ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Государственное образовательное учреждение
высшего профессионального образования

“МАТИ” — РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. К. Э. ЦИОЛКОВСКОГО

Кафедра “Моделирование систем и информационные технологии”

ПРОГРАММИРОВАНИЕ ТЕОРЕТИКО-МНОЖЕСТВЕННЫХ ОПЕРАЦИЙ

Методические указания к лабораторной работе по курсу
“Системы искусственного интеллекта”

Составитель В. В. Лидовский

Москва 2008

ВВЕДЕНИЕ

Настоящие методические указания предназначены для обеспечения учебного процесса студентов четвертого курса дневной формы обучения специальности 220200 “АСОИиУ” при выполнении лабораторной работы по предмету “Системы искусственного интеллекта”.

Цели лабораторной работы: изучить возможности средств языков программирования Лисп и Пролог для реализации математических операций с множествами.

1. СИСТЕМНЫЕ ТРЕБОВАНИЯ

Для выполнения лабораторной работы необходимы следующие системные компоненты:

а) компьютер, работающий под управлением операционной системы Linux;
б) транслятор Лиспа, совместимый с ANSI Common Lisp, например, GNU CLisp;
в) транслятор Пролога, соответствующий стандарту ISO, например, GNU Prolog.

2. ТЕОРЕТИКО-МНОЖЕСТВЕННЫЕ ОПЕРАЦИИ

Множество — это неупорядоченная совокупность различных элементов. Понятие множества часто используется и для него существует много синонимов: класс, группа, толпа, стая, совокупность, ... Принадлежность элемента a множеству S записывается $a \in S$.

Подмножеством S множества U называется любое множество S, все элементы которого принадлежат U. Это отношение между S и U записывается как $S \subset U$. Подмножество S множества U часто определяется как множество всех тех элементов $x \in U$, которые обладают некоторым определенным свойством. Свойство — это утверждение относительно переменной x, функция-предикат $p(x)$. Итак, символически определение S можно записать как $S = \{x \in U \mid p(x)\}$ или $S = \{x \mid p(x)\}$. Последние формулы читаются так: “S есть множество всех элементов x множества U, для которых справедливо утверждение $p(x)$”. Например, $S = \{x \in U \mid x/2 \in \mathbb{Z}\}$ определяет множество S четных элементов U, $\mathbb{Z}$ — целые числа.

Множество называется пустым, если оно не содержит ни одного элемента. Пустое множество обозначается знаком $\varnothing$.

Множество всех подмножеств множества U обозначается через $P(U)$ и называется множеством-степенью. Множество-степень всегда

содержит в качестве своих элементов само U и пустое множество. Множество всех частей $P(U)$ множества $U = \{a, b, c\}$ — это $\{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, U\}$.

Если U состоит из n элементов $a_1, a_2, \ldots, a_n$, то $P(U)$ состоит из 2^n подмножеств $S_1, S_2, \ldots, S_{2^n}$. Этот подсчет основан на том, что для каждого элемента $a_i \in U$ $(1 \leqslant i \leqslant n)$ и каждого подмножества $S_k \subset U$ имеет место ровно одна из двух возможностей: $a_i \in S_k$ или $a_i \notin S_k$. Иными словами, любое множество S_k задается цепочкой длины n из нулей и единиц (двоичным представлением числа $k - 1$), где 0 в i-й позиции означает, что $a_i \notin S_k$, а 1 — $a_i \in S_k$. Получается, что между элементами $P(U)$ и числами от 0 до $2^n - 1$ существует взаимно-однозначное соответствие. Аналогичным образом можно установить соответствие между $P(P(U))$ и числами от 0 до $2^{2^n} - 1$, причем соответствие для $P(U)$ сохраняется, и т. д.

Характеристическая функция множества S, $\chi_S(x)$, определяется соотношением

$$\chi_S(x) = \begin{cases} 1, & \text{если } x \in S; \\ 0, & \text{если } x \notin S. \end{cases}$$

Например, если $S = \{1, 2, 3\}$, то $\chi_S(2) = 1$ и $\chi_S(4) = \chi_S(0) = 0$. Очевидно, что $\chi_S(a) = 1$ тогда и только тогда, когда $\{a\} \subset S$ или $a \in S$.

