Содержание

Предисловие

Современное декларативное программирование благодаря развитию вычислительной техники все более активно используется в области разработки прикладного программного обеспечения. Декларативные языки программирования, логические и функциональные, наиболее близки к человеческому мышлению. Программы на декларативных языках, как правило, не просто короче, чем программы на императивных языках; они также позволяют значительно сократить время разработки компьютерного приложения. В них описывается, что нужно сделать, на языке отношений в логических языках или на языке функций в функциональных, но не детализируется как. Знание основных принципов логического программирования и умение применять их в практике программирования полезно любому программисту.

Предмет настоящего учебника – логическое программирование и его использование для разработки современных компьютерных приложений.

Учебник предназначен для студентов магистратуры, которые изучают методы разработки и анализа интеллектуальных систем. Он может быть полезен студентам бакалавриата и специалитета, а также аспирантам, которые специализируются в области информационных технологий, прикладной математики и информатики, программной инженерии и других областях, так или иначе связанных с программированием.

Цель учебника – изложение основных теоретических положений и формирование практических умений по использованию логического программирования в профессиональной деятельности, в частности, приобретение навыков программирования на языке Пролог – основном языке логического программирования.

Основу учебника составляет курс «Логическое программирование», который автор в течение многих лет преподавал студентом Российского государственного гуманитарного университета, специализирующимся в области разработки интеллектуальных систем.

Исторически первым возникло императивное программирование, сначала на языках низкого уровня, затем высокого. Когда предметом активных исследований стали задачи искусственного интеллекта – символьные вычисления, анализ текстов на естественном языке, развитие банков данных и знаний, экспертные системы, проблемы биоинженерии, создание роботов и многие другие, появилась необходимость разработки специализированных языков программирования высокого уровня, удобных для их решения. В результате возникла декларативная парадигма программирования.

Императивная парадигма программирования основана на модели архитектуры ЭВМ, предложенной фон Нейманом в 40-е годы. XX в., поэтому программирование в императивном стиле позволяет использовать ресурсы компьютера наиболее эффективно. Теоретической моделью императивного программирования является машина Тьюринга. Первоначально программы занимались в основном разного рода вычислениями. Проблемы искусственного интеллекта стали причиной появления парадигмы функционального программирования.

Функциональное программирование основано на математических функциях. Теоретической основой функционального программирования является лямбда-исчисление. В функциональных языках имеется возможность отделить определение функции от ее имени; в этом случае функция определяется лямбда-выражением. Успехи, достигнутые в области автоматического доказательства теорем, привели к тому, что возникла парадигма логического программирования.

Логическим программированием называют программирование, основанное на продукционных правилах и использующее язык символьной логики. Логическое программирование состоит в описании задачи с помощью аксиом и применении автоматического логического вывода на основе этих аксиом. Теоретической основой логического программирования является исчисление предикатов первого порядка.

Современные исследования в области автоматического доказательства теорем начались с создания Г. Фреге исчисления предикатов первого порядка (1879). В первой трети XX в. были разработаны процедуры формального доказательства теорем (Ж. Эрбран, Т. Скулем, К. Гедель). В середине 50-х годов появились первые попытки программирования процедуры доказательства. В середине 60-х годов появился метод резолюций – эффективный метод логического вывода (Дж. Робинсон). Примерно в то же время В.Ф. Турчин создал функциональный язык РЕФАЛ, использующий концепции сопоставления с образцом и переписывания термов. В основе модели вычислений этого языка лежит нормальный алгоритм Маркова. В конце 60-х годов в США К. Хьюитт разработал первый язык логического программирования Planner.

Язык Пролог был создан в Марселе в начале 70-х годов прошлого века Аланом Колмероэ. Первый интерпретатор языка Пролог написал Ф. Руссель. После создания в конце 70-х годов в Эдинбурге Д. Уорреном и Ф. Перейрой компилятора языка Пролог, написанного в значительной степени на нем самом, Пролог стал языком практического программирования. Эдинбургский Пролог был положен в основу стандарта языка Пролог, опубликованного в 1995 г.

В начале 80-х годов начался новый этап развития логического программирования, связанный с тем, что язык Пролог был положен

в Японии в основу проекта ЭВМ пятого поколения. В середине 80-х годов XX в. в Копенгагене Л. Йенсен, Ф. Гронсков и Дж. Хофман разработали компилятор Turbo Prolog для персональных компьютеров. Статическая типизация, которая использовалась в языке Turbo Prolog, позволила транслировать программы непосредственно в машинные коды. На этом языке стали разрабатываться крупные коммерческие системы. Несмотря на несоответствие стандарту, система Turbo Prolog быстро стала популярной во всем мире, в том числе в России. В середине 90-х годов появилась система Visual Prolog. Современная версия языка Visual Prolog разрабатывается с начала 2000 годов.

