Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э. Баумана

(национальный исследовательский университет)»

(МГТУ им. Н.Э. Баумана)

Домашнее задание по БКиТ

Реферат

«Язык Пролог»

Студент Яровенко М. В. ИУ5Ц-52Б
(Фамилия И. О.) (Группа)

Преподаватель Гапанюк Ю.Е.
(Фамилия И. О.

2020 г.

Оглавление

Введение

Prolog - это язык логического программирования. Логическое программирование и связанное с ним функциональное программирование в корне отклонились от основного пути развития языков программирования. Логическое программирование не строится с помощью каких-то последовательностей абстракции и преобразований, а строится на основе абстрактных моделей, которые не имеют ничего общего с каким-либо типом модели машины. Логическое программирование основано на убеждении, что людей не следует учить думать в терминах компьютерных операций, но что компьютеры должны следовать человеческим инструкциям.

Логическое программирование — это метод информатики, который использует логику предикатов первого порядка в качестве языка высокого уровня в форме фраз говорящего. Логика предикатов первого порядка — это общий абстрактный язык, предназначенный для выражения знаний и решения проблем. При использовании языка логического программирования главная задача - описать структуру задач приложения, а не предъявлять к компьютеру требования о том, что ему нужно делать.

Язык Prolog основан на небольшом наборе основных механизмов, включая сопоставление с образцом, представление структур данных в виде дерева и автоматическую итерацию с возвратами. Это очень подходит для решения проблемы рассмотрения объектов (особенно структурированных объектов) и взаимосвязи между ними. Задача программы Prolog - доказать, является ли данное целевое предложение результатом существующих фактов и правил.

История создания и развития языка

Название Prolog — это аббревиатура от "Logic Programming".

Разработка языка Prolog началась в 1970 году Аланом Кулмерроу и Филипом Расселом. Их цель - создать язык, на котором можно делать логические выводы из данного текста. Название пролог — это аббревиатура от "Logic Programming". Язык был разработан в Марселе в 1972 году.

Интерес к Prolog несколько раз повышался и падал, а энтузиазм сменялся резким неприятием. Наибольший интерес к прологу как языку будущего был вызван во время разработки японского национального программного обеспечения пятого поколения в 1980-х годах. В то время разработчики надеялись, что с помощью пролога можно будет сформулировать новые принципы и привести к более высокому творчеству. Горизонтальный умный компьютер.

В 1980-х годах язык пролога был включен в учебники по информатике в некоторых советских университетах и школах для изучения математической логики, принципов логического программирования и элементов проектирования баз знаний и моделей экспертных систем. По этой причине перевод пролога на русский язык был реализован на ПК IBM и некоторых советских школьных компьютерах.

Бытует мнение, что Prolog - почти мертвый язык. За короткий промежуток времени (1970-1980) он прошел весь жизненный цикл - от начальной разработки до стремительного роста интереса узких специалистов, а затем его популярность быстро упала. Падший и почти полностью забытый. Такая судьба - обычное явление в мире, где с каждым годом в информационных технологиях появляются новые, более совершенные, проповедующие правильные, теоретически более красивые языки программирования. Сторонники мнения утверждают, что структуру программ на Прологе очень трудно понять из-за того, что иногда невозможно визуально

предсказать ход логического вывода - единственного предиката-факта, спрятанного где-то в конце текста программы, может направить работу в совершенно непредсказуемое русло. В некотором смысле они правы: сложность поддержки программ на Прологе и сложность мышления на Прологе для обычных программистов оказались непреодолимыми препятствиями для его широкого распространения. Конечно, при разработке Prolog от него ожидали большего, но потом оказалось, что сам язык программирования не может помочь решить самые серьезные проблемы, например, связанные с принятием компьютерных решений (машинное мышление).

Но язык Prolog до сих пор используется для решения целого класса довольно специфических задач.

Сфера использования

Каждый язык программирования имеет свой набор задач, при решение которых наиболее эффективно используется. Для Prolog это задачи, связанные с разработкой систем искусственного интеллекта (различные экспертные системы, программы перевода, интеллектуальные игры). Он используется для обработки естественного языка и имеет мощные инструменты для извлечения информации из базы данных. Используемый метод поиска в корне отличается от "традиционного" метода поиска.

Prolog также нашел применение в других областях, таких как решение сложных задач планирования. Однако это не язык программирования общего назначения и не предназначен, например, для решения задач, связанных с графикой или численными методами.

Пролог используется в различных системах, но обычно не как основной язык, а как язык для разработки определенных частей системы. Например, Prolog часто используется для написания функций, взаимодействующих с базами данных.

Синтаксис

Основные концепции в Prolog — это факты, правила вывода и запросы, описывающие базу знаний, процесс рассуждений и принятие решений. В логическом программировании, реализованном в прологе, используется только одно правило вывода.

На языке пролога исходный набор формул для поиска пустого разрешения представлен как так называемые дизъюнкты Хорна:

1. *Термы*

Программа Prolog описывает отношения, определенные с помощью предложений. Как и любой другой язык, ориентированный на символические вычисления, предложения состоят из терминов, которые подразделяются на атомы, числа, переменные и структуры. Атом пишется строчной буквой или заключен в кавычки, когда требуется прописная буква.