Декартовым или прямым произведением множеств A и B называется совокупность всех упорядоченных пар $\langle x, y\rangle$ таких, что $x \in A$, а $y \in B$. Обозначение — $A \times B$. Например, если $A = \{1, 2, 3\}$ и $B = \{2, 4\}$, то $A \times B = \{\langle 1, 2\rangle, \langle 1, 4\rangle, \langle 2, 2\rangle, \langle 2, 4\rangle, \langle 3, 4\rangle, \langle 3, 2\rangle\}$, а $B \times A = \{\langle 2, 1\rangle, \langle 4, 1\rangle, \langle 2, 2\rangle, \langle 4, 2\rangle, \langle 4, 3\rangle, \langle 2, 3\rangle\}$. Компоненты декартова произведения — это не множества, а упорядоченные наборы, что требует для них специального представления на уровне бит.

Объединением двух множеств A и B, $A \cup B$, называют множество, состоящее только из тех элементов, которые входят либо в A, либо в B, т. е. $A \cup B = \{x \mid x \in A \text{ или } x \in B\}$.

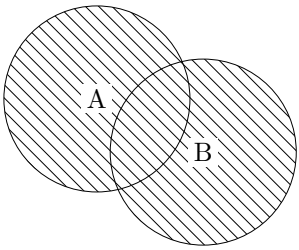
Пересечением двух множеств A и B, $A \cap B$, называют множество, состоящее только из тех элементов, которые входят как в A, так и в B, т. е. $A \cap B = \{x \mid x \in A \text{ и } x \in B\}$.

Относительным дополнением множества A до множества B или разностью множеств B и A называется множество тех элементов B, которые не принадлежат A. Символически это записывается $B \setminus A = B - A = \{x \in B \mid x \notin A\}$.

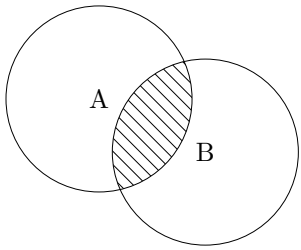
Симметрической разностью множеств A и B, $A + B$, называется множество тех элементов из $A \cup B$, которые не входят в $A \cap B$, т. е. $A + B = (A \cup B) \setminus (A \cap B)$.

Если $A = \{1, 2, 3, 4, 5\}$ и $B = \{2, 4, 6\}$, то $A \cup B = \{1, 2, 3, 4, 5, 6\}$,

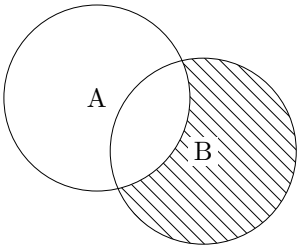
$A \cap B = \{2,4\}$, $A \setminus B = \{1,3,5\}$, $B \cup A = \{6\}$, $A + B = \{1,3,5,6\}$ (см. рисунок).



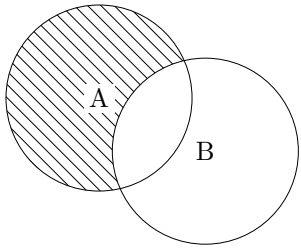
$A \cup B$



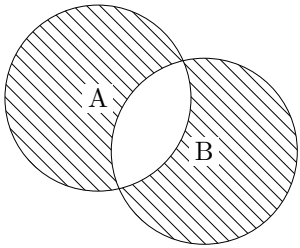
$A \cap B$



$B \setminus A$



$A \setminus B$



$A + B$

Если U множество из n элементов, то, как это отмечалось ранее, каждому $S \subset U$ можно взаимно-однозначно сопоставить двоичное представление числа из диапазона от 0 до $2^n - 1$. Операции $\cup$, $\cap$, $\setminus$ и $+$ над множествами A и B эквивалентны соответственно поразрядным логическим операциям OR, AND, NOT IMP, XOR над их двоичными представлениями. Если $f(X)$ — это функция-биекция по множеству X,

$X \subset U$, возвращающая двоичное представление X, то верно, что

$$f(A \cup B) = f(A) \vee f(B),$$

$$f(A \cap B) = f(A) \,\&\, f(B),$$

$$f(A \setminus B) = \neg(f(A) \Rightarrow f(B)) = f(A) \,\&\, \neg f(B),$$

$$f(A + B) = f(A) + f(B),$$

где & — это операция конъюнкции AND (И), $\vee$ — дизъюнкции OR (ИЛИ), $\Rightarrow$ — импликации IMP (ЕСЛИ-ТО), $\neg$ — отрицания NOT (НЕ), $+$ (в правой части) — сложения по модулю 2, XOR (исключающее ИЛИ).