Система программирования Visual Prolog обладает всеми средствами для быстрой разработки современных приложений. Она предоставляет возможность сочетать логическое, функциональное и объектно-ориентированное программирование. В настоящее время язык Visual Prolog используется для создания систем управления ресурсами больших комплексов (в частности, аэропортов), обработки текстов на естественном языке, экспертных систем, систем медицинской диагностики и многого другого. Для учебных целей предназначена версия Personal Edition, для разработки коммерческих приложений – Commercial Edition.

Содержание учебника соответствует программе курса «Программирование на языке Пролог для задач искусственного интеллекта» магистерской программы «Когнитивное и программное обеспечение интеллектуальных роботов и программирование интеллектуальных систем» направления подготовки 45.04.04 «Интеллектуальные системы в гуманитарной сфере» и охватывает все темы курса.

Первые восемь глав посвящены введению в программирование на языке Пролог. В первой главе излагаются основные теоретические положения логического программирования. Во второй главе показано, как создавать консольные приложения на языке Visual Prolog. Третья глава описывает, как устроены вычисления, которые осуществляет машина вывода Пролога, в ней обсуждаются механизмы унификации и отката, а также предикат отрицания. В четвертой главе рассматриваются основные средства управления процессом вычислений. Пятая глава связана с созданием внутренних баз данных. В ней рассматриваются циклы, которые управляются только за счет отката, а также средства для написания игр в консоли, в частности, использование цвета и событий мыши. Приводятся примеры программ, которые могут использоваться для проверки знаний учащихся. Шестая глава посвящена рекурсии и рекурсивным алгоритмам; рассматривается задача о ханойской башне; приводится пример моделирования арифметики с помощью логических программ. Седьмая и восьмая главы посвящены технике обработки списков. Вводится понятие

полиморфизма, рассматриваются элементы функционального программирования, появляются анонимные предикаты и предикаты высших порядков. Кроме того, рассматриваются примеры порождения комбинаторных соединений, алгоритмы сортировки списков, а также методы решения логических задач с помощью списков, в частности создается универсальный решатель задач, аналогичных «головоломке Эйнштейна»; приводятся примеры использования бинарных и двумерных массивов. Девятая глава связана с графами и корневыми деревьями. Рассматриваются основные алгоритмы поиска путей на графах и методы обработки деревьев, в том числе красно-черных и цифровых деревьев. Глава 10 посвящена порождающим грамматикам и синтаксическому анализу текстов; в ней создается язык запросов к базе данных, близкий к естественному языку, а также приводится пример универсального разбора. В главе 11 обсуждаются проблемы, связанные с программированием игр; рассматриваются методы минимакса, альфа-бета отсечения и ван Эмдена для реализации хода игрока-компьютера; проводится анализ позиций и партий для игры «крестики-нолики». Глава 12 посвящена методам разработки графического интерфейса пользователя; в частности, в ней рассматривается пример построения графа, а также реализация игры «Жизнь» Джона Конвея.

По окончании каждой главы приводится список упражнений, которые предназначены для закрепления и более глубокого усвоения материала, а также для приобретения навыков программирования на языке Пролог и создания приложений в среде Visual Prolog.

Завершают учебник список литературы и предметный указатель.

Язык Visual Prolog является объектно-ориентированным, тем не менее основное внимание в учебнике уделяется логическому и функциональному стилям программирования. Исключением является глава 12, где эти стили программирования сочетаются с объектным стилем.

Автор выражает глубокую признательность д-ру физ.-мат. наук, проф. Е.М. Бениаминову, д-ру физ.-мат. наук, проф. О.М. Аншакову и д-ру физ.-мат. наук Д.В. Виноградову за полезные обсуждения, а также представителям компании Prolog Development Center (Дания) – Лео Йенсену (Leo Schou-Jensen), Томасу Линдеру Пулсу (Thomas Linder Puls), Карстену Келеру Холсту (Carsten Kehler Holst) — и ее отделения в Санкт-Петербурге – канд. техн. наук В.А. Юхтенко, С.В. Мухину и А.Б. Доронину за предоставление версии Visual Prolog Commercial Edition и за внимание к работе.

1. Основы логического программирования

Основная идея логического программирования состоит в том, чтобы решение задачи сводить к описанию множества исходных утверждений на некотором формальном логическом языке, из которого можно было бы получать необходимые заключения с помощью некоторой системы логического вывода. В качестве формального языка в логическом программировании используется язык логики предикатов, так что логическая программа представляет собой последовательность формул. В классическом логическом программировании эти формулы являются хорновскими дизъюнктами. В языках логического программирования также используются внелогические средства, к которым относятся, например, операции ввода и вывода. Наиболее известным языком логического программирования является язык Пролог.

