможностями компьютера.

Предикаты, соответствующие фактам из БД, должны быть объявлены в специальном разделе DATABASE. Например, чтобы хранить факты об игроках и соревнованиях, можно поступить следующим образом:

```
DATABASE
        player(name, team, pos)
        competition(id, place, time)
```

Синтаксис предикатов, объявляемых в разделе DATABASE, полностью совпадает с синтаксисом предикатов, объявляемых в разделе PREDICATES. Отличие между предикатами ДБД и обычными предикатами заключается в следующем:

- факты из ДБД, в отличие от обычных, могут быть добавлены и удалены во время работы программы;
- предикаты ДБД могут быть использованы как аргументы специальных встроенных предикатов, работающих с ДБД

Если ДБД имеет имя, оно указывается после символа «-». Если ДБД не именована, ей будет дано имя dbasedom.

```
DATABASE – mydatabase
```

Эта запись аналогична следующей:

```
DOMAINS
        mydatabase= player(name, team, pos)
        competition(id, place, time)
```

Основные предикаты для работы с ДБД:

assert(факт, имя ДБД) – добавляет факты в ДБД после существующих фактов

asserta(факт, имя ДБД) – добавляет факты в ДБД перед существующими фактами
assertz(факт, имя ДБД) – добавляет факты в ДБД после существующих фактов
retract(образец, имяДБД) – удаляет из ДБД утверждение, которое сопоставимо с образцом
retractall(образец, имяДБД) – удаляет из ДБД все утверждения, которые сопоставимы с образцом
save(имя файла, имя ДБД) – сохраняет содержимое БД в текстовом файле на диске
consult(имя файла, имя ДБД) – загружает в память содержимое текстового файла
В ДБД Турбо Пролога, в отличие от других реализаций языка Пролог, могут содержаться факты, но не правила.
Требования в файлу с фактами ДБД:

- синтаксис предикатов из файла соответствует описаниям из раздела DATABASE
- описания фактов не содержат специальных символов (перевод строки, табуляция) и пробелов
- отсутствуют пустые строки
- отсутствуют комментарии
- значения всех объектов типа symbol представлены в виде строк.

Факты, используемые в логической программе, можно вводить с клавиатуры при помощи предиката *readterm*.

Лекция 8 Программирование внешних баз данных

План лекции

1. *Понятие и структура внешней базы данных*
2. *Понятие индексации, B+ дерева*
3. *Основные предикаты работы с внешней базой данных*

Внешняя база данных создается в случае, если объем данных больше объема свободной части ОЗУ или предполагается значительное расширение БД.

При работе с ВБД используются 4 стандартных домена:

- db_selector – домен для объявления селектора (имени) ВБД
- bt_selector – домен для объявления селектора B+ дерева
- place – место расположения ВБД
- ref – адрес в цепочке, по которому расположен терм

ВБД может быть расположена в файле на диске (in_file) или в оперативной памяти (in_memory). ВБД состоит из двух компонент: *элементов данных* (термы Турбо Пролога) и *цепочек* (chain), в которых хранятся термы. В цепочке может храниться неограниченное количество термов. В ВБД может быть любое число цепочек. Цепочка выбирается по имени. Имя цепочки – это просто строка символов. Для быстрого поиска данных цепочка может быть индексирована методом B+дерева.

Имена предикатов, работающих с ВБД, построены следующим образом: <тип объекта данных>_<операция>, например, db_delete, term_delete.

Объявление имени ВБД задается в разделе DOMAINS:

DOMAINS:

db_selector=имя базы1; имя базы2;…

Индексация цепочек методом B+дерева

При индексации методом B+дерева с каждым ссылочным числом связан ключ. Так как это число ссылается на уникальную запись (вход) БД, нахождение правильного ключа быстро приводит к искомой записи данных. B+дерево(индексная цепочка) создается с помощью предиката

bt_create(имя_БД, имяB+дерева, селектор-B+дерева, длина_ключа, длина_узла).

Здесь третий аргумент является выходным и используется для идентификации B+дерева. Длина ключа – это длина самого длинного ключа, по которому осуществляется поиск записи. Аргумент длина_узла определяет, сколько ключей запоминается в каждом узле B+дерева. Для БД среднего размера рекомендуется выбирать длину узла=4.

B+дерево разбито на отдельные страницы или узлы. Каждый узел является либо последним, либо порождает два низлежащих узла. Каждый узел содержит группу ключей из заданного диапазона. Так как никакие два узла не содержат одинаковые значения ключей, можно быстро проверить, не находится ли

искомый ключ внутри диапазона конкретного узла. Если да, то быстро находится ссылочное число, если ключ меньше, то поиск ведется по левой ветви B+дерева, если больше – по правой.

Заключение

В результате освоения материала студент приобретает знания о теоретической основе логического программирования – логике предикатов первого порядка, основных способах задания семантики логических программ, их формальной спецификации и верификации; современных методах и средствах разработки алгоритмов и программ, приемах логического программирования, способах записи алгоритма на логическом языке, способах отладки, испытания и документирования программ. Курс учит навыкам логического мышления, развивает способности к алгоритмизации, помогает в практическом структурировании программного материала.