Главной проблемой реализации работы с множествами на компьютере является произвольная природа компоненты множества — эта компонента сама может быть произвольным множеством. При реализации множеств при помощи вырожденных ассоциативных массивов, как это принято в стандартной библиотеке Си++, все элементы множества должны быть однотипными. Хотя, как это было показано выше, несложно свести операции с множествами к логическим операциям с бинарными данными, это сведение требует предварительной декларации всех элементов множеств, что не всегда возможно. Символьное представление множества, например, средствами языка Лисп или Пролог, не требует такой декларации и является наиболее общим способом представления произвольных конечных множеств. Например, множество

$$\{1, 2, \{2, 3\}, 4, \{1, \{4, 5\}\}\}$$

естественно представляется списком Лиспа вида (1 2 (2 3) 4 (1 (4 5))) или списком Пролога вида [1, 2, [2, 3], 4, [1, [4, 5]]]. Реализация операций над множествами в символьной форме значительно сложнее и медленнее, чем в числовой. В частности, несложная теоретически операция сравнения двух множеств на равенство становится относительно непростой.

3. БАЗОВЫЕ СРЕДСТВА ЛИСПА

Базовый тип данных Лиспа — это атом. Списки — это последовательности из атомов и других списков в круглых скобках. Пустой список, (), имеет название NIL и является также атомом. Кроме того, NIL является логическим значением ложь, а любое отличное от NIL значение считается истинным. Принято использовать атом T как стандартное значение истинного выражения.

Программы на Лиспе обычно пишутся в функциональном стиле, который делает ненужными операторы присваивания, циклов и перехода. Вся лисп-программа — это последовательность описаний функций. Строчные и прописные буквы не различаются. При описании функций можно пользоваться и операторным стилем с циклами и присваиваниями. Лисп-система предоставляет широкий набор уже готовых функций, называемых примитивными.

Вызов функции F с аргументами X и Y на Лиспе записывается (F X Y), т. е. любой список рассматривается как вызов функции с именем, первым элементом списка. Если нужно рассматривать список как данные, то перед ним нужно ставить апостроф. Аналогичным образом, атом рассматривается как значение переменной с его именем. Если же нужно использовать имя атома, то перед ним ставится апостроф. Числа и NIL всегда означают самих себя и перед ними ставить апостроф необязательно.

Основные примитивные функции:

• CAR — первый элемент списка, например, (CAR '((2 3) 4)) будет равно (2 3);

• CDR — список без первого элемента, например, (CDR '((2 3) 4)) будет равно (4), (CDR (CAR '((2 3) 4))) — (3);

• CONS — вставка элемента в начало списка, например, (CONS '(2 3) '(4)) будет равно ((2 3) 4);

• ATOM — функция-предикат, истинна, если ее аргумент — атом, например, (ATOM '(2 3)) будет равно NIL, а (ATOM 4) — T;

• EQL — функция-предикат, истинна, если ее аргументы-атомы равны, например, (EQL 2 (CAR '(2 3))) будет равно T, (EQL NIL ()) — T, а (EQL '(2) '(2)) и (EQL 3 '4) — NIL;

• NULL — функция-предикат, истинна, если ее аргумент — NIL, например, (NULL '(2 3)) будет равно NIL, а (NULL ()) — T;

• LIST — заключение списка аргументов в скобки — образование списка из аргументов, например, (LIST '(2 3) '(4)) будет равно ((2 3) (4)), а (LIST 1 2 NIL) — (1 2 NIL);

• APPEND — соединяет списки-аргументы, например, (APPEND '(2 (3)) '(4) NIL) будет равно (2 (3) 4), а (APPEND '(1 2) '(2 3)) — (1 2 2 3);

• LENGTH — длина списка, например, (LENGTH '((2 3) 4)) будет равно 2, а (LENGTH NIL) — 0;

• EQUAL — функция-предикат, истинна, если ее аргументы, атомы или списки, равны, например, (EQUAL 2 2) будет равно T, (EQUAL '(A B) '(A B)) — T, а (EQUAL '(2 3) '(3 2)) — NIL;

• NTH — обобщение CAR с двумя аргументами, n-й с нуля элемент

списка, например, (NTH 1 '((2 3) 4)) будет равно 4, а (NTH 0 '((2 3) 4)) — (CAR '((2 3) 4));
• MAPCAR — применение заданной первым аргументом функции к каждому элементу второго аргумента-списка, например, вызов (MAPCAR 'NULL '(A () (A) NIL)) приводит к результату (NIL T NIL T).