В настоящей главе вводятся основные понятия классического логического программирования. Рассматриваются декларативная и процедурная семантика логической программы.

1.1. Декларативная семантика логической программы

Логическая программа – это последовательность предложений, которые с помощью формул логики предикатов описывают отношения между некоторыми элементами, или объектами.

1.1.1. Язык логики предикатов первого порядка

Определение. *Алфавит* языка логики предикатов состоит из следующего набора символов:

1) *переменные* – символы, которые в языке Пролог пишутся с прописной буквы, например X, Y, Z, …;
2) *константы*, например «программирование» и 25;
3) *функциональные символы арности n*, которые в языке Пролог называют *функторами*, например f, g, h, …;
4) *предикатные символы арности n*, например p, q, r, …;
5) *логические связки* – символы & (конъюнкция), $\vee$ (дизъюнкция), $\neg$ (отрицание), $\rightarrow$ или $\leftarrow$ (импликация), $\leftrightarrow$ (эквиваленция);
6) *кванторы* $\forall$ (общности) и $\exists$ (существования);

7) *вспомогательные символы*, такие как скобки, запятая, точка, точка с запятой.

Арность предикатного или функционального символа – это неотрицательное целое число, указывающее количество аргументов, с которыми будет использоваться этот символ. Предикатные символы и функторы арности n будут обозначаться в виде p/n или f/n (с синтаксической точки зрения обозначения одинаковы). Символы p и f называются также *именами* предикатов или функторов.

Пусть $\mathcal{A}$ – алфавит языка логики предикатов.

Объекты, между которыми описываются отношения в программе, представляются термами. Понятие терма определяется индуктивно.

Определение. *Терм* над алфавитом $\mathcal{A}$ – это переменная, константа или выражение вида $f(t_1, t_2, \ldots, t_n)$, где f/n – функциональный символ и $t_1, t_2, \ldots, t_n$ – термы над алфавитом $\mathcal{A}$.

Определение. Константы и переменные называются *простыми* термами, остальные термы – *сложными*, или *составными*.

Определение. Терм, не содержащий переменных, называется *основным термом*.

Пусть $\mathcal{T}$ – множество термов над алфавитом $\mathcal{A}$.

Определение. *Формула* определяется индуктивно:

1) выражение $p(t_1, t_2, \ldots, t_n)$, где p/n – предикатный символ из $\mathcal{A}$ и $t_1, t_2, \ldots, t_n$ – термы из $\mathcal{T}$, является формулой;
2) каждое из выражений $(F \,\&\, G)$, $(F \vee G)$, $(\neg F)$, $(F \to G)$, $(G \leftarrow F)$, $(F \leftrightarrow G)$, (F), где F и G – формулы, является формулой;
3) выражения $\forall X(F)$ и $\exists X(F)$, где X – переменная из $\mathcal{A}$ и F – формула, являются формулами.

Ниже скобки иногда будут опускаться. Кроме того, используются следующие приоритеты относительно символов логических связок и кванторов: сначала $\forall$, $\exists$, затем $\neg$, затем $\&$, затем $\vee$, после этого $\to$ ($\leftarrow$) и, наконец, $\leftrightarrow$.

Определение. Подвыражение, входящее в формулу F и являющееся формулой, называется ее *подформулой*.

Определение. Формула, не содержащая переменных, называется *основной формулой*.

Определение. Формула вида $p(t_1, t_2, \ldots, t_n)$, где p – имя предиката, а $t_1, t_2, \ldots, t_n$ – термы, называется *атомарной формулой*.

Определение. Атомарная формула или ее отрицание называются *литералом*.

Атомарная формула называется также *положительным* литералом, а ее отрицание – *отрицательным* литералом.

Определение. Вхождение переменной X в формулу F называется *связанным*, если она входит только в ее подформулы вида $\forall X(G)$ или $\exists X(G)$, где G – формула. Если переменная в формуле не связана квантором, то она называется *свободной*.

Определение. Формула, не содержащая свободных переменных, называется *замкнутой*.

Определение. *Предложение* – это замкнутая формула.

Пусть F – формула и $X_1, X_2, \ldots, X_n$ – все ее свободные переменные. Тогда формула $\forall X_1 \forall X_2 \ldots \forall X_n\ F$ будет коротко записываться в виде $\forall F$. Формула $\forall F$ называется *универсальной* формулой.

Очевидно, что формула $\forall F$ является замкнутой.

Определение. Терм t называется *свободным для переменной* X в формуле F, если после замены всех свободных вхождений X на t все свободные переменные терма t остаются свободными.

Определение. Пусть $\mathcal{F}$ – множество всех формул над алфавитом $\mathcal{A}$. *Языком логики предикатов первого порядка* называется подмножество множества $\mathcal{F}$.