Знания, полученные при изучении курса логического программирования, могут быть применены в дальнейшем при изучении таких дисциплин информационной направленности, как теории баз данных. теории алгоритмов и структур данных.

В практической деятельности логическое программирование используется прежде всего для построения систем искусственного интеллекта. В 1950 году Алан Тьюринг в статье "Вычислительная техника и интеллект" (книга "Может ли машина мыслить?") предложил эксперимент, позднее названный "тест Тьюринга", для проверки способности компьютера к "человеческому" мышлению. В упрощенном виде смысл этого теста заключается в том, что можно считать искусственный интеллект созданным, если человек, общающийся с двумя собеседниками, один из которых человек, а второй — компьютер, не сможет понять, кто есть кто. То есть в соответствии с тестом Тьюринга, компьютеру требуется научиться имитировать человека в диалоге, чтобы его можно было считать "интеллектуальным". Так, Джозефом Вейценбаумом в лаборатории искусственного интеллекта массачусетского технологического института в 1966 году была создана программа «Элиза». Программа имитировала разговор психотерапевта с пациентом. В 1977 г. Кеннет Колби, основываясь на принципах организации "Элизы", создал программу, которая подобным образом вводила в заблуждение уже не клиентов психиатров, а самих докторов. В 1996 г. Грег Гарви создал программную модель католического исповедника, которая опиралась на те же идеи, что и "Элиза".

Еще один пример применения Пролога в области искусственного интеллекта - создание экспертных систем. Экспертными системами обычно называют программы, которые могут заменить эксперта в какой-то предметной области. В идеале экспертная система должна уметь объяснять пользователю свое решение, а также почему она задает тот или иной вопрос. В октябре 1981 года Японское министерство международной торговли и промышленности объявило о создании исследовательской организации — Института по разработке методов создания компьютеров нового поколения. Целью данного проекта было создание систем обработки информации, базирующихся на знаниях. Предполагалось, что эти системы будут обеспечивать простоту управления за счет возможности общения с пользователями при помощи

естественного языка. Эти системы должны были самообучаться, использовать накапливаемые в памяти знания для решения различного рода задач, предоставлять пользователям экспертные консультации, причем от пользователя не требовалось быть специалистом в информатике. Появление таких систем могло бы изменить технологии за счет использования баз знаний и экспертных систем. Основная суть качественного перехода к пятому поколению ЭВМ заключалась в переходе от обработки данных к обработке знаний. Японцы надеялись, что им удастся не подстраивать мышление человека под принципы функционирования компьютеров, а приблизить работу компьютера к тому, как мыслит человек, отойдя при этом от фон неймановской архитектуры компьютеров. Поставленные цели в полной мере так и не были достигнуты, однако этот проект послужил импульсом к развитию нового витка исследований в области искусственного интеллекта и вызвал взрыв интереса к логическому программированию. В настоящее время Пролог остается наиболее популярным языком искусственного интеллекта в Японии и Европе.

Библиографический список

Основная литература

1. Ездаков, А. Л. Функциональное и логическое программирование [Текст]: учеб. пособие / А. Л. Ездаков. - 2-е изд. - М.: БИНОМ. Лаборатория знаний, 2011. - 119 с.
2. Прыкина, Е. Н. Основы логического программирования в среде Турбо Пролог [Электронный ресурс]: учеб. пособие / Е. Н. Прыкина. - Кемерово: КемГУКИ, 2006. - 68 с. – Режим доступа: http://biblioclub.ru/index.php?page=book_red&id=227891&sr=1
3. Рублев, В. С. Языки логического программирования [Электронный ресурс] / В. С. Рублев. - М.: Интернет-Университет Информационных Технологий, 2008. - 115 с. – Режим доступа: http://biblioclub.ru/index.php?page=book_red&id=234653&sr=1

Дополнительная литература

4. Программирование и основы алгоритмизации [Электронный ресурс]: учебное пособие / В. К. Зольников, П. Р. Машевич, В. И. Анциферова, Н. Н. Литвинов. - Воронеж: Воронежская государственная лесотехническая академия, 2011. - 341 с. - Режим доступа: http://biblioclub.ru/index.php?page=book&id=142309
5. Ин, У. Использование Турбо Пролога / У. Ин, Д. Соломон. – М.: Мир, 1987. – 608 с.
6. Искусственный интеллект. В 3-х кн. Кн. 3. Программные и аппаратные средства: справочник / под ред. В.Н. Захарова, В.Ф. Хорошевского. – М.: Радио и связь, 1990. – 368 с.
7. Стерлинг, Л. Искусство программирования на языке Пролог / Л. Стерлинг, Э. Шапиро – М.: Мир, 1990. – 235 с.
8. Логическое программирование [Электронный ресурс]: электронный учеб.-метод. комплекс / сост. П.А. Егармин. – Лесосибирск, 2012. – Режим доступа: http://www.lfsibgu.ru/elektronnyj-katalog