Математические примитивные функции:
• $+ - */$ — четыре основные функции арифметики, например, (+ 1 (∗ 2 3) 4) и (− 25 (/ 24 8) 5 6) будут равны 11;
• TRUNCATE — целочисленное деление с отбрасыванием остатка, например, (TRUNCATE 17 3) будет равно 5;
• EXPT — возведение в степень, например, (EXPT 2 8) будет равно 256, а (EXPT 8 2) — 64;
• ODDP — предикат, проверка на нечетность, (ODDP 5) равно T, а (ODDP 8) — NIL;
• NOT — отрицание, действует в точности как и NULL, например, (NOT NIL) будет равно T.

Средства общие для функционального и операторного стилей:
• COND — основное средство разветвления вычислений, с помощью которого можно строить условные предложения вида

(COND (p_1 e_{11} e_{12} ...)
(p_2 e_{21} e_{22} ...)
...
(p_N e_{N1} e_{N2} ...)),

где p_i — это предикаты, а e_{ij} — выражения. Если все p_1, p_2, ... ложны, то значение COND тоже ложь. Если p_m первый истинный предикат, то значением COND будет значение последнего выражения в списке с p_m или сам p_m, если выражений после него нет;
• DEFUN — определение новой функции, например,

```
(DEFUN SECOND (LST)
    (CAR (CDR LST)) )
```

определяет новую функцию SECOND с одним аргументом, результат которой — второй элемент списка-аргумента. Если в список формальных аргументов поставить &OPTIONAL, то все аргументы после него считаются необязательными — их можно не писать при вызове функции. Необязательные параметры можно использовать как локальные переменные, инициализированные значением NIL.

Средства операторного стиля:
• SETQ — присваивание, например, после (SETQ N 2) атом N будет иметь значение 2 и (+ N N) будет равно 4;

• PUSH — помещение в список, например, если значение L — это список (B C D), то после (PUSH 'A L) со значением (A B C D) значение L тоже станет (A B C D);
• POP — извлечение из списка, например, если значение L — это список (A B C D), то после (POP L) со значением A значением L станет (B C D);
• LOOP — цикл, выход из которого возможен только при помощи RETURN;
• RETURN — выход из цикла с заданным значением, например,

```
(SETQ F 1)
(LOOP
    (COND ((EQL N 0) (RETURN F)))
    (SETQ F (* F N))
    (SETQ N (- N 1)))
```

организует вычисление факториала числа N — этот факториал является значением вызова LOOP, а также значением атома F после вычисления LOOP.

4. РЕАЛИЗАЦИЯ ВСПОМОГАТЕЛЬНЫХ ФУНКЦИЙ

Как уже отмечалось, сравнение на равенство множеств в списочном представлении — это относительно непростая операция. Главная проблема, обусловленная неупорядоченностью элементов множества, состоит в том, что одному множеству из n элементов можно сопоставить не менее $n!$ различных списков.

Для реализации операции сравнения на равенство множеств в списочном представлении можно воспользоваться идеей "вычеркивания" — ищем первый элемент первого множества во втором множестве и, если находим, то вычеркиваем оба и продолжаем сравнение, а если не находим, то сравниваемые множества не равны. При продолжении сравнения оба множества будут уменьшатся. Если они оба станут пустыми, то это будет означать их равенство. Если одно множество станет пустым, а другое нет, то это будет означать их неравенство.

На Лиспе идею "вычеркивания" можно реализовать следующей функцией-предикатом

```
(DEFUN SETEQUAL (SET1 SET2)
    (COND
        ((EQUAL SET1 SET2))
        ((ATOM SET1) NIL)
        ((ATOM SET2) NIL)
        ((INSET (CAR SET1) SET2)
            (SETEQUAL (CDR SET1) (REMELM (CAR SET1) SET2)) ))),
```

где INSET — это функция, реализующая операцию $\in$ для списочного представления множеств, а REMELM — это функция, реализующая "вычеркивание" элемента из второго множества. "Вычеркивание" элемента из первого множества не требует специальной операции, так как "вычеркиваемый" элемент здесь всегда первый и можно воспользоваться стандартной функцией CDR. Функция SETEQUAL может корректно сравнивать и атомы.