Пример 1. Утверждения «Лебедь – это птица» и «Каждая птица – это животное» на языке логики предикатов первого порядка над алфавитом $\mathcal{A}$ = {"swan", bird/1, animal/1} можно записать соответственно следующим образом:

bird("swan");

$\forall X(\text{bird}(X) \rightarrow \text{animal}(X))$.

Здесь "swan" – константа, а bird и animal – имена унарных предикатных символов.

1.1.2. Семантика формул

Рассмотрим семантику формул. Аналогично тому, как утверждения естественного языка, описывающие реальный мир, могут быть истинными или ложными в этом мире, значение логической формулы

также интерпретируется как истинное или ложное в некотором абстрактном мире, который представляется алгебраической системой. Таким образом, интерпретация определяет соответствие между множеством формул и некоторой алгебраической системой. Непустую область интерпретации называют *доменом*.

Определение. *Интерпретацией* $\mathfrak{I}$ алфавита $\mathcal{A}$ называется пара, состоящая из домена $\mathcal{D}$ и функции, которая ставит в соответствие:

1) константе c – элемент $\mathfrak{I}(c)$ из $\mathcal{D}$;

2) функциональному символу f/n – функцию $\mathfrak{I}(f)$: $\mathcal{D}^n \to \mathcal{D}$;

3) предикатному символу p/n – отношение $\mathfrak{I}(p) \subseteq \mathcal{D}^n$.

Пример 2. Пусть $\mathcal{A}$ = {"swan", bird/1, animal/1}, $\mathcal{D}$ = {*лебедь*, *воробей*, *слон*} и ι – функция, такая что

ι("swan") = *лебедь*;

ι(bird) = {*лебедь*, *воробей*};

ι(animal) = {*лебедь*, *воробей*, *слон*},

где ι(bird) и ι(animal) – унарные отношения на $\mathcal{D}$. Тогда пара ($\mathcal{D}$, ι) является интерпретацией алфавита $\mathcal{A}$.

Значением формулы является элемент множества {*true*, *false*}, который называется *истинностным значением*. Если значение формулы F в интерпретации $\mathfrak{I}$ равно *true*, то используют обозначение $\mathfrak{I} \models F$; если оно равно *false*, то пишут $\mathfrak{I} \not\models F$. Ясно, что значение основных формул зависит только от интерпретации.

Будем использовать символы $\Leftrightarrow$ и $\Rightarrow$ как метасимволы, означающие соответственно «тогда и только тогда» и «влечет».

Определение. *Значение замкнутой формулы* определяется следующим образом:

1) $\mathfrak{I} \models p(t_1, t_2, \ldots, t_n) \Leftrightarrow (\mathfrak{I}(t_1), \mathfrak{I}(t_2), \ldots, \mathfrak{I}(t_n)) \in \mathfrak{I}(p)$;

2) $\mathfrak{I} \models F \,\&\, G \Leftrightarrow \mathfrak{I} \models F$ и $\mathfrak{I} \models G$;

$\mathfrak{I} \models F \vee G \Leftrightarrow \mathfrak{I} \models F$ или $\mathfrak{I} \models G$;

$\mathfrak{I} \models \neg F \Leftrightarrow \mathfrak{I} \not\models F$;

$\mathfrak{I} \models F \to G \Leftrightarrow \mathfrak{I} \models G$ или $\mathfrak{I} \not\models F$;

$\mathfrak{I} \models F \leftrightarrow G \Leftrightarrow \mathfrak{I} \models F$ и $\mathfrak{I} \models G$, либо $\mathfrak{I} \not\models F$ и $\mathfrak{I} \not\models G$,

3) $\mathfrak{I} \vDash \forall X(F(X)) \Leftrightarrow$ для любого элемента d из $\mathcal{D}$ выполняется: $\mathfrak{I}' \vDash F(c)$, где $\mathfrak{I}'$ – интерпретация, полученная из $\mathfrak{I}$ добавлением элемента $\mathfrak{I}'(c) = d$ для $c \notin \mathcal{A}$;

$\mathfrak{I} \vDash \exists X(F(X)) \Leftrightarrow$ для некоторого элемента d из $\mathcal{D}$ выполняется: $\mathfrak{I}' \vDash F(c)$, где $\mathfrak{I}'$ – интерпретация, полученная из $\mathfrak{I}$ добавлением элемента $\mathfrak{I}'(c) = d$ для $c \notin \mathcal{A}$.

Определение. Пусть F – замкнутая формула. Интерпретация $\mathfrak{I}$ называется *моделью* формулы F, если $\mathfrak{I} \vDash F$.

Определение. Замкнутая формула называется *общезначимой*, если в любой интерпретации она имеет значение *true*.

