computing@tanet.edu.te.ua
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

# РЕАЛІЗАЦІЯ ТРАНСЛЯТОРА МОВИ ПРОГРАМУВАННЯ PROLOG: ЛЕКСИЧНИЙ ТА СИНТАКСИЧНИЙ АНАЛІЗ

**Олександр Цимбал**

Харківський національний університет радіоелектроніки,
м. Харків, проспект Леніна 14, mcdulcimer@ukr.net

**Резюме:** *У статті розглядається розробка транслятора мови програмування Prolog у складі лексичного та синтаксичного аналізаторів, блоків керування таблицями та інтерпретації. Надано схеми синтаксичного аналізу та елементи програмного коду транслятора. Програмне забезпечення реалізовано у середовищі розробки Visual C++ 2005 (Beta).*

**Ключові слова:** *Мова програмування, транслятор, Prolog, інтерпретатор, граматика, лексичний аналіз, синтаксичний аналіз.*

## 1. ВСТУП

Кожна мова програмування має на меті реалізацію певної моделі подання даних і методів їх обробки. Під час розробки програми відбувається перетворення конструкцій початкових даних задачі у вигляд, відповідний моделі, лексичним та синтаксичним особливостям мови програмування. Вибір мови програмування обумовлюється здатністю у найефективніший спосіб здійснити поставлені перед програмістом завдання.

Порівняно з універсальними мовами, мова програмування Prolog (Programming in Logic) виглядає зовсім незвично, реалізація методів обробки даних у ній забезпечується побудовою програми у формі предикатів першого роду. Такі особливості Prolog, проте, не заважають ефективно виконувати складні обчислювальні процедури. Головною особливістю є те, що Prolog з практичної точки зору реалізує логічну модель подання даних та знань [1].

Розвиток методів штучного інтелекту та необхідність удосконалення мовних засобів їх практичного втілення ставлять завдання модернізації мови програмування Prolog, тому розробка нового транслятора мови Prolog, адаптованого на розв'язання задач робототехніки, є актуальною та своєчасною.

## 2. АНАЛІЗ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

Хоча свого часу саме мова програмування Prolog визнавалася провідною у проекті «ЕОМ V покоління» [1], в наш час, нею займаються лише спеціалісти в галузі логічного програмування та штучного інтелекту. Натомість Prolog вважається однією з класичних мов програмування для систем штучного інтелекту [2, 3]. Prolog дозволяє розглядати експертні системи не тільки з теоретичної, але і з практичної точки зору [4].

Використання Prolog виглядає особливо доречним з точки зору програмної реалізації моделей подання знань, зокрема продукційної та фреймової [2]. Використання рекурсивних процедур обробки списків також надає широкі можливості організації пошукових функцій інформаційних систем, розв'язання завдань планування, в тому числі у робототехніці [3].

Цікавий спосіб використання Prolog описано у [6], де вирішується зворотне завдання – за допомогою Prolog описуються принципи реалізації інших мов програмування. При цьому визнається максимальна простота та зрозумілість реалізації основних мовних конструкцій, спроектованих саме за допомогою Prolog.

Розгляд можливостей побудови власної реалізації мови Prolog вимагає використання класичних методів розробки, описаних у [4]. Проте, з практичної точки зору, доцільно використовувати програмні конструкції, описані у [7, 8]. Після незначної переробки ці матеріали можна легко використати для створення варіанту власної версії транслятора.

Таким чином, якщо задача вимагатиме розробки транслятора мови Prolog, її розв'язання полягатиме у застосуванні відомих методів

розробки мов програмування.

## 3. ПОСТАНОВКА ЗАДАЧІ

Якщо розглядати проблематику моделювання робототехнічних систем, можна вказати на існування, на першому рівні, проблем реалізації численних розрахунків у кінематичних моделях, які доцільно реалізувати за допомогою універсальних мов програмування, наприклад мови Сі. Крім того, мова Сі (або мова Асемблер) стає у нагоді під час безпосереднього формування керуючих сигналів системі керування маніпулятора робота.