Функция-предикат INSET ищет первый аргумент во втором аргументе-списке. В случае успеха поиска результат — истина, в случае неуспеха — ложь. При реализации INSET нужно опять игнорировать порядок в списках-множествах — это достигается использованием SETEQUAL. Функция INSET последовательно сравнивает при помощи функции SETEQUAL свой первый аргумент с элементами второго аргумента-списка.

```
(DEFUN INSET (ELM TSET)
    (COND
        ((NULL TSET) NIL)
        ((SETEQUAL ELM (CAR TSET)))
            ((INSET ELM (CDR TSET))) ))
```

Функция REMELM удаляет из второго аргумента-списка элемент, равный первому аргументу. Если удаляемый элемент — первый, то он удаляется при помощи CDR, а если не первый, то из списка при помощи CDR выделяется его "хвост", в котором удаляемый элемент возможно будет первым, и т. д.

```
(DEFUN REMELM (ELM TSET)
    (COND
        ((SETEQUAL ELM (CAR TSET)) (CDR TSET))
        ((CONS (CAR TSET) (REMELM ELM (CDR TSET)))) ))
```

Для реализации прямого произведения множеств будет полезна вспомогательная функция AUXDES, которая по заданному элементу и списку-множеству генерирует все пары из заданного элемента и элементов списка. Например, вызов (AUXDES 'A '(A B)) должен сгенерировать список ((A B) (A A)).

```
(DEFUN AUXDES (ELM SET1 &optional SET2)
    (LOOP
        (COND ((NULL SET1) (RETURN SET2)))
        (PUSH (LIST ELM (POP SET1)) SET2) ))
```

При реализации этой функции можно обойтись без явной итерации или рекурсии.

```
(DEFUN AUXDES (ELM SET)
   (MAPCAR (LAMBDA (X) (LIST ELM X)) SET))
```

Один из способов построить множество-степень заданного множества состоит в использовании вспомогательной функции CONS2ALL, которая добавляет первый аргумент в первую позицию каждого члена-списка второго аргумента. Например, (CONS2ALL 'A '((B) (C D) NIL))) произведет список ((A B) (A C D) (A)).

```
(DEFUN CONS2ALL (ELM SET)
    (COND
      ((NULL SET) NIL)
      ((CONS (CONS ELM (CAR SET)) (CONS2ALL ELM (CDR SET))))))
```

5. ТЕОРЕТИКО-МНОЖЕСТВЕННЫЕ ОПЕРАЦИИ НА ПРОЛОГЕ

Средства языка программирования Пролог позволяют, как и средства Лиспа, производить сколь угодно сложные операции над списками. Неитерационные функции Лиспа естественно преобразуются в их эквиваленты на Прологе. Например, рассмотренные выше функции SETEQUAL, INSET, REMELM и CONS2ALL на Прологе имеют следующий вид.

```
setequal(X,Y) :-
    X=Y, !;
    atom(X), !, fail;
    atom(Y), !, fail;
    X=[CarX|CdrX], remelm(CarX,Y,Z), setequal(CdrX,Z).


inset(X,Y) :-
    Y=[], !, fail;
    Y=[CarY|CdrY], (setequal(CarY,X), !; inset(X,CdrY)).


remelm(Elm,S,R):-
    S=[CarS|CdrS], (
        setequal(CarS,Elm), R=CdrS, !;
        remelm(Elm,CdrS,T), R=[CarS|T]).
```

```
cons2all(X,Y,R):-
    Y=[], R=[], !;
    [CarY|CdrY]=Y, cons2all(X,CdrY,Z), [[X|CarY]|Z]=R.
```

Легко заметить соответствие между функциями Лиспа и Пролога. В Прологе пустой список, [], также является атомом, а голова списка отделяется от хвоста при помощи знака |. Таким образом, аналогом базовой функции Лиспа CAR будет предикат

```
car(X,CarX):-X=[CarX|_],
```

а аналогом CDR — предикат

```
cdr(X,CdrX):-X=[_|CdrX].
```

Функции Пролога — предикаты, поэтому нелогические величины они могут возвращать только при помощи дополнительных параметров, таких как CarX или CdrX. Таким образом, при вызове car([[1,2],3],X) значением X станет [1,2], а при вызове cdr([[1,2],3],X) — [3]. Часто удобнее получать голову и хвост списка за одну операцию (см. примеры выше). Для сравнения как атомов, так и списков в Прологе можно использовать предикат =, который также можно использовать и для присваивания значений переменным, например, [X|[2,3]]=[1|Y] приведет к присваиванию списка [2,3] переменной Y и атома 1 переменной X (X и Y должны не иметь значения до этой операции).