Определение. Пусть P – множество замкнутых формул. Интерпретация $\mathfrak{I}$ называется *моделью* множества P, если она является моделью каждой формулы из P.

Определение. Если модель множества формул P существует, то это множество называется *выполнимым*, иначе *невыполнимым*.

Определение. Пусть P – множество замкнутых формул. Замкнутая формула F называется *логическим следствием* множества P, если каждая модель множества P является моделью формулы F.

Если F является логическим следствием P, то пишут $P \vDash F$.

Утверждение 1. Пусть P – множество замкнутых формул и F – замкнутая формула. Тогда $P \vDash F$ тогда и только тогда, когда множество формул $P \cup \{\neg F\}$ невыполнимо.

Доказательство. Пусть $P \vDash F$ и существует модель $\mathfrak{I}_0$ множества формул $P \cup \{\neg F\}$. Тогда $\mathfrak{I}_0 \vDash F$ и $\mathfrak{I}_0 \vDash \neg F$, что невозможно. Обратно, пусть множество формул $P \cup \{\neg F\}$ невыполнимо. Если и P невыполнимо, то по определению $P \vDash F$. Пусть теперь P выполнимо, но $P \nvDash F$. Тогда $P \vDash \neg F$, что невозможно. □

Определение. Замкнутые формулы F и G называются *логически эквивалентными*, если они имеют одно и то же истинностное значение для всех интерпретаций.

Если F и G логически эквивалентны, то пишут $F \equiv G$.

1.1.3. Правила вывода

Если из множества формул P с помощью некоторых правил получается новая формула F, то говорят, что множество P является *посылкой*, а F – *заключением*. Сами эти правила называются *правилами вывода*.

Например, если $P \models F$, где P – множество замкнутых формул и F – замкнутая формула, то P – это посылка, а F – заключение.

Пример 3. Часто используются следующие правила логического вывода:

1) Modus Ponens, или правило исключения импликации: из посылки $\{F, F \to G\}$ выводится заключение G, что записывается в виде
$$\frac{F, F \to G}{G};$$
2) правило конкретизации: из посылки $\{\forall X\, F(X)\}$ выводится формула $F(t)$, полученная из формулы $F(X)$ заменой всех свободных вхождений переменной X термом t, свободным для переменной X, т. е.
$$\frac{\forall X\, F(X)}{F(t)};$$
3) правило обобщения: из посылки $F(X)$ выводится формула $\forall X(F(X))$, т. е.
$$\frac{F(X)}{\forall X(F(X))}$$

(подразумевается, что X – это произвольный элемент домена).

В математических теориях некоторые формулы могут быть объявлены аксиомами. В качестве логических аксиом берутся некоторые из общезначимых формул, например $F \to (G \to F)$, $F \to (G \vee F)$ или $\neg\neg F \to F$. Каждой аксиоме можно поставить в соответствие правило вывода без посылки, заключением которого является эта аксиома. В дальнейшем мы не будем специально говорить об аксиомах, а можем предполагать, что некоторые правила вывода не имеют посылок.

Определение. Если формулу F с помощью правил вывода можно получить из некоторых формул множества P или формул, которые можно получить из них, то говорят, что F *выводится* из P, и пишут $P \vdash F$.

Пример 4. Пусть P = {bird("swan"), $\forall X$(bird(X) $\to$ animal(X))}. По правилу конкретизации из посылки {$\forall X$(bird(X) $\to$ animal(X))} следует заключение bird("swan") $\to$ animal("swan"). Далее по правилу Modes Ponens из посылки {bird("swan"), bird("swan") $\to$ animal("swan")} следует заключение animal("swan").

Таким образом, формула animal("swan") выводится из множества P с помощью множества правил вывода, приведенных выше.

Определение. Множество правил вывода называется *корректным*, если для любого множества замкнутых формул P и произвольной замкнутой формулы F из $P \vdash F$ следует $P \models F$. Множество правил вывода называется *полным*, если из $P \models F$ следует $P \vdash F$.

1.1.4. Подстановки

Пусть $\mathcal{Var}$ – множество переменных алфавита $\mathcal{A}$ и $\mathcal{T}$ – множество термов над этим алфавитом.

Определение. *Подстановкой* θ называется множество пар термов $\theta = \{X_1/t_1, X_2/t_2. \ldots, X_n/t_n\}$, где $Dom(\theta) = \{X_1, X_2. \ldots, X_n\}$ – множество переменных, так что $X_i \neq X_j$ при $i \neq j$, и $t_1, t_2. \ldots, t_n$ – термы, такие, что $X_k \neq t_k$ для $k = 1, 2, \ldots, n$. Пустая подстановка обозначается ε.

Определение. Подстановка $\theta = \{X_1/Y_1, X_2/Y_2. \ldots, X_n/Y_n\}$, где Y_1, $Y_2 \ldots, Y_n$ – переменные, называется *переименованием переменных*.