На іншому рівні – рівні прийняття рішень, розрахунки стають допоміжним фактором і на передній план виступає необхідність методів логічного програмування. Таким чином загальне завдання моделювання робототехнічних систем можна розглядати сукупністю різнопланових задач, вирішення яких цілком ймовірно потребуватиме залучення різних програмних засобів. З точки зору цілісності проекту такий підхід є небажаним, бо викликатиме проблему взаємодії різних засобів програмування. Інший підхід, має передбачати реалізацію основи проекту базовою (універсальною) мовою програмування та реалізацію додаткових програмних засобів на рівні трансляторів.

Під час розробки системи моделювання та керування інтелектуальним роботом пропонується реалізувати підсистему підтримки та прийняття рішень мовою Prolog. Для забезпечення її функціонування у складі системи виникає завдання розробки інтерпретатора мови програмування Prolog.

Метою статті є опис основних особливостей розробки транслятора мови програмування Prolog. Ставляться завдання побудови граматик мови Prolog, реалізацій лексичного та синтаксичного аналізаторів. Засобом створення транслятора є мова програмування Сі (середовище розробника Visual C++).

## 4. ОПИС РЕАЛІЗАЦІЇ МОВИ ПРОГРАМУВАННЯ PROLOG

Реалізуємо інтерпретатор мови програмування Prolog, що у своїх синтаксичних конструкціях в основному відповідає Turbo-Prolog, розробленому компанією Borland Inc. За основу побудови транслятора використаємо принципи, застосовані у [7] під час побудови простої мови програмування SPL.

Алфавіт мови Prolog складається з символів латинської мови, десяткових цифр, множини службових символів: {'(', ')', ',', ';', '=', '+', '-', '*', '/', '%' ,'[',']', '.', '|', '>', '<'} та пробілу. З символів алфавіту складаються лексеми – мінімальні мовні ланцюжки із самостійним змістом.

Ідентифікатором у Prolog може бути послідовність літер та цифр, яка починається з літери. Беззнаковим числом є послідовність цифр. З точки зору синтаксису Prolog усі ідентифікатори є однаковими і утворюють лексему IDEN.

Числа без знаку утворюють лексему NUMB.

Окрім IDEN та NUMB у Prolog визначаються спеціальні дванадцять ідентифікаторів – ключових слів мови. До них належать {"domains", "database", "predicates", "clauses", "goal", "integer", "real", "string", "char", "symbol", ":-", "write"}.

Символи пробілу, "\n" та "\t" припустимі у кожному місці програми за винятком ідентифікаторів та беззнакових чисел. Інші службові символи утворюють окремі лексеми мови.

Prolog у даній реалізації має скалярні дані цілого (integer), дійсного (real), символьного (char та symbol) типів та типу рядок (string). Крім того, послідовність значень визначеного скалярного типу може складати список елементів. Елементи списку відокремлюються комами та розміщуються всередині квадратних дужок, наприклад: [12, 31, -5, 6] або ['a', 'd', 'E']. Довжина списку не визначається. У першому варіанті транслятора реалізовано лише тип integer.

Ідентифікатори змінних Prolog-програми записуються з великої, ідентифікатори імен предикатів та констант – з малої літери. Усі змінні є локальними в межах предикату, де вони визначені або до якого передані.

Вирази Prolog будуються з констант, змінних та імен предикатів (з круглими дужками для позначення списку параметрів або без них). Знаками операцій є "+", "-", "*", "/", "%". Вирази мають вигляд: v, c, ($e_1$), predicate($e_1$, $e_2$,…,$e_n$ ), $e_1$ + $e_2$, $e_1$ – $e_2$, $e_1$ * $e_2$, $e_1$ / $e_2$, $e_1$ % $e_2$, + $e_1$, –$e_1$, де v – змінна, c – константа, predicate – ідентифікатор імені предиката, $e_1$, $e_2$,…,$e_n$ – вирази.

У Prolog визначаються вбудовані оператори: присвоєння/співвідношення - v=c; та друку - write(v), де v – змінна, e – вираз, c – константа.

Програма мовою Prolog має логічний та / або обчислювальний зміст. Далі в основному розглядається її обчислювальний зміст.

