МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)
Кафедра информационных систем и технологий

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г.Локтионова
«15» 02 2018 г.

МОДЕЛИ ПРЕДСТАВЛЕНИЯ И ОБРАБОТКИ ЗНАНИЙ В ИНФОРМАЦИОННО-АНАЛИТИЧЕСКИХ СИСТЕМАХ

Методические указания к лабораторным занятиям для магистров направления 02.04.03 Математическое обеспечение и администрирование информационных систем

Курск 2018

УДК 004.832.3
Составитель Ю.А. Халин

Рецензент
Кандидат технических наук, доцент С.Ю. Сазонов

Модели представления и обработки знаний в информационно-аналитических системах: методические указания к лабораторным занятиям / Юго-Зап. гос. ун-т; сост. Ю.А. Халин. Курск, 2018. 28 с. Библиогр.: с. 28.

Описываются основные модели представления и обработки знаний в информационно-аналитических системах. Изложены рекомендации по разработке программ на языке Prolog, используя составные объекты, арифметические и логические операции, управление поиском решения, обработку списков, обработку строк, файловые операции.

Методические рекомендации предназначены для студентов, обучающихся по направлению подготовки 02.04.03 Математическое обеспечение и администрирование информационных систем.

Текст печатается в авторской редакции.

Подписано в печать 15.02.18 . Формат 60x84 1/16.
Усл.печ. л. 1,59 п.л . Уч.-изд. л. 1,28 . Тираж 100 экз. Заказ. 1434
Бесплатно.
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

ВВЕДЕНИЕ

Лабораторные работы проводятся на IBM PC в среде любой операционной системы, в которой реализован язык PDC Prolog.

Для лучшего понимания предмета и сокращения потерь машинного времени при выполнении работ на PC необходима предварительная теоретическая проработка материала. Для этого в предварительно студентом прорабатывается весь теоретический материал, изложенный в лекционном курсе.

Каждому студенту по каждой из работ выдаются преподавателем индивидуальные задания. Методические указания к каждой лабораторной работе состоят из четырех разделов: цели работы, основных теоретических положений, порядка выполнения работы и содержания отчета.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Получить задание у преподавателя.
2. Написать программу (или программы), реализующую полученное задание.
3. Реализовать программу, получить результаты, дать пояснения.

СОДЕРЖАНИЕ ОТЧЕТА

1. Наименование и цель работы.
2. Основные этапы выполнения работы и их результаты.
3. Выводы.

Примечание. Лабораторный практикум может быть выполнен студентом самостоятельно при наличии персонального компьютера по месту работы или дома. В этом случае необходимо предварительное согласование индивидуального задания с преподавателем. Во всех случаях содержание отчета по каждой работе должно включать задание, листинг текста программы и результатов реализаций.

Лабораторная работа 1.

РЕАЛИЗАЦИЯ АРИФМЕТИЧЕСКИХ И ЛОГИЧЕСКИХ ОПЕРАЦИИ

I. ЦЕЛЬ РАБОТЫ

Изучение способов реализации арифметических вычислений и логических операций.

II. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Язык Prolog обладает всеми арифметическими возможностями, присущими другим языкам программирования (Pascal, C). Для этого используется мощный набор математических функций и стандартных предикатов.

Реализация арифметических операций

Выполнение четырех основных арифметических операций с указанием соответствия типов операндов и результатов выполнения приведены в табл. 2.1.

Таблица 2.1.

Операнд 1	Оператор	Операнд 2	Результат
integer	+,-,*	integer	Integer
integer	+,-,*	real	Real
real	+,-,*	integer	Real
real	+,-*	real	Real
integer		integer	
или	/	или	Real
real		real	

Необходимо отметить, что числа можно представлять в различных системах счисления: десятичной или шестнадцатиричной. Последние начинаются со знака доллара **($),** например: **$2EA = 2*16*16+14*16+10 =746.**

В простом примере 3.1 осуществляется сложение двух чисел, представленных в различных системах счисления.

```
/* Пример 2.1.*/
goal
   write("AA="),readint(AA),nl,
   A=-$2EA+AA, write("A=",A)
```

Математические функции языка приведены в табл. 2.2. Причем аргумент X этих функций является арифметическим выражением.

АРИФМЕТИЧЕСКИЕ ВЫРАЖЕНИЯ, такие, например, как **4, W** или **X = (cos(Y) - 1.5) / 3 + 2.5,** это константа, или переменная, или конструкция, построенная из них путем использования операторов, функций, или скобок и предиката равенства (=).

Таблица 2.2.

Предикат	Описание
abs(X)	Абсолютное значение числа X
exp(X)	Вычисляется число e в степени X (e)
ln(X)	Натуральный логарифм X
lg(X)	Десятичный логарифм (X)
sqrt(X)	Квадратный корень из X
sin(X)	Тригонометрические функции
cos(X)	(X - число, выражающее угол в радианах)
tan(X)	
arctan(X)	Арктангенс действительного числа X
random(X)	Устанавливает X, как псевдослучайное веществен-ное число с равномерным распределением: 0<=X<1
random(Y,X)	Случайное целое число X с равномерным распределением 0<=X<Y
round(X)	Округляет значение X
trunc(X)	Усекает X (отбрасывает младшие разряды)
X mod Y	Выдает остаток от деления X на Y
X div Y	Выдает частное от деления X на Y

Использование функции **random(RandomReal)** позволяет получать последовательность величин **RandomReal,** равномерно распределенных в интервале **0<= RandomReal <1.** А для установления правого граничного значения для случайных величин используется функция **randominit(Y).** Функция же **random(MaxValue,Randomint)** выдает последовательность псевдослучайных величин равномерно распределенных в интервал **0<=Randomint<MaxValue.**

При необходимости получить случайные вещественные числа **Z** в диапазоне от **A** до **B** включительно, нужно воспользоваться следующей целью:

random(X), Z = A + X * (B-A+1).

Для получения случайных целых чисел **Z** в диапазоне от **A** до **B** включительно можно воспользоваться целью:

random(Y,X), Z = A + Y +1.

Значение арифметического выражения может быть вычислено только тогда, когда все переменные, находящиеся в правой части, станут конкретизированными. Вычисления осуществляются с учетом приоритета арифметических операций в следующем порядке:

- вычисляются выражения в скобках,
- во всем выражении реализуются слева направо операции умножения и/или деления,
- выполняются слева направо операции сложения и/или вычитания.

В примере 2.2. реализуется вычисление значений арифметического выражения. При A=2 и B=0,3 X будет равен - 2,1815545036. Это пример, демонстрирующий программу на языке Prolog, содержащую лишь один раздел (раздел goal). Использование предикатов ввода-вывода будет пояснено ниже.

```
/* Пример 2.2.*/
goal
      write(«A=»),  readint(A),  nl,
      write(«B=»),   readreal(B),nl,
      X = -A + ((cos(B) - 1.5) / 3),
     write("X=",X).
```

Примеры 2.1 и 2.2 показывают, что язык Prolog может использоваться просто в режиме калькулятора. Для этого, не вводя никакой программы,

необходимо войти в окно диалога и набрать то арифметическое выражение, значение которого необходимо вычислить.

Пусть необходимо написать программу (пример 2.3), реализующую арифметические операции сложения, вычитания, умножения и деления. Величины, над которыми производятся операции имеют тип **real,** т.е. они могут быть представлены в любой форме записи целой или вещественной. Запись самой операции имеет тип **symbol.** Тогда предикат ОПЕРАЦИЯ (**operation**) имеет, например, следующий вид:

operation(symbol, real, real).

В разделе **clauses** описаны правила выполнения операций с выводом результатов вычислений на экран.

```
/*Пример 2.3.*/
predicates
    operation(symbol,real,real)
clauses
    operation(«+»,X,Y):-
        Z=X+Y, write(X, «+»,Y, «=»,Z),nl.
    operation(«-»,X,Y):-
        Z=X-Y, write(X, «-»,Y, «=»,Z),nl.
    operation(«*»,X,Y):-
        Z=X*Y, write(X, «*»,Y, «=»,Z),nl.
    operation(«/»,X,Y):-
        Z=X/Y, write(X, «/»,Y, «=»,Z),nl.
```

Цели внешние и имеют, например, следующий вид:

operation(«/»,5.5,3.4).

Результат: **5.5 / 3.4 = 1.6176470588.**

operation(«+»,3.4,0.006E2)

Результат: **3.4 + 0.6 = 4.**

Реализация логических операций

Язык Prolog позволяет выполнять бинарные операции отношений между арифметическими выражениями, символами, строками и

идентификаторами. Для этого используются следующие операторы сравнения:

=, <, <=, >, >=, <>(><).

Пример сравнения: **Y + 2 > 8 - X .**

Операция «=» устанавливает соответствие между левой и правой частями выражения. При согласовании переменных действуют следующие правила:

- если X неконкретизированная переменная, а Y -конкретизированная, то X и Y равны,

- целые числа и атомы всегда равны самим себе: например, **613 = 613 , book = book** являются верными утверждениями, а **245 = 45, abc = bcd** - неверными.

- две структуры равны, если они имеют один и тот же функтор и одинаковое число аргументов, причем все соответствующие аргументы равны.

Когда для сравнения вещественных величин используется предикат равенства, нужно позаботиться о том, чтобы приближенное представление вещественных чисел не привело к непредсказуемым результатам. Так цель **4.9999999999 = 5.0000000000** не будет удовлетворена. Это указывает на то, что при проверке на равенство двух вещественных чисел лучше определять, лежит ли их разность в заранее определенных пределах.

Кроме числовых выражений можно также сравнивать простые символы, строки и символические имена, например:

'a' < 'w', /*тип **char*/**

«MISHA» < «MASINA», /*string*/

X1=misha, X2=masha, X1>X2 /*symbol*/

При этом Prolog преобразует **'a' < 'w'** в выражение **97 < 119,** т.е. сравнивает значения кодов **ASCII,** и отношение истинно. Сравнение **«MISHA» < «MASINA »** ложно, т.к. первые отличающиеся символы в сравниваемых строках **I** и **A,** а их коды **73** и **65.** Соответственно сравнение **«bb» > «b»** истинно.