Определение. Пусть F – формула. Преобразование $F \mapsto F\theta$, где $F\theta$ – формула, полученная из F одновременной заменой каждой свободной переменной X_i из $Dom(\theta)$, входящей в эту формулу, на терм t_i, такой что $X_i/t_i \in \theta$, называется *применением подстановки* θ *к формуле* F.

Определение. Формула $F\theta$ называется *примером формулы* F. Если формула $F\theta$ является основной, то она называется *основным примером* формулы F.

Определение. Если подстановка θ является переименованием переменных, то формула $F\theta$ называется *вариантом формулы* F.

Пример 5. Пусть $\theta = \{X/\text{"swan"}\}$. Тогда

$$(\text{bird}(X) \rightarrow \text{animal}(X))\theta = (\text{bird}(\text{"swan"}) \rightarrow \text{animal}(\text{"swan"})).$$

Полученная формула является основным примером исходной.

Пример 6. Пусть X – переменная и θ – подстановка. Тогда

$$X\theta = \begin{cases} t, \text{если } X/t \in \theta, \\ X \text{ иначе.} \end{cases}$$

1.1.5. Понятие логической программы

Определение. Формула вида $\forall(C_1 \vee C_2 \vee \ldots \vee C_n)$, где $C_1, C_2, \ldots, C_n$ – литералы, называется *дизъюнктом*.

Если $n = 0$, то дизъюнкт называется *пустым* и обозначается $\square$.

Дизъюнкт $\forall(C_1 \vee C_2 \vee \ldots \vee C_n)$ представляется в теоретико-множественном виде как множество литералов $\{C_1, C_2, \ldots, C_n\}$. Если $B_1, B_2, \ldots, B_l$ и $\neg A_1, \neg A_2, \ldots, \neg A_k$ – все входящие в него соответственно положительные и отрицательные литералы, то этот дизъюнкт также может быть записан следующим образом:

$$\forall(A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_k \rightarrow B_1 \vee B_2 \vee \ldots \vee B_l).$$

Определение. Если дизъюнкт содержит не более одного положительного литерала, то он называется *хорновским дизъюнктом*.

Логическая программа – это конечная последовательность предложений.

Предложение логической программы – это формула вида

$$\forall(A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_k \rightarrow A_0),$$

где A_0 – атомарная формула, $A_1, A_2, \ldots, A_k$ – литералы.

На языке Пролог такое предложение записывается следующим образом:

$$A_0 \text{ :- } A_1, A_2, \ldots, A_k.$$

Предложения в языке Пролог называют *правилами*. В конце правила ставится знак точки.

Формула A_0 называется *заголовком* правила, а формула $A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_k$ – его *телом*.

Таким образом, на языке Пролог правило имеет вид:

заголовок :- тело.

Эквивалентным образом предложение может быть записано в виде

$$\forall(A_0 \vee \neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_k).$$

Тело может быть пустым. Правило с пустым телом называется *фактом*. Таким образом, факт – это формула вида

$$\forall(A_0).$$

Вопрос, или *запрос* к программе, который также называется *целью*, выражается в виде правила без заголовка. Соответственно, запрос – это формула

$$\forall(\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_k).$$

Подформулы A_i называются *подцелями*.

На языке Пролог факт записывается в виде

$$A_0.$$

Запрос, или цель, имеет вид:

$$:- A_1, A_2, \ldots, A_k.$$

Вместо этого также пишут

$$?- A_1, A_2, \ldots, A_k.$$

Пусть $X_1, X_2, \ldots, X_n$ – все свободные переменные формулы

$$\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_k.$$

Тогда эквивалентным образом запрос записывается в виде

$$\neg \exists X_1 \exists X_2 \ldots \exists X_n (A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_k).$$

При вычислении система пытается доказать невыполнимость множества формул программы и запроса, построив контрпример, т. е. найти подстановку, в результате применения которой к *формуле цели* $A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_k$ получается формула, истинная во всех моделях программы. Такая подстановка называется *решением*, или *правильным ответом* на запрос. Таким образом, система ищет логическое следствие программы, которое является примером формулы цели.

Если формула цели не содержит переменных, то ответом программы является “yes” или “no” в зависимости от того, является множество формул невыполнимым или не является. Если формула запроса содержит переменные, то программа находит все решения. Если решений нет, то ответом программы является “no”.

1.1.6. Классическая логическая программа

В классическом случае все A_i в правиле $A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_k \rightarrow A_0$ являются атомарными формулами, так что правило представляет собой хорновский дизъюнкт. Поэтому логическое программирование в узком смысле называют программированием на хорновских дизъюнктах.