Програма на Prolog має чітко визначену структуру і складається з таких розділів:

| | |
|---|---|
| domains | визначення імен типів даних, складених типів та списків; |
| database | визначення вигляду фактів, що складатимуть вміст бази даних Prolog-програми; |
| predicates | визначення прототипів предикатів; |
| clauses | реалізація предикатів програми; |
| goal | виклик предикатів на виконання або запуск програми. |

Розділ "goal" Prolog-програми відіграє ту є саму роль, що функція main() програми, написаної мовою Сі. У "goal" відбувається запуск першого з необхідних предикатів, проводиться необхідна ініціалізація параметрів. Додамо, що усі змінні, ініціалізовані у goal є локальними в його межах. Також локальними є змінні, розміщені у записах окремих предикатів Prolog-програми, що цілком відповідає стандарту мови.

## 5. РОЗРОБКА ЛЕКСИЧНОГО ТА СИНТАКСИЧНОГО АНАЛІЗАТОРІВ

Текст програми, що є реалізацією інтерпретатора мови Prolog розпочинається із директив підключення заголовочних файлів:

```
#include<iostream>      // за VC++ 8.0 (beta)
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<ctype.h>
#include<string.h>
#include "pt04a.h"      // опис прототипів
```

Останній з них містить прототипи усіх використаних у тексті транслятора функцій (їх вигляд легко зрозуміти з анведених далі у тексті функцій) та визначення структур, що забезпечують режими роботи інтерпретатора:

```
typedef struct  {int cod,opd;}cmd;
typedef struct  {char *name,*owner; int what,val;}odc;
typedef struct  {char *name; int isd,cpt,start;}fnd;
```

Також слід визначити глобальні змінні програми:

```
cmd TCD[300], cmd *pcmd=TCD;
char TNM[400], *ptn=TNM, char *lstr;
static char nch='N';
odc TOB[100];
odc *pto=TOB; odc *ptol=TOB;
fnd TFN[30], fnd *ptf=TFN;
int adrnm,cpnm,out,lval,lex;
int cgv=0,clv=0,tc=0,nst=0;
int st[500],sp,t,p;
using namespace std;
FILE *PF; errno_t err;
```

Під час проведення лексичного аналізу необхідно визначити лексеми, з яких складається програма, зокрема службові слова, якими є "domains", "database", "predicates", "clauses", "goal", "integer", "real", "string", "char", "symbol", ":-", "write". Вказані службові лексеми кодуються відповідними константами DOMAINSL, DATABASEL, PREDICATESL, CLAUSESL, GOALL, INTL, REALL, STRINGL, CHARL, SYMBOLL, IFL, CONSTL, WRITEL. Також до лексем належать знаки операцій та роздільники, беззнакові цілі дані з кодом NUMB та значенням lval, ідентифікатори із кодом IDEN (входять до таблиці ідентифікаторів TNM), ідентифікатор кінця файла із кодом EOF.

Інформацію про коди лексем можна зберегти за допомогою перелічення:

```
enum {DOMAINSL=257, DATABASEL, PREDICATESL,
CLAUSESL, GOALL, INTL, REALL, STRINGL, CHARL,
SYMBOLL, IFL, CONSTL, NUMB, IDEN, WRITEL};
```

Запуск програми транслятора здійснюється у звичайному стилі консольної програми:

```
void main(int argc,char *argv[])
{printf("\n Prolog translator 01 \n");
 if(argc>1 && argv[1])
 {analys(argv[1]);
  cout << "\n Analys is finished \n \n";
   if(!error_syntax) {interp();
          cout << "\n Interpreting is finished \n \n"; }
   else cout << "\n Interpreting is impossible \n \n";
 }
 else printf("\n no program file \n");
}
```

Для виконання Prolog-програми необхідно викликати її з командного рядка операційної системи вказавши у якості параметра ім'я файла з текстом програми, наприклад:

D:\ My Projects\PT04a.exe 1.pro

Якщо вхідний Prolog-файл задано, аналіз програми забезпечується викликом функції analys():

```
void analys (char *nf)
{err=fopen_s(&PF,nf,"r"); // відкриття Prolog-файлу
 if(!err)  {get ();          // введення нової лексеми
            prog ();         // аналіз блоку програми
  }
  else printf("Error %s file isn't opened",nf);
 }
```