Символические строки нельзя сравнивать непосредственно. В примере **X1=misha, X2=masha** символическое имя **misha** не может быть сравнено непосредственно с **masha** . Они должны быть связаны с переменными или записаны как строки (тип **string).**

В примере 2.4 осуществляется проверка принадлежности целых чисел **Z** интервалу **(X,Y),** где **X** - минимальная граница интервала, а **Y** - максимальная.

```
/*Пример 2.4.*/
predicates
    include(integer, integer, integer)
clauses
    include(X,Y,Z):-
    Z>=X, Z<=Y, nl,
    write(«Число »,Z, « лежит в интервале от »,X,« до », Y,«.»).
```

Вопросы для самопроверки

1. Какое назначение предиката **random(RandomReal)?**
2. Как использовать язык Prolog в режиме калькулятора?
3. Перечислите основные правила установления соответствия между левой и правой частями выражения при выполнении операции «=».

Лабораторная работа №2. УПРАВЛЕНИЕ ПОИСКОМ РЕШЕНИЯ

I. ЦЕЛЬ РАБОТЫ

Изучение способов реализации рекурсивных зависимостей.

II. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

В языке Prolog используются две конструкции правил, реализующих многократное выполнение задачи. Это ***ПОВТОРЕНИЕ*** и ***РЕКУРСИЯ.***

Структура правила, реализующего повторение, такая:

repetitiverule:-
<предикаты и правила>,
fail. /* неудача*/.

Структура правила, реализующего рекурсию:

recursiverule:-
<предикаты и правила>,
recursiverule.

В правиле, реализующем повторение, используется встроенный предикат **fail** (неудача). Это всегда ложный предикат, вызывающий откат для поиска других утверждений, которые бы обеспечили вычисление цели. В теле правила рекурсии последним правилом является само правило **recursive_rule.**

Используемая конструкция <предикаты и правила> в теле правил содержит предикаты и правила программы.

МЕТОД ОТКАТА ПОСЛЕ НЕУДАЧИ Для управления вычислением внутренней цели ***ПРИ ПОИСКЕ ВСЕХ ВОЗМОЖНЫХ РЕШЕНИЙ*** используется метод отката после неудачи. Правило

repetitiverule:-
<предикаты и правила>,
fail.

будет выполняться столько раз, сколько существует альтернатив для введенной цели.

Для демонстрации использования предиката **fail** и метода ОПН можно привести пример 3.1 о городах. В результате выполнения этой

программы выводится небольшой перечень городов северо-запада России, включенных в базу данных.

```
/*Пример 3.1.*/
domains
        name=symbol
predicates
        town(name)
        show_towns
goal
        write("Города северо-запада России:"),nl,
        show_towns.
clauses
        town ("Петербург"). town ("Новгород"). Town ("Псков").
        town ("Петрозаводск"). town ("Выборг"). town
        ("Ивангород"). town ("Ладога"). town ("Вологда").
show_ towns:-
        town (Town), write("  ", Town),nl,

fail.
```

На рис. 3.1 изображен процесс отката при реализации примера 3.1. База данных примера содержит 8 альтернативных утверждений для предиката **town(name).** Prolog ищет в базе данных сопоставимые утверждения, при этом внутренние унификационные программы (ВУПы) сначала означивают переменную **Town** объектом первого утверждения **Петербург.** Но существуют и другие утверждения в базе, которые могли бы быть использованы при вычислении подцели **town(Town).** Поэтому для запоминания места возможного отката ВУПы устанавливают указатель отката на следующее утверждение базы данных - **town("Новгород").** Значение **Петербург** переменной **Town** выводится на экран монитора, предикат **fail** вызывает неуспешное завершение правила, переменная **Town** освобождается, а ВУПы осуществляют откат в точку 1 и так далее.

Этот пример показывает применение метода ОПН для извлечения данных из каждого утверждения базы данных при использовании внутренней цели.

МЕТОД ОТСЕЧЕНИЯ И ОТКАТА

Для того, чтобы при выполнении программы была возможность ОГРАНИЧИТЬ ПОИСК В БАЗЕ ДАННЫХ, используют средства управления откатом. Это средство называется МЕТОДОМ ОТСЕЧЕНИЯ И ОТКАТА (ОО) и позволяет, используя встроенный предикат **cut,** который обозначается символом восклицательного знака **(!),** осуществлять «фильтрацию данных» по некоторым
условиям, т.е. УПРАВЛЯТЬ МЕХАНИЗМОМ АВТОМАТИЧЕСКИХ ОТКАТОВ.

Этот предикат, вычисление которого всегда успешно, заставляет ВУПы заблокировать все указатели откатов, установленные при вычислении предыдущих подцелей. Например, имеется некоторое правило:

pr:-aa,bb,!,cc

и предикату **aa** соответствуют факты и правила **a1, a2, a3;** предикату **bb - bl, b2; cc - cl, c2, c3, c4.** Если бы отсутствовал предикат **cut,** то было бы предпринято 3*2*4=24 попытки поиска решений. Но пусть осуществляется последовательный поиск решения с откатом до тех пор, пока факт **a1** не удовлетворит предикату **aa,** а **bl** - предикату **bb,** далее происходит успешное завершение предиката **cut.** Затем при переборе фактов, соответствующих предикату **cc,** делается попытка отката на предыдущий предикат **bb,** но **cut** (отсечение) не позволяет это сделать и процесс поиска альтернативных решений в правиле заканчивается.

В связи с отсутствием в языке Prolog конструкций, аналогичных операторам циклов процедурных языков, отличительной его особенностью от большинства языков программирования является широкое использование РЕКУРСИИ.

Рекурсией называется правило, содержащее само себя в качестве компоненты. Обычно Prolog-программа представляет собой совокупность рекурсивных или взаиморекурсивных определений.

ПРОСТАЯ РЕКУРСИЯ

Прежде всего необходимо показать подводные камни, возникающие при использовании рекурсии. Следующее правило

pr:- pr

имеет такой смысл: "**pr** истинно, если **pr** истинно". Непосредственное использование этого правила может привести к бесконечному циклу (бесконечной рекурсии), а это приведет к переполнению стека, что

нежелательно, т.к. могут быть потеряны промежуточные результаты вычислений. Ниже приводится пример программы с использованием бесконечной рекурсии:

```
predicates
        write_string
goal
        write_string.
clauses
        write_string:-
                write("Изучайте PDC Prolog!"),nl,
                write_string.
```

При запуске этой программы на экран выдается строка "Изучайте PDC Prolog!", а т.к. последняя компонента правила **write_string** является самим правилом, то опять осуществляется выдача на экран той же строки и т.д. Такая рекурсия называется БЕСКОНЕЧНОЙ и практического применения не имеет.

Рекурсия может стать конечной, если в правило включается УСЛОВИЕ ВЫХОДА. Пусть, например, генерируется последовательность целых случайных величин, равномерно распределенных в интервале от 0 до 9, до тех пор пока не будет получено число 5. В этом случае процесс генерации и выполнение программы завершается (пример 3.2).

```
/*Пример 3.2.*/
    predicates
        write_number
    goal
        write_number.
    clauses
        writenumber:-
                random(9,Randomint),
                Randomint < > 5, write(«
                »,Randomint),
                write_number.
```

Первой компонентой правила **write_number** является встроенный предикат **random,** генерирующий последовательность случайных величин. Значение случайной величины присваивается переменной **Randomint.** Следующее подправило проверяет, является ли означенное значение переменной **Randomint** числом 5. Если нет, то подправило

успешно, число выводится на экран и производится рекурсивный вызов **write_number.** Процесс продолжается до тех пор, пока не будет сгенерировано число 5. После этого выполнение программы завершается.

ОБОБЩЕННОЕ ПРАВИЛО РЕКУРСИИ Схематически обобщенное правило рекурсии (ОПР) записывается следующим образом:

<имя 1фавила рекурсии>:-
<список предикатов>,
<предикат, определяющий условие выхода>,
<список предикатов>,
<имя правила рекурсии>,
<список предикатов>.

Принцип работы этого правила идентичен работе правила простой рекурсии. Три конструкции **<список предикатов>** на рекурсию не оказывают влияния. Успешное выполнение предиката, определяющего условие выхода, вызывает продолжение рекурсии, а неуспешное - ее прекращение. Конструкция в теле правила **<имя правила рекурсии>** - это само рекурсивное правило и его успех вызывает (продолжает) рекурсию.

Для получения навыков использования ОПР-метода необходимо рассмотреть еще два примера: генерации (без использования предиката **random)** ряда чисел (пример 3.3) и вычисления факториала (3.4).

Пусть требуется сгенерировать последовательность целых чисел от 1 до 5 (пример 3.3).

```
/*Пример 3.3.*/
predicates
        number(integer)
goal
        write("Числовой ряд:"),
        number(1),
        write(" . Конец!").
clauses
        number(6).
        number(Number):-
                Number<6,
                write(" ", Number),
```

New_number=Number+1,
number(Newnumber).

В рассматриваемом примере первая и последняя конструкции **<список предикатов>** общего правила рекурсии не используются. Именем правила рекурсии является: **number(Number).** В этом правиле предикатом, определяющим, условие выхода, является **Number<6** и, когда **Number=6,** правило успешно, а программа завершается.

Средняя конструкция **<список предикатов>** выводит на экран значение переменной **Number** и затем увеличивает ее на единицу **New_number=Number+1.** Причем для нового числа используется новая переменная **New_number.** И, наконец, программа завершается именем правила рекурсии в теле правила **number(New_number).**

При запуске программы осуществляется попытка вычислить подцель **number(1)** и для этого она безуспешно сопоставляется с утверждением **number(6).** Затем эта подцель сопоставляется с головой правила **number(Number)** и сопоставление успешно, а переменная **Number** принимат значение **1.** Сравнение **1<6** успешно, поэтому условие выхода не выполняется, и значение переменной **Number=1** выводится на экран, а переменная **New_number** становится равной **2.** Затем правило **number(New_number)** со значением параметра **2,** присвоенным переменной **New_number,** вызывает само себя. Несовпадение имен переменных **Number** и **New_number** не имеет значения, т.к. они указывают при передаче значений только на позицию в списке параметров.