Итак, классическая логическая программа – это множество хорновских дизъюнктов. Поэтому всегда существует модель программы. Ниже будет показано, что каждая классическая логическая программа имеет минимальную модель.

Пример 7. Рассмотрим классическую логическую программу *Animals* вида

```
mammal("ox").
mammal("lion").

bird("dove").
bird("eagle").

animal("ant").
animal(X) :-
```

```
    mammal(X).
animal(X) :-
    bird(X).

?- animal(X).
```

Программа состоит из пяти фактов и двух правил:

$$\forall X(mammal(X) \rightarrow animal(X));$$
$$\forall X(bird(X) \rightarrow animal(X)).$$

Ответ на запрос имеет вид:

```
X = "ant";
X = "ox";
X = "lion";
X = "dove";
X = "eagle".
```

Среди вопросов к программе различают частные и общие, простые и составные. *Частные* вопросы – это вопросы, не содержащие переменных. С их помощью можно, например, выяснить, верно ли, что заданные элементы принадлежат отношению:

```
?- animal("lion").
```

В ответе на частный запрос отмечается истинность или ложность цели. Ответом на указанный запрос будет “yes”. Ответом на запрос

```
?- mammal("tiger").
```

будет “no”, так как формула цели не является логическим следствием программы.

Общие вопросы содержат переменные. С их помощью можно, например, найти, какие элементы принадлежат отношению (см. выше).

Если программа находит ответ на запрос, то цель называется успешной, в противном случае – неуспешной.

Простые вопросы состоят из одной цели, составные – из нескольких, их называют подцелями. В составных вопросах подцели соединяются знаками конъюнкции «,» или знаками дизъюнкции «;». В первом случае запрос называется *конъюнктивным*, во втором – *дизъюнктивным*.

Например, следующий запрос является конъюнктивным:

```
?- bird(X), mammal(Y).
```

Ниже приведен пример дизъюнктивного вопроса:

```
?- mammal(X); bird(X).
```

Значения переменных в подцелях, разделенных знаками дизъюнкции, не связаны между собой. Сначала находятся решения подцели, расположенной до первого знака дизъюнкции, затем, независимо от этого, подцели, расположенной между первым и вторым знаками, и т. д.

Переменные в языке Пролог локальны и имеют математический смысл. Их значение сохраняется только на протяжении одного правила. Они не объявляются, в любом предложении их всегда можно переименовать.

Переменные пишутся с прописной буквы или начинаются со знака подчеркивания. Состоят только из знака подчеркивания или начинаются с него анонимные переменные, которые не принимают никаких значений. Если в предложении встречается несколько анонимных переменных, то все они считаются разными переменными, хотя обозначаться могут одинаково, с помощью только знака подчеркивания «_». Например, с помощью приведенного ниже запроса можно узнать о том, имеется ли хотя бы один элемент, удовлетворяющий отношению bird:

```
?- bird(_).
```

Ответом программы *Animals* на данный запрос является “yes”.

1.1.7. Декларативная семантика

Декларативная семантика программы отвечает на вопрос, **что** делает программа. Запрос Q логически следует из программы P, если во всякой интерпретации, или модели, в которой выполняются все формулы из P, выполняется также и Q. Поэтому декларативную семантику называют еще *теоретико-модельной семантикой*. Программа представляет собой теорию (множество замкнутых формул), аксиомами которой являются правила, а само множество объектов с заданными на нем отношениями – модель этой теории. Ниже будет показано, что если аксиомы теории описываются хорновскими дизъюнктами, то среди всех ее моделей существует единственная, с точностью до изоморфизма, минимальная модель. Эта модель строится на эрбрановом универсуме. Ответом на вопрос к программе являются значения переменных, вычисленные в минимальной модели программы. Минимальная модель программы называется ее *декларативным значением*. Перейдем к основным определениям.

Определение. Пусть $\mathcal{A}$ – алфавит, содержащий по крайней мере одну константу. Тогда множество $U_{\mathcal{A}}$ основных термов, построенных из констант и функциональных символов из $\mathcal{A}$, называется *эрбрановым*

универсумом алфавита $\mathcal{A}$. Множество $B_{\mathcal{A}}$ всех основных атомарных формул над $\mathcal{A}$ называется *эрбрановым базисом* алфавита $\mathcal{A}$.

Определение. Пусть P – классическая логическая программа и $\mathcal{A}$ – алфавит, содержащий константы, а также функциональные и предикатные символы, входящие в программу. Если в множество формул программы не входит ни одна константа, то в алфавит добавляется произвольная константа. *Эрбрановым универсумом* U_P и *эрбрановым базисом* B_P *программы* P называются соответственно эрбранов универсум и эрбранов базис алфавита $\mathcal{A}$.