За введення лексем відповідає функція get():

```
void get ()
{if(nch= ='N')nch=getc(PF);
 if(nch= =EOF)  {lex=EOF;return;}
 while(isspace(nch))
{if(nch= ='N')nst++;nch=getc(PF);}
 if(isdigit(nch))number();
 else if(islower(nch))  word();
 else if(index(nch)) {lex=nch;
  nch=getc(PF);}
        else if(nch= =EOF)lex=EOF;
 else printf("\n error %c illegal simbol",nch); }
```

Введення лексем типу "ціле число" та "слово" забезпечується функціями number() та word():

```
void number ()
{for(lval=0;isdigit(nch);nch=getc(PF))
      lval=lval*10+nch-'0';
  lex=NUMB;  }
void word()
{static char
*serv[]={"domains","database","predicates","clauses","goal",
"integer","real","string","char","symbol",":-","write"};
```

```
static int cdl[]={DOMAINSL, DATABASEL,
PREDICATESL, CLAUSESL, GOALL, INTL, REALL,
STRINGL,CHARL,SYMBOLL,IFL,WRITEL};
char tx[40]={""},*p;
int i;
bool keyword=false;
if(nch==':'){p=tx;*(p++)=nch;nch=getc(PF);
            if(nch=='-'){*(p++)=nch;p=tx;nch='$';}
           }// окреме визначення лексеми ":-"
else for(p=tx;(islower(nch) || isupper(nch) || isdigit(nch) ||
nch=='_') && !isspace(nch) && !index(nch) &&
nch!=10;nch=getc(PF))  *(p++)=nch;
p="\0"; for(i=0;i<12;i++)
if(strcmp(serv[i],tx)==0){lex=cdl[i];keyword=true;break;}
if(!keyword){lex=IDEN;  lstr=add(tx);lval=(int)lstr;
}}
```

Пошук лексем-роздільників забезпечує index():

```
bool index(char b)
{bool ft=false;
 char a[16]={'(',')',',',';','=','+','-','*','/','%','[',']','.','|','>','<'};
 for(int i=0;i<16;i++)  if(a[i]==b){ft=true;break;}
 return ft;
}
```

Ідентифікатори, що не знайдені серед списку службових лексем та не введені раніше, мають бути додані до таблиці ідентифікаторів TNM[40], що й забезпечує функція add():

```
char *add (char * nm)
{char *p;
 for(p=TNM;p<ptn;p+=strlen(p)+1)
        if(strcmp(nm,p)==0)return (p);
 if(atoi(ptn+=strlen(nm)+1)>atoi(TNM+400))
       printf("\n overflow TNM");
 strcpy (p,nm);
 return p;
}
```

Зазначимо, що введення лексем проводиться послідовно, при цьому застосовується непряма рекурсія функцій get(), word(), number().

Синтаксис Prolog (відповідно [4]) описується розширеними граматиками із використанням символів "[", "]" в регулярних виразах у правих частинах правил.

Якщо r – регулярний вираз, що задає множину R, то [r] – регулярний вираз для множини $R \cup \{\varepsilon\}$, де $\varepsilon$ - порожній ланцюжок. Термінальними символами розширеної граматики є лексеми, записані малими літерами. Нетерміналами є послідовності літер, що позначають синтактичні конструкції програми: TYPEL – тип; TYPELS – список типів, DOMAINSL – розділ "domains", PREDICATESL – розділ "predicates", PARAMSL – список параметрів, CLAUSEL – розділ "clauses", PREDIMP – реалізація предиката, FACTLS – список фактів, RULELS - список правил, FACT – опис факту, CONSTLS – список атомів, ATOM - атом, RULE - правило, RULEL – ліва частина правила (консеквент), RULER – права частина правила (антецедент), STML – список операторів, STAT - оператор, PREDCALL – виклик предиката, GOAL – розділ "goal", PROG - програма, EXPR – вираз, TERM - доданок, FACT - множник.

Повний синтаксис мови Prolog пропонується у такому вигляді:

TYPEL → (intl | reall | stringl | symbol)
TYPELS – TYPE '*'
DOMAINSL → (iden=TYPEL | iden TYPELS) *
DATABASEL → (iden '(' PARAMSL ')') *
PREDICATESL → (iden '(' PARAMSL ')') *
PARAMSL → (TYