Московский авиационный институт

(национальный исследовательский университет)

Факультет прикладной математики и физики

Реферат

«Как научить младшего брата логическому программированию»

Работу выполнила

Студентка группы 8О-208Б

Понагайбо Анастасия Олеговна

Москва

2016

У меня есть двоюродный брат, которому 16 лет. В дальнейшем он хочет связать свою жизнь с информатикой и программированием. Тут я опишу, что бы я сказала, если бы он попросил объяснить ему логическое программирование. 

В настоящее время мы не можем представить свою жизнь без какой-либо техники и электроники. Практически у каждого есть собственный компьютер или ноутбук, мобильный телефон, а дома - холодильник, микроволновая печь, телевизор. Вне дома мы также сталкиваемся с различными электронными устройствами: турникеты в автобусах и метро, светофоры, банкоматы и терминалы. Все они управляются заложенными в них программами, которые могут быть написаны на разных языках программирования. Эти языки в свою очередь поддерживают одну или несколько существующих парадигм программирования. Парадигма – это набор идей, методов, подходов и понятий, которые определяют стиль написания программ. Примеры парадигм: объектно-ориентированная, функциональная, логическая.

Основной идеей объектно-ориентированного программирования являются объекты класса. Класс описывает свойства объектов одного типа и возможные действия с ними. Сама программа состоит из совокупности различных объектов, которые взаимодействуют друг с другом.

В функциональном программировании описываются функции, которые могут вызывать другие функции. Все они возвращают некоторое значение, соответствующее конкретным входным данным. Таким образом, для получения результата необходимо описать функцию, при применении которой к входным данным получается требуемое решение.

Программы на логическом языке программирования описывают совокупность фактов и объектов, а также отношения между ними и их свойства. Из них создается база знаний. Кроме того, задаются правила вывода новых фактов из уже существующих. Подробнее про это я хотела бы рассказать на примере самого распространенного языка логического программирования – Prolog.

Чтобы понять, как работать с этим языком, нужно для начала рассмотреть его синтаксис.

Единственным типом данных в Prolog является **терм**. Из термов строятся все факты, правила и запросы. Они могут быть простыми и составными. К простым термам относятся константы и *переменные*, а к составным – *структуры*. Константами являются *атомы* и *числа*.

**Атом.** Атом – это имя, которое не несет никакой смысловой нагрузки. Чаще всего используются для обозначения имен предикатов и аргументов предикатов. Они записываются со строчной буквы с помощью цифр, специальных символов (+ - \* / ^ = : ; ? @ $ &) и цепочек букв. Второй вариант записи допускает любую последовательность символов, заключенную в апострофы, в том числе и с заглавной буквы.

Например:

* atom
* ’Prolog:Al@in-1972’
* new\_Atom2017

**Числа.** В языке Prolog допускается использование целых и вещественных чисел. Их диапазон и точность задаются конкретной реализацией.

**Переменные.** Переменные обозначают произвольные или неизвестные объекты в предикатах. Все переменные являются локальными для каждого предиката, то есть переменные с одинаковыми именами, но встречающиеся в разных предикатах, будут обозначать разные объекты. Все переменные записываются с прописной буквы либо начиная с символа подчеркивания. Могут содержать только цифры, буквы и символы подчеркивания.

Пример названий переменных:

* Someone
* \_an0nym\_

Существуют также *анонимные* переменные, они обозначаются одним символом подчеркивания: \_. Используется тогда, когда имя переменной неважно или оно используется в предикате только один раз.

Например:

has\_pet(‘Mary’, cat).

has\_pet(‘Mary’, dog).

love\_pets(X):- has\_pet(X,\_).

В этом примере мы утверждаем, что кто-то (подразумеваемый под переменной Х) любит животных, если у него есть питомец. Анонимная переменная в данном предикате обозначает, что нам не важно, какое именно домашнее животное у человека.

**Структуры.** Структура в Prolog – это конструкция, представляющая собой *функтор*, после которого в скобках перечисляются его аргументы. Под функтором подразумевается атом, стоящий перед скобками, который является именем этой структуры. В качестве аргументов функтора могут выступать любые термы, в том числе и другие структуры. Количество аргументов называется *арностью*, причем функторы с одинаковым именем, но разной арностью, различаются. Функтор с нулевым количеством аргументов является атомом. Арность записывается в следующем виде: *имя\_функтора*/*арность*. Таким образом, functor/2 и functor/3 обозначают functor(\_,\_) и functor(\_,\_,\_) соответственно.

Примеры структур:

* has\_pet(’Mary’, cat).
* person(name, birth(day,month,year)).

Во втором примере первый аргумент структуры является атомом, а второй – составным термом.

Таким образом, составные термы очень удобны для представления сложных структур данных, так как позволяют записывать неограниченное количество термов, которые могут вкладываться друг в друга.

**Строки**. Еще одним структурным термом являются **строки**. Строкой является последовательность символов, заключенная в двойные кавычки. Она представляет собой список кодов этих символов. К ним применимы все операции списка.

**Список**. Список – это упорядоченный набор термов. Термы записываются через запятую внутри квадратных скобок. Рекурсивное определение списка звучит так: список может быть либо пустым, либо состоящим из *головы* и *хвоста*, причем голова списка – это его первый элемент, а хвост – исходный список без первого элемента.

Примеры списков:

* List = []. %пустой список
* List = [1,2,[3,point(4,5),6],7,8].

При введении списка можно явно задать его голову и хвост с помощью вертикальной черты:

List = [a,b|X].

**Правила и факты.** Тело программ на языке Prolog задается в виде правил и фактов, которые описывают отношения между объектами.

Фактами задаются известные отношения. Они записываются как имя факта и набор его аргументов.

Примеры фактов:

* fruit(apple).
* like(ann,apples).

В этом примере сказано, что яблоко – это фрукт, и что Анна любит яблоки.

Правила состоят из заголовка и тела, разделенных с помощью знаков :- . Тело правила задает условия, при которых будет верен заголовок, то есть заголовок является истинным, если тело правила истинно.

Пример правила:

has\_cat(mary).

has\_dog(mary).

has\_hamster(mary).

has\_pet(X):- has\_cat(X); has\_dog(X); has\_hamster(X).

Сначала мы задаем факты того, что у Марии есть кошка, собака и хомяк. Далее записано правило, как определить, есть ли у человека домашнее животное. Знак ; означает логическое ИЛИ (дизъюнкцию). Для выполнения правила необходимо, чтобы выполнялся хотя бы один из предикатов, разделенных знаками дизъюнкции. Данное правило будет читаться следующим образом: у человека есть домашнее животное, если у него есть кошка ИЛИ собака ИЛИ хомяк. То есть если у человека есть хотя бы один из перечисленных питомцев, то правило has\_pet будет выполняться.

Так же для задания правил используется знак , , который обозначает логическое И (конъюнкцию). Чтобы правило выполнялось, необходимо, чтобы выполнялись все предикаты, разделенные этим знаком.

Пример:

done(ann,labs).

done(ann,coursework).

permit(X):- done(X,labs), done(X,report), done(X,coursework).

В примере говорится, что для допуска к экзамену необходимо сделать лабораторные работы И написать реферат И написать курсовую работу. Если хотя бы одно требование выполнено не будет, то и не будет получен допуск.

В языке Prolog существуют уже встроенные в него предикаты. Вот некоторые из них:

* write(X). Выводит значение Х на экран.
* nl. Осуществляет перевод строки.
* tab(N). Выводит на экран знак табуляции N раз.
* read(X). Связывает с переменной Х введенное с клавиатуры значение.

**Вопросы.** Для получения какого-либо результата нужно сформулировать вопрос программе. Он записывается после символов **?-** по тем же правилам, что и факты с правилами. После ввода вопроса программа будет пытаться доказать или опровергнуть утверждение либо найти конкретный объект, подходящий по условиям.

Рассмотрим предыдущий пример с вопросом:

done(ann,labs).

done(ann,coursework).

permit(X):- done(X,labs), done(X,report), done(X,coursework).

?- permit(ann).

В этом примере программа напишет нам No. Перепишем пример следующим образом:

done(mary,labs).

done(mary,report).

done(mary,coursework).

done(ann,labs).

done(ann,coursework).

permit(X):- done(X,labs), done(X,report), done(X,coursework).

?- permit(X), write(X).

В этом случае программа выведет на экран mary, а после этого No, так как больше никто по условиям допуска к экзамену не подходит.

После получения вопроса программа будет пытаться подставить на место переменных значения из заданных нами фактов и проверять, выполняются ли правила. После нахождения первого подходящего ответа поиск можно остановить или продолжить до тех пор, пока не будут проверены все возможные варианты.

Удобство логического программирования и языка Prolog в частности состоит в том, что мы может сами написать необходимые нам предикаты, причем они будут намного более компактными, чем процедуры в программах, написанных на императивных языках программирования. Нет необходимости продумывать последовательность действий для получения ответа на вопрос, нужно только описать необходимые для поиска решения факты. Кроме того, можно легко находить множество подходящих решений. Еще одним преимуществом языков логического программирования является легкость понимания описанных предикатов, так как они строятся способом, близким к привычному нам языку.

Логическое программирование используется для обработки текстов на естественном языке, в области искусственного интеллекта, в системах управления базами данных.