Функция atom имеет одинаковый смысл и в Лиспе, и в Прологе. Вместо функции NULL в Прологе можно использовать ее эквивалент в форме X=[], где X — это проверяемый на пустоту список. Функция append в Прологе всегда имеет три аргумента — она соединяет два первых аргумента-списка в один список, передаваемый в третий аргумент. Аналогичным образом, для возвращения результата nth в Прологе использует свой третий аргумент. Кроме того, в отличие от Лиспа нумерация элементов списка начинается в Прологе с 1. Функция length также отличается от одноименной в Лиспе только наличием дополнительного аргумента для результата. Возведение в степень в Прологе записывается при помощи $**$, например, $2**3 = 8.0$, т. е. результат этой операции — всегда дробный. Для округления числа до ближайшего целого можно использовать функцию round, $\mathrm{round}(4.2) = 4$. Операция mod возвращает остаток от деления, $7 \bmod 4 = 3$.

Конструкция Пролога p:-p1,(p2;p3) используется для сокращения записи и эквивалента конструкции p:-p1,p2;p1,p3.

В Прологе практически отсутствуют средства для описания итерационных вычислений. Всегда истинный предикат repeat (его побочный эффект состоит в создании узла с бесконечным числом альтернатив в дереве вычисления) можно использовать вместо функции Лиспа LOOP. Отсечение, !, ведет себя подобно функции RETURN, но оно выходит сразу из всех циклов, т. е. использовать вложенные циклы напрямую в Прологе практически невозможно. Вложенные циклы можно реализовать только при помощи вспомогательных процедур-предикатов. Примером такой вспомогательной процедуры может быть insert_subset, которая добавляет подмножество, созданное на основе значений глобальных/динамических переменных l, t1 и t2, к множеству в глобальной/динамической переменной r. Переменная l должна иметь при старте работы этой процедуры значение пустого множества, []. При выходе из процедуры ее значение — это созданное на основе значений t1 и t2 подмножество, которое добавляется к r. Переменная t2 — это двоичный образ создаваемого подмножества, число от 0 до $2^n - 1$, где n — количество элементов в множестве, подмножество которого строим. Переменная t1 — это счетчик от 1 до n.

```
insert_subset(S):-
    repeat, (
        var(t2,0), retract(var(r,R)), var(l,L), assertz(var(r,[L|R])), !;
        var(t2,T2), 1 is T2 mod 2, var(t1,T1), nth(T1,S,S2),
          retract(var(l,L)), assertz(var(l,[S2|L])), fail;
        retract(var(t2,T2)), NT2 is T2//2, assertz(var(t2,NT2)),
          retract(var(t1,T1)), NT1 is T1+1, assertz(var(t1,NT1)), fail).
```

Главная проблема при реализации итерационных вычислений на чистом Прологе — это отсутствие возможности менять значения переменных, например, изменить значение переменной X с 1 на 2. Для подобных операций нужно либо использовать глобальные переменные, работа с которыми нестандартизирована и поддерживается не всеми реализациями языка, либо использовать аппарат работы с динамическими данными на основе средств assert и retract.

Любые итерационные вычисления можно преобразовать в вычисления, использующие рекурсию. В частности, аналог функции AUXDES может иметь следующий вид.

```
auxdes(Elm,S,R) :-
    S=[], R=[], !;
    S=[CarS|CdrS], auxdes(Elm,CdrS,R1), R=[[Elm,CarS]|R1].
```

6. ПОРЯДОК ВЫПОЛНЕНИЯ И ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

6.1. Порядок выполнения лабораторной работы

Лабораторная работа выполняется за следующие три этапа:

1) реализация функции, соответствующей варианту задания;
2) проведение тестирования полученной программы;
3) анализ хода выполнения лисп/пролог-программы при вычислении значений функции из варианта задания.