Переменные с заглавными буквами отличаются от переменных в процедурных языках программирования; они не связаны с конкретными ячейками памяти, но ближе к математическим переменным.

Структура — это ряд терминов, заключенных в круглые скобки, включая другие структуры. Эта структура представлена именем (функтором), которое стоит перед скобками.

Другая структура — это список, элементы которого заключены в квадратные скобки. Списки в Prolog основаны на связанных списках.

2. *Правила*

Правила в Prolog записываются в виде правил вывода с логическими выводами и списком логических условий. В чистом Прологе предложения ограничиваются дизъюнкциями Хорна.

3. *Факты*

Факты в Prolog описываются логическими предикатами с определенными значениями. Факты в базах знаний на языке Prolog представляют собой конкретную информацию (знания). Обобщенная информация и знания в языке пролога задаются правилами вывода (определениями) и наборами таких правил вывода (определений) над конкретными фактами и обобщенной информацией. Предложения с пустыми телами называются фактами.

Примеры кода на языке Prolog

1. *Пример - поиск совершенных чисел*

Задача: написать программу, которая будет находить все совершенные числа.

решение очевидно, мы перебираем все целые числа и проверяем, идеальны ли они, эта стратегия очень хорошо применима к императивным языкам, мы не замечаем, как сразу ищем алгоритм для поиска решения, и не анализируем проблему. В Прологе мы не должны пытаться описать поиск решения проблемы, а постараемся описать постановку проблемы. Для этого соблюдайте правило:

Не пытайтесь описывать инструкции по поиску решения, предполагайте, что вы уже нашли решение, и ваша задача - только проверить, что решение найдено.

Как ни странно, эта стратегия работает отлично.

```
%% Декларативное определение натуральных чисел
ints(0).
ints(X) :- ints(Y), X is Y + 1.

%% Совершенное число - это 1) натуральное число 2) сумма делителей равна числу
perfect_number(X) :- ints(X), Y is X - 1, calculatesum_divisors_till(Sum, X, Y), Sum =
X.

%% Проверка суммы делителей 1-й аргумент Сумма, 2-й - число для которого ищем делители,
%%  3-е - число до которого ищем делители
calculatesum_divisors_till(0, _NumberToDivide, 0).
calculatesum_divisors_till(Sum, NumberToDivide, Till) :- Till > 0,
        Rem is NumberToDivide mod Till,  Rem = 0,  Ts is Till - 1,
        calculatesum_divisors_till(SumPrev, NumberToDivide, Ts),
        Sum is SumPrev + Till.

calculatesum_divisors_till(Sum, NumberToDivide, Till) :- Till > 0,
        Rem is NumberToDivide mod Till, Rem > 0, Ts is Till - 1,
        calculatesum_divisors_till(Sum, NumberToDivide, Ts).
```

2. *Пример - генерация сочетаний.*

Генерация комбинаций сходна по простоте с генерацией перестановок. Нам нужен предикат member / 2 - членство элемента в списке. Предположим, у нас есть 2 списка: 1-й-исходный список, 2-й-предполагаемая комбинация, нам нужно проверить правильность комбинации. Комбинированные элементы расположены в порядке исходного списка.

```
member(X, [X|_]).
member(X, [_|L]) :- member(X, L).

comb([], []).
%% Вариант 1 : 1-й элемент сочетания содержится в исходном списке
comb([X|List], [X|Tail]) :- comb(List, Tail).
%% Вариант 2 : сочетание является правильным сочетанием хвоста списка,
%%     то есть 1-й элемент исходного списка не содержится в сочетании
comb([_|List], Tail) :- comb(List, Tail).
```

3. Пример – сортировка

Мы рассмотрим этот пример достаточно подробно и попробуем оптимизировать основное решение. процесс написания пролога выглядит следующим образом:

1) первоначальное описание проблемы и получение объемного решения

2) логическая оптимизация перестановкой предикатов справа

3) логическая оптимизация за счет введения упрощенных проверок или удаления ненужных условий

4) внедрение эвристики и оптимизация отдельных кейсов путем отсечения.

Вариант 1. Сортировка наивная: первый элемент отсортированного массива должен быть минимальным, остальные элементы должны быть отсортированы. Первый массив является оригинальным, второй массив

является отсортированным оригиналом.

```
        sort([], []).
        sort(List, [Min|SortRest]) :- min_list_exclude(Min, List, Exclude), sort(Exclude, SortR
est).

        %% Рекурсивно исключаем минимальное число, если в списке одно число исключаем его
        min_list_exclude(M, [M],  []).
        min_list_exclude(Min, [M|L],  ExcludeRes) :- min_list_exclude(Ms, L, Exclude),
                  find_result(result(M, L), result(Ms, [M|Exclude]), result(Min, ExcludeRes)).

        %% Дополнительный предикат для определения пары с минимальным ключом
        find_result(result(M, L), result(Ms, _), result(M, L)):- M < Ms.
        find_result(result(M, _), result(Ms, Exclude), result(Ms, Exclude)):- Ms =< M.
```

Можно отметить, что сложность этого алгоритма квадратичная, и основная проблема в том, что мы каждый раз ищем минимальный элемент, не сохраняя никакой полезной информации.