Далее продолжается выполнение программы пока **New_number** не станет равным **6,** а это значит, что условие выхода выполняется успешно и программа завершается. Результат на экране такой:

Числовой ряд: 1,2,3,4,5. Конец!

Следующий пример посвящен вычислению факториала

$$n! = n * (n-1) * (n-2) *...* 3 * 2 * 1.$$

Причем $n! = n * (n-1)!$, а $1! = 1$. Например $6! = 6 * 5 * 4 * 3 * 2 * 1 = 720$.

```
/* Пример 3.4.*/
domains
      n=integer
```

```
        f=real
predicates
        factor(n,f)
goal
        write("n="),readint(N),nl,write(N,"!="),
        factor(N,F), write(F),nl.
clauses
        factor(0,1):-!.
        factor(N,F):-
        NN=N-1,factor(NN,PF),F=N*PF.
```

Вопросы для самопроверки

1. Объясните назначение предикатов **fail** и **cut.**
2. Как реализуется метод отсечения и отката?
3. Объясните структуру обобщенного правила рекурсии.

Лабораторная работа 3.

ОБРАБОТКА СПИСКОВ

I. ЦЕЛЬ РАБОТЫ

Изучение приемов задания списков и способов реализации операций над списками.

II. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

СПИСОК - это динамическая структура, элементами которой могут быть атомы, списки или структуры. Список является еще одним типом данных наравне с основными типами **char, integer, real, string, symbol** и **file.** **Э**то упорядоченная последовательность равноправных объектов, которая может иметь произвольную длину. Список - это набор объектов одного и того же типа.

Элементом списка может быть атом, переменная, составной терм или список. Основное отличие списков от массивов, используемых в процедурных языках заключается в том, что в списке никогда не задается количество элементов (массив же обычно имеет фиксированную длину), в него не могут входить элементы разных типов данных.

Элементы списка заключаются в квадратные скобки и отделяются друг от друга запятыми. Ниже приведены примеры списков с элементами типа **real, integer** и **symbol:**

[1.4,2.4,3.5,3.6,4.7],
[0,1,2,3,4,5,6],
["Информатика", "Программирование", "Интеллект"].

Важно, что все элементы списка принадлежат к одному типу доменов. Поэтому список

[a,'b',"c"]

некорректен, т.к. состоит из элементов разных типов. Количество элементов списка называется его длиной. Длина списка

["Миша", "Маша", "Вася"]

равна 3. Пустым или нулевым списком называется список, не содержащий элементов. В этом случае он называется атомом и обозначается так: **[].**

Использование списка требует его описания по крайней мере в трех разделах программы. В разделе **domains** описывается используемый тип данных. Например:

domains
town_list=town*/* объявляется список городов*/
town=symbol /* объявляются элементы town списка townlist; их тип symbol*/

или объявление типа данных может быть таким:
town_list=symbol*.

Если список состоит, например, из списков целых чисел **[[1,2,3],[2,3,4],[3,6,7],[]] ,** то его объявление ничем не отличается от предыдущего: **listList = integer* .** В разделе **predicates** указывается имя предиката и в скобках -имя списка **towns(town_list)** или **numbers(listList).**

В разделе **clauses,** как всегда, приводятся утверждения, т.е. конкретные значения предикатов. Пример 4.1 иллюстрирует описание списка городов.

```
/* Пример 4.1.*/
domains
    town_list=town*
    town=symbol
predicates
    towns(town_list)
clauses
    towns([«Петербург»,«Псков»,«Новгород»,«Ладога»,«Ямбург»]).
```

Реализация такой программы возможна при использовании, например, следующих внешних целей:

towns(All). или **towns([X,_,_,Y,_]).**

Список не имеет никаких предикатов для своей обработки. Для снятия этого неудобства введена единственная операция над элементами списка, называемая **МЕТОДОМ РАЗДЕЛЕНИЯ СПИСКА НА ГОЛОВУ И ХВОСТ.** При этом непустой список рассматривается как структура, состоящая из двух частей:

[Head|Tail].

Переменная **Head** (голова списка) это его первый элемент или фиксированное количество элементов, отделенных символом "|"

(вертикальная черта). А переменная **Tail** (хвост) это список оставшихся элементов списка.

В таблице 4.1 приведены примеры деления списка на голову и хвост. При этом видно, что хвост списка - всегда список, а голова - элемент списка, и разделение списка на голову и хвост не зависит от длины списка. Последовательное применение этого метода позволяет легко осуществлять различные операции над списками и написать практически любую программу по обработке любого мыслимого списка.

Таблица 4.1

Список	Голова	Хвост
[1,2,3,4]	1	[2,3,4]
[abcd]	abcd	[]
["Изучайте","PDC","Prolog"]	Изучайте	["PDC","Prolog"]
[A*B,C-D]	A*B	[C-D]
[[E,F,G],h,[i,J]]	[E,F,G]	[h,[i,J]]
[]	Не определено	Не определено

Лабораторная работа 4.
ПРЕДСТАВЛЕНИЕ ЗНАНИЙ НА ОСНОВЕ АЛГЕБРЫ ВЫСКАЗЫВАНИЙ

Краткие теоретические сведения.

Высказывание - повествовательное предложение *(утверждение, суждение),* о котором имеет смысл говорить, что оно *истинно (значение 1)* или *ложно (значение 0).*

Табличное задание *логических операций.*

Исходные высказывания		*Вид операции*							
A	*B*	$\neg A$	$A \wedge B$	$A \vee B$	$A \rightarrow B$	$A \leftrightarrow B$	$A \mid B$	$A \uparrow B$	$A \oplus B$
0	*0*	*1*	*0*	*0*	*1*	*1*	*1*	*1*	*0*
0	*1*	*1*	*0*	*1*	*1*	*0*	*1*	*0*	*1*
1	*0*	*0*	*0*	*1*	*0*	*0*	*1*	*0*	*1*
1	*1*	*0*	*1*	*1*	*1*	*1*	*0*	*0*	*0*

Формулой алгебры высказываний называется выражение, составленное из переменных высказываний с помощью операций над высказываниями и обращающееся в конкретное высказывание при подстановке вместо переменных высказываний конкретных высказываний.

Подформулой формулы **A** называется любое подслово (часть) слова **A**, которое само являющееся формулой.

Формула алгебры высказываний называется *тавтологией* или *тождественно истинной,* если при любых значениях входящих в нее переменных высказываний она превращается всегда в истинное высказывание.

Формула алгебры высказываний называется *тождественно ложной* или *противоречием,* если она ложна, как бы ни определялись значения переменных высказываний, входящих в эту формулу

Формула алгебры высказываний называется *выполнимой* (разрешимой*),* если при определенном наборе значений

переменных высказываний она превращаться в истинное высказывание.

Формулы A и B называются эквивалентными (равносильными). (обозначается A≡B), если при любых значениях переменных высказываний значение A совпадает со значением B.

Если **V** является подформулой формулы **F** и **V** эквивалентна **W**, то формула $\mathbf{F_1}$, полученная из **F** заменой подформулы **V** на **W**, эквивалентна формуле **F.**

Если две формулы эквивалентны A≡B, то формула $A \leftrightarrow B$ тождественно истинна.

Примеры выполнения заданий

Задача 1. Определите, является ли данное выражение формулой. Если это формула, то выпишите последовательность построения формулы.

Выражение $(A \vee B)(C \to A)$ формулой не является, т.к. выражения $(A \vee B)$ и $(C \to A)$ формулами являются в соответствии с определением, но между ними нет ни какой операции.

Задача 2. Сколькими способами можно расставить скобки в последовательности, чтобы получилась формула. Выписать все возможные получаемые формулы.

В выражении $A \to B \vee C$ можно расставить скобки для получения формулы двумя способами: $((A \to B) \vee C)$ и $(A \to (B \vee C))$. В первом случае первой выполняется операция импликации, а вторая – дизъюнкция. Во второй формуле первой выполняется дизъюнкция, а второй – импликация.

Задача 3. Выписать все подформулы данной формулы.

В формуле $((A \to B) \vee C)$ пять подформул: A, B, C, $(A \to B)$ и сама формула, которая является несобственной подформулой. Первые 4 являются собственными подформулами.

Задача 4. Указать тип формулы. Доказать вывод.

Для формулы $((A \to B) \vee C)$ составим таблицу истинности, с помощью которой и определим тип формулы.

A	*B*	*C*	$(A \to B)$	$((A \to B) \vee C)$
0	*0*	*0*	*1*	*1*
0	*0*	*1*	*1*	*1*
0	*1*	*0*	*1*	*1*
0	*1*	*1*	*1*	*1*
1	*0*	*0*	*0*	*0*
1	*0*	*1*	*0*	*1*
1	*1*	*0*	*1*	*1*
1	*1*	*1*	*1*	*1*

Так как у формулы есть значение 0, то она не является тождественно истинной. А так как есть значения 1, то она является выполнимой.

Задача 5. С помощью таблиц истинности, а так же с помощью эквивалентных преобразований проверить на эквивалентность формулы:

$((A \to B) \vee C)$ и $((A \vee B) \vee C)$.

Для первой формулы таблица истинности составлена в предыдущей задаче, поэтому составим таблицу истинности только для второй формулы.

A	*B*	*C*	$(A \vee B)$	$((A \vee B) \vee C)$
0	*0*	*0*	*0*	*0*
0	*0*	*1*	*0*	*1*
0	*1*	*0*	*1*	*1*
0	*1*	*1*	*1*	*1*
1	*0*	*0*	*1*	*1*
1	*0*	*1*	*1*	*1*
1	*1*	*0*	*1*	*1*
1	*1*	*1*	*1*	*1*

Формулы принимают значение 0 на разных наборах значений переменных, поэтому они не эквивалентны.

К первой формуле применим эквивалентное преобразование,

заменив импликацию через отрицание и дизъюнкцию. В силу ассоциативности дизъюнкции в формулах можно опустить некоторые скобки.

$$((A \to B) \vee C) \equiv ((\neg A \vee B) \vee C) \equiv \neg A \vee B \vee C \not\equiv A \vee B \vee C \equiv ((A \vee B) \vee C)$$

Задача 6. Представьте логическими формулами пословицы и поговорки.

В поговорке «Без труда не вынешь и рыбки из пруда» выделим простые высказывания, обозначив из буквами: А- Человек трудится, В – поймать рыбку в пруду. Тогда пословица примет вид $((\neg A) \to (\neg B))$.

Задача 7. Доказать законы логики. Закон двойного отрицания.

A	$\neg A$	$\neg\neg A$	$A \leftrightarrow \neg\neg A$
0	*1*	*0*	*1*
1	*0*	*1*	*1*

Задача 8. При каких значениях переменных формула ложна.

Для формулы $(X \vee Y) \to X$ можно составить таблицу истинности и выбрать те наборы простых высказываний, на которых формула имеет значение 0. Но можно эти наборы получить рассуждениями. Импликация ложна только в случае истинности посылки и ложности заключения. Следовательно, $X = 0$, но при этом посылка будет истинной только при Y= 1. В данном случае это единственный набор, на котором формула ложна.

Практические задания по вариантам

Задача 1. Определите, является ли данное выражение формулой. Если это формула, то выпишите последовательность построения формулы.

Вариант	Выражение
1	$(A_0 \& A_1) A_2 \neg A_3$
2	$(A_0 \& A_1) \Rightarrow A_5$
3	$((A_3 \Rightarrow A_0) \& \neg A_0)$
4	$(((\neg A_0) \Rightarrow A_1) \Rightarrow \neg(A_2 \vee A_3))$

5	$((X) \Rightarrow (У))\&(Z)$
6	$(X\&(У))\vee(Z)$
7	$(\neg A \Rightarrow B\&C)\vee(D\&\neg A \Rightarrow C)$
8	$((A_0 \Rightarrow A_1) \Rightarrow((A_0 \neg A_1) \Rightarrow \neg A_1)$
9	$A_0 \Rightarrow(\neg A_1 \vee (A_1\&)A_2$
10	$A_1 \Rightarrow A_2 \Rightarrow A_3 \Rightarrow \neg A_1 \Rightarrow \neg A_2$

Задача 2. Сколькими способами можно расставить скобки в последовательности, чтобы получилась формула. Выписать все возможные получаемые формулы.

Вариант	Выражение
1	$A_1 \Rightarrow A_2 \Rightarrow A_3 \Rightarrow \neg A_1 \Rightarrow \neg A_2$
2	$A_0 \Rightarrow \neg A_1 \vee A_1\&A_2$
3	$\neg A_0 \Rightarrow A_1 \Rightarrow \neg A_2 \vee A_3$
4	$X \Rightarrow У\&Z$
5	$A_0 \Rightarrow A_1 \Rightarrow A_0 \Rightarrow \neg A_1$
6	$A_0\&A_1 \Rightarrow A_2\&\neg A_3$
7	$A_0\&A_1 \Rightarrow A_5$
8	$A_3 \Rightarrow A_0\&\neg A_0$
9	$\neg A \Rightarrow B\&C \vee D\&\neg A \Rightarrow C$
10	$X\&У \vee Z$

Задача 3. Выписать все подформулы данной формулы.

Вариант	Выражение
1	$(((A_0 \Rightarrow A_1)\&(A_1 \Rightarrow A_2)) \Rightarrow(\neg A_0 \vee A_2))$
2	$(\neg A_0 \Rightarrow A_1) \Rightarrow (\neg A_2 \vee A_3)$
3	$((A_0 \Rightarrow \neg A_1) \vee A_1)\&A_2$
4	$((A_0 \Rightarrow A_1) \Rightarrow((A_0 \Rightarrow \neg A_1) \Rightarrow \neg A_1)$
5	$(\neg A \Rightarrow B \wedge C) \vee ((D \wedge \neg A) \Rightarrow \neg C)$
6	$(\neg(A \Rightarrow B)\&C) \vee (((D\&(\neg A)) \Rightarrow C)$
7	$((A \wedge B) \Rightarrow C) \wedge (D \vee (A \Leftrightarrow C))$
8	$(\neg A \Rightarrow B \wedge C) \vee (D \wedge \neg A \Rightarrow \neg C)$
9	$(A_0 \Rightarrow (A_1 \Rightarrow A_0)) \Rightarrow (\neg A_1)$
10	$(((\neg A_0) \Rightarrow A_1) \Rightarrow \neg(A_2 \vee A_3))$

Задача 4. Указать тип формулы. Доказать вывод.

Вариант	Выражение
1	$(A\wedge B\Rightarrow C)\wedge(D\vee A\Leftrightarrow C)$
2	$(\neg(A\Rightarrow B)\&C)\vee(((D\&(\neg A))\Rightarrow C)$
3	$(A\Rightarrow B)\wedge(B\Rightarrow A)$
4	$(A\vee B)\wedge(B\Rightarrow A)$
5	$(A\wedge B\Rightarrow C)\wedge(D\wedge A\Rightarrow C)$
6	$(\neg A\Rightarrow B)\wedge(B\Rightarrow A)$
7	$((A\wedge B)\Rightarrow C)\wedge(D\vee(A\Leftrightarrow C))$
8	$(\neg A\Rightarrow\neg B)\wedge(B\Rightarrow A)$
9	$(\neg A\Rightarrow B\wedge C)\vee((D\wedge\neg A)\Rightarrow\neg C)$
10	$(\neg A\Rightarrow B\wedge C)\vee(D\wedge\neg A\Rightarrow C)$

Задача 5. С помощью таблиц истинности, а так же с помощью эквивалентных преобразований проверить на эквивалентность формулы.

Вариант	Формулы	
1	$(A\Rightarrow A\wedge C)\vee(B\wedge\neg A)$	$(A\Rightarrow(A\wedge C\Leftrightarrow B))$
2	$(A\Rightarrow A\wedge C)\vee(B\wedge\neg A)$	$(A\Rightarrow(C\Rightarrow B))$
3	$(A\Rightarrow A\wedge C)\vee(B\wedge\neg A)$	$(A\wedge C)\vee(\neg B\vee\neg A\wedge B)$
4	$(A\Rightarrow A\wedge C)\vee(B\wedge\neg A)$	$\neg A\vee B\vee\neg C$
5	$(A\wedge B\Rightarrow\neg B\vee A\wedge C)$	$(\neg B\Leftrightarrow A)\Rightarrow C)$
6	$(A\wedge B\Rightarrow\neg B\vee A\wedge C)$	$(A \mid B)\vee C$
7	$(A\wedge B\Rightarrow\neg B\vee A\wedge C)$	$(A\uparrow C) \mid B$
8	$(A\wedge B\Rightarrow\neg B\vee A\wedge C)$	$B\Rightarrow A\wedge C\vee\neg A\wedge B$
9	$B\Rightarrow A\wedge C\vee\neg A\wedge B$	$(A\Rightarrow(A\wedge C\Leftrightarrow B))$
10	$B\Rightarrow A\wedge C\vee\neg A\wedge B$	$(A\wedge C)\vee(\neg B\vee\neg A\wedge B)$

Задача 6. Представьте логическими формулами пословицы и поговорки.

Вариант	Выражение
1	Без еды не будет и беседы.
2	Без недостатка только Бог, без грязи только вода.

3	Близкому не говори ложь, постороннему не говори правду.
4	Если тебе угощать нечем – хоть говори ласково.
5	Когда грома много – дождя мало.
6	Гнев впереди, ум позади.
7	Доброе слово — половина счастья.
8	Если уважаешь отца, люби и сына; если уважаешь хозяина, корми и его собаку.
9	Кочерга длинная – не обожжешь руки; много родных – люди не обидят.
10	Кто много видит – становится умнее, кто много говорит – становится красноречивее.

Задача 7. Доказать законы логики.

Вариант	Название законов.
1	Закон двойного отрицания. Идемпотентность дизъюнкции.
2	Коммутативный закон конъюнкции. Закон тождества.
3	Коммутативный закон дизъюнкции. Идемпотентность конъюнкции.
4	Ассоциативность конъюнкции. Закон поглощения для дизъюнкции.
5	Ассоциативность дизъюнкции. Закон поглощения для конъюнкции.
6	Дистрибутивность конъюнкции относительно дизъюнкции. Закон противоречия.
7	Дистрибутивность дизъюнкции относительно конъюнкции. Свойства единицы.
8	Закон де Моргана - отрицание конъюнкции. Выражение импликации через дизъюнкцию.
9	Закон де Моргана - отрицание дизъюнкции. Выражение импликации через конъюнкцию.
10	Закон исключения третьего. Свойства нуля.

Задача 8. При каких значениях переменных формула ложна.

Вариант	Формула
1	$(((X \Rightarrow (Y \wedge Z)) \Rightarrow (\neg Y \Rightarrow \neg X)) \Rightarrow \neg Y)$
2	$(((X \vee Y) \vee Z) \Rightarrow ((X \vee Y) \wedge (X \vee Z)))$
3	$(((X \vee Y) \wedge ((Y \vee Z) \wedge (Z \vee X))) \Rightarrow ((X \wedge Y) \wedge Z))$
4	$((X \vee Y) \Rightarrow ((\neg X \wedge Y) \vee (X \wedge \neg Y)))$
5	$(((X \Rightarrow Y) \Rightarrow (Y \Rightarrow X))$
6	$((Q \Rightarrow (P \wedge R)) \wedge \neg((P \vee R) \Rightarrow Q))$
7	$\neg(X \Rightarrow \neg X)$
8	$(((X \vee Y) \wedge Z) \Rightarrow ((X \vee Y) \wedge (X \vee Z)))$
9	$(((X \Rightarrow (Y \wedge Z)) \Rightarrow (\neg Y \vee \neg X)) \Rightarrow \neg Y)$
10	$(((X \vee Y) \Rightarrow (Y \Rightarrow X))$