6.2. Отчет по лабораторной работе

Отчет по работе выполняется на отдельных листах или в отдельной тетради и должен содержать:

1) цель исследования;
2) спецификацию задания;
3) примеры тестовых прогонов программы с результатами и описанием порядка получения этих результатов лисп/пролог-системой;
4) выводы.

7. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Каково назначение языка программирования Лисп, его сильные и слабые стороны?
2. Каково назначение языка программирования Пролог, его сильные и слабые стороны?
3. Как представить конечное множество при помощи булевых данных?
4. Как связаны теоретико-множественные и булевы операции?
5. Каковы особенности представления множеств в виде списков?
6. Каковы недостатки и преимущества использования Лиспа или Пролога при реализации математических операций с множествами?
7. Как можно усовершенствовать приведенные в этом методическом указании функции?
8. Можно ли предложенными функциями работать с бесконечными множествами?
9. Оптимальным ли был предложенный способ вычислений?
10. Можно ли в предложенной реализации функции SETEQUAL вместо функции ATOM использовать функцию NULL?
11. Каков алгоритм работы предложенной реализации вспомогательной функции REMELM?

12. Опишите алгоритм работы функции POWER-SET, включенной в приложение, или предложите альтернативный алгоритм.
13. Опишите алгоритм работы предложенной реализации функции для вычисления разности множеств или предложите альтернативный алгоритм.
14. Использовался ли в предложенной реализации операции объединения множеств механизм косвенной рекурсии?
15. Использовался ли в предложенной реализации операции получения прямого произведения множеств операторный стиль?
16. Преобразуйте рекурсивную (итерационную) функцию в итерационную (рекурсивную).
17. Сравните преимущества приведенных функций для расчета множества-степени.

8. ВАРИАНТЫ РАБОТ

Лабораторная работа по теме “Программирование теоретико-множественных операций” имеет 8 вариантов заданий, определяемых следующей таблицей.

Вариант	*Операция*
1	$\cup$
2	$\cap$
3	$\setminus$
4	$+$
5	$\subset$
6	χ
7	множество-степень
8	$\times$

ЛИТЕРАТУРА

1. Лорьер Жан-Луис *Системы искусственного интеллекта* — М.: Мир, 1991. – 568 с.
2. Нефедов В. Н., Осипова В. А. *Курс дискретной математики* — М.: МАИ, 1992. – 264 с.
3. Стерлинг Л., Шапиро Э. *Искусство программирования на языке Пролог* — М.: Мир, 1990. – 235 с.
4. Хювнен Э., Сеппянен И. *Мир Лиспа* — М.: Мир, 1990. – 766 с.

ПРИЛОЖЕНИЕ 1

Пример лисп-программы для реализации теоретико-множественных операций, использующей приведенные вспомогательные функции.

```
(DEFUN UNION1 (SET1 SET2)
  (COND
    ((NULL SET1) SET2)
    ((INSET (CAR SET1) SET2)
      (UNION1 (CDR SET1) SET2) )
    ((UNION1 (CDR SET1) (CONS (CAR SET1) SET2))) ))

(DEFUN INTERSECTION2 (SET1 SET2)
  (COND
    ((NULL SET1) NIL)
    ((INSET (CAR SET1) SET2)
      (CONS (CAR SET1) (INTERSECTION2 (CDR SET1) SET2)) )
    ((INTERSECTION2 (CDR SET1) SET2)) ))

(DEFUN SET-DIFFERENCE3 (SET1 SET2)
  (COND
    ((NULL SET2) SET1)
    ((INSET (CAR SET2) SET1)
      (SET-DIFFERENCE3 (REMELM (CAR SET2) SET1) (CDR SET2)) )
    ((SET-DIFFERENCE3 SET1 (CDR SET2))) ))

(DEFUN SET-EXCLUSIVE-OR4 (SET1 SET2)
  (COND
    ((NULL SET1) SET2)
    ((INSET (CAR SET1) SET2)
      (SET-EXCLUSIVE-OR4 (CDR SET1) (REMELM (CAR SET1) SET2)) )
    ((SET-EXCLUSIVE-OR4 (CDR SET1) (CONS (CAR SET1) SET2))) ))

(DEFUN SUBSETP5 (SET1 SET2)
  (COND
    ((NULL SET1))
    ((INSET (CAR SET1) SET2)
      (SUBSETP5 (CDR SET1) SET2) )))

(DEFUN CHI6 (ELM TSET)
  (COND
    ((INSET ELM TSET) 1)
    (0) ))

(DEFUN POWER-SET7 (SET1 &optional SET2 SET3 P T1 T2)
  (SETQ P (- (EXPT 2 (LENGTH SET1)) 1))
  (LOOP
    (COND ((EQL P 0) (RETURN (CONS NIL SET2))))
    (SETQ T1 0)
    (SETQ T2 P)
    (SETQ SET3 NIL)
    (LOOP
      (COND
        ((EQL T2 0) (RETURN (PUSH SET3 SET2)))
```

```
        ((ODDP T2) (PUSH (NTH T1 SET1) SET3)))
      (SETQ T2 (TRUNCATE T2 2))
      (SETQ T1 (+ T1 1)) )
    (SETQ P (- P 1)) ))
(DEFUN POWER-SET7A (SET)
  (COND
    ((NULL SET) '(NIL))
    (((LAMBDA (X) (APPEND X (CONS2ALL (CAR SET) X)))
       (POWER-SET7A (CDR SET)))) ))
(DEFUN DESCARTES8 (SET1 SET2 &optional SET3)
  (COND
    ((NULL SET1) NIL)
    ((LOOP
      (COND ((NULL SET1) (RETURN SET3)))
      (SETQ SET3 (APPEND (AUXDES (POP SET1) SET2) SET3)) ))))
```