Также обратите внимание, что мы пытаемся определить, что такое 1-й элемент отсортированного массива.

Вариант 2. Быстрая сортировка.

Давайте посмотрим на проблему со второй стороны и попробуем определить место 1-го элемента списка в отсортированном массиве (применим рекурсию к исходному массиву).

```
        sort_b([], []).
        sort_b([T|R], List) :- split(T, R, Less, Great), sort_b(Less, LessSort), sort_b(Great,
GreatSort),
                                        append(LessSort, [T|GreatSort], List).

        %% Разделяем массив на 2 массива больше и меньше
        split(_, [],[], []).
        split(T, [M|R],[M|Less], Great) :- M < T, split(T,R, Less,Great).
        split(T, [M|R],Less, [M|Great]) :- M >= T, split(T,R, Less,Great).

        %% Склеиваем 2 списка
        append([], M, M).
        append([L|Left], Right, [L|Res]) :- append(Left, Right, Res).
```

Вы можете заметить, что мы улучшили результаты сортировки, поскольку быстрая сортировка, очевидно, быстрее, чем пузырьковая. Чтобы

еще больше улучшить результаты, мы можем вспомнить сортировку слиянием, которая в любом случае дает O (n lg n), но, к сожалению, такая сортировка применима только к массивам, а не к связанному списку, с которым мы работаем. единственный вариант - использовать дополнительную структуру данных для хранения - дерево.

Вариант 3. Сортировка с использованием бинарного дерева.

Для такого рода сортировки давайте переведем исходный список в двоичное дерево, а затем, используя обход дерева слева, получим сортированный массив. Дерево будет представлено рекурсивным деревом терминов (Object, LeftSubTree, RightSubTree).

```
sort_tree([], nil).
sort_tree([X|L], Tree) :- sort_tree(L, LTree), plus(X, LTree, Tree).

%% Добавление в элемента в дерево
plus(X, nil, tree(X, nil, nil)).
plus(X, tree(O, L, R), tree(O, ResL, R)) :- O >= X, plus(X, L, ResL).
plus(X, tree(O, L, R), tree(O, L, ResR)) :- O < X, plus(X, R, ResR).

sort_t(X, Y) :- sort_tree(X, Tree), tree_list(Tree, Y).

append_list([], L, L).
append_list([X|L], R, [X|T]) :- append_list(L, R, T).

tree_list(nil, []).
tree_list(tree(X, L, R), List) :- tree_list(L, ListL), tree_list(R, ListR),
    append_list(ListL, [X|ListR], List).
```

Критика

Пролог критикуется, прежде всего, за его неполный декларативный характер: создание каких-либо сложных и практически полезных Пролог-программ в полностью декларативном стиле практически невозможно, программист вынужден прибегать к процедурным приемам, что приводит к резкому увеличению сложности создания и отладки программ, а также плохой управляемости промежуточными результатами.

Еще одним часто критикуемым свойством языка является отсутствие типизации (в то время как в Visual Prolog - одном из объектно-ориентированных расширений языка - реализована сильная типизация, что, однако, снижает гибкость пролога).

Заключение

В общем целом, очевидно, что за последние несколько лет с момента своего создания Prolog «не достиг высоких целей логического программирования», но, тем не менее, представляет собой мощный, производительный и практически подходящий формализм программирования. Вопрос о том, куда пойдет Prolog дальше и станет ли он стандартным языком разработки в некоторых узких, но многообещающих областях, все еще открыт. Конечно, у Prolog много возможностей, но для его грамотной разработки нужно забыть об особенностях мышления «традиционного» программирования, что непросто. Возможно, в ближайшее время появится новый язык логического программирования, который благодаря удачному PR станет стандартом в разработке «интеллектуальных» систем. Возможно, таким языком станет обновленный Prolog. Пока же массовый рынок не требует таких разработок, и языком Prolog больше интересуются в учебных целях.

Библиографический список

1. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог: Пер. с англ. — М.: Мир, 1990. — 235 с. : ил.
2. Братко И. Алгоритмы искусственного интеллекта на языке Prolog, 3-е издание. : Пер. с англ. — М. : Издательский дом "Вильямс", 2004. — 640 с. : ил. — Парал. Тит. англ.
3. https://ru.wikipedia.org/wiki/Пролог_(язык_программирования)#Архитектура – статья в Интернете
4. http://masters.donntu.org/2009/fvti/bandurka/library/article3.htm - статья в Интернете
5. https://ru.qaz.wiki/wiki/Prolog - статья в Интернете
6. https://habr.com/ru/post/124636/ - статья в Интернете
7. https://habr.com/ru/post/124820/ - статья в Интернете