Пример 8. Для программы *Animals* (см. п. 1.1.6) имеем:

$U_{Animals}$ = {"ox", "lion", "dove", "eagle", "ant"};

$B_{Animals}$ = {*mammal*("ox"), *mammal*("lion"), *mammal*("dove"), *mammal*("eagle"), *mammal*("ant"), *bird*("ox"), *bird*("lion"), *bird*("dove"), *bird*("eagle"), *bird*("ant"), *animal*("ox"), *animal*("lion"), *animal*("dove"), *animal*("eagle"), *animal*("ant")}.

Определение. Пусть P – классическая логическая программа. *Эрбранова интерпретация* программы P – это интерпретация $\mathfrak{I}$ с областью интерпретации U_P, такая что

1) если c – константа, то $\mathfrak{I}(c) = c$;
2) если f/k – функциональный символ, то

   $\mathfrak{I}(f)(x_1, x_2, \ldots, x_k) = f(x_1, x_2, \ldots, x_k)$, где $x_1, x_2, \ldots, x_k \in U_P$;
3) если p/n – предикатный символ, то $\mathfrak{I}(p) \subseteq U_P^n$.

Областью эрбрановой интерпретации является эрбранов универсум, который зависит только от построенного по программе алфавита. Поэтому эрбранова интерпретация определяется отношениями на эрбрановом универсуме, которые представляют значения предикатных символов: для предикатного символа p/n

$$\mathfrak{I}(p) = \{(t_1, t_2, \ldots, t_n) \in U_P^n \mid \mathfrak{I} \vDash p(t_1, t_2, \ldots, t_n)\}.$$

Поэтому эрбранову интерпретацию можно рассматривать как подмножество эрбранова базиса: $\{F \mid F \in B_P, \mathfrak{I} \vDash F\}$.

Определение. Эрбранова интерпретация множества замкнутых формул P, являющаяся моделью для каждой формулы из P, называется *эрбрановой моделью*.

Для того чтобы определить, является ли эрбранова интерпретация $\mathfrak{I}$ моделью формулы $\forall F$, достаточно проверить, что все основные примеры формулы F истинны в этой интерпретации. В частности, что-

бы проверить, что формула $\forall(A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_k \to A_0)$ истинна в $\mathfrak{I}$, достаточно показать, что если $(A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_k \to A_0)\theta$ – основной пример этой формулы и $A_1\theta, A_2\theta, \ldots, A_n\theta \in \mathfrak{I}$, то $A_0\theta \in \mathfrak{I}$.

Покажем, что для того чтобы определить, является ли атомарная формула логическим следствием классической логической программы, достаточно проверить, что каждая эрбранова модель этой программы является также эрбрановой моделью этой формулы.

Теорема. Пусть P – классическая логическая программа, G – формула цели (хорновский дизъюнкт, состоящий только из отрицательных литералов) и $\mathfrak{I}'$ – модель множества формул $P \cup \{G\}$. Тогда интерпретация $\mathfrak{I} = \{F \mid F \in B_P, \mathfrak{I}' \vDash F\}$ является эрбрановой моделью этого множества.

Доказательство. Заметим, что $\mathfrak{I}$ является эрбрановой интерпретацией. Далее предположим, что $\mathfrak{I}$ не является эрбрановой моделью. Тогда существует основной пример формулы $A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_k \to A_0$, который не является истинным в $\mathfrak{I}$. Таким образом, формула $A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_k$ является истинной в $\mathfrak{I}$, а формула A_0 – не является. Но тогда это верно и для $\mathfrak{I}'$, по определению $\mathfrak{I}$, что невозможно. Таким образом, $\mathfrak{I}$ является моделью множества формул $P \cup \{G\}$. □

На множестве эрбрановых моделей, рассматриваемых как подмножества эрбранова базиса, существует естественный порядок, который определяется включениями. Покажем, что существует минимальная эрбранова модель.

Теорема. Пусть P – классическая логическая программа. Пересечение $\mathfrak{I}$ произвольного семейства эрбрановых моделей программы P является эрбрановой моделью программы P.

Доказательство. Предположим, что $\mathfrak{I}$ не является эрбрановой моделью программы P. Тогда существует основной пример формулы $A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_k \to A_0$, который не является истинным в $\mathfrak{I}$, так что формула $A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_k$ является истинной в $\mathfrak{I}$, а формула A_0 – не является. Но тогда это верно для некоторого элемента семейства $\mathfrak{I}_i$, что невозможно. □

Пересечение всех эрбрановых моделей классической логической программы является ее минимальной эрбрановой моделью.

Теорема. Пусть P – классическая логическая программа. Минимальная эрбранова модель M_P программы P совпадает с множеством основных атомарных формул, которые являются логическими следствиями программы: $M_P = \{F \mid F \in B_P, P \vDash F\}$.