ПРИЛОЖЕНИЕ 2

Пример пролог-программы для реализации теоретико-множественных операций, использующей приведенные вспомогательные функции.

```
union1(S1,S2,R) :-
  S1=[], R=S2, !;
  S1=[CarS1|CdrS1], (
    inset(CarS1,S2), union1(CdrS1,S2,R), !;
    union1(CdrS1,[CarS1|S2],R)).

intersection2(S1,S2,R) :-
  S1=[], R=[], !;
  S1=[CarS1|CdrS1], (
    inset(CarS1,S2), intersection2(CdrS1,S2,T), R=[CarS1|T], !;
    intersection2(CdrS1,S2,R)).

set_difference3(S1,S2,R) :-
  S2=[], R=S1, !;
  S2=[CarS2|CdrS2], (
    remelm(CarS2,S1,T), set_difference3(T,CdrS2,R), !;
    set_difference3(S1,CdrS2,R)).

set_exclusive_or4(S1,S2,R) :-
  S1=[], R=S2, !;
  S1=[CarS1|CdrS1], (
    remelm(CarS1,S2,T), set_exclusive_or4(CdrS1,T,R), !;
    set_exclusive_or4(CdrS1,[CarS1|S2],R)).

subset5(S1,S2):-
  S1=[], !;
  S1=[CarS1|CdrS1], inset(CarS1,S2), subset5(CdrS1,S2).

chi6(Elm,S,R) :-
  inset(Elm,S), R=1, !;
  R=0.

power_set7(S,PS):-
  length(S,N), P is round(2**N)-1, assertz(var(p,P)), assertz(var(r,[])), fail;
  repeat, (
    retract(var(p,0)), retract(var(r,R)), PS=[[]|R], !;
    assertz(var(t1,1)), retract(var(p,P)), assertz(var(t2,P)),
      assertz(var(l,[])), insert_subset(S), retract(var(t1,_)),
      retract(var(t2,_)), retract(var(l,_)), N is P-1, assertz(var(p,N)),
      fail).

power_set7a(S,PS):-
  S=[], PS=[[]], !;
  S=[CarS|CdrS], power_set7a(CdrS,T), cons2all(CarS,T,U), append(T,U,PS).

descartes8(S1,S2,R) :-
  S1=[], R=[], !;
  S1=[CarS1|CdrS1], auxdes(CarS1,S2,R1),
     descartes8(CdrS1,S2,R2), append(R1,R2,R).
```

ОГЛАВЛЕНИЕ

Владимир Викторович Лидовский

ПРОГРАММИРОВАНИЕ ТЕОРЕТИКО-МНОЖЕСТВЕННЫХ ОПЕРАЦИЙ

Методические указания к лабораторной работе по курсу
“Системы искусственного интеллекта”

Редактор М. А. Соколова

Оригинал-макет подготовлен в пакетах Plain-TEX и MetaPost при использовании средств ConTEXt

Под. в печ. 12.03.2008 Объем 1,25 п.л. Тираж 70 экз. Зак. 12

Издательский центр МАТИ, Берниковская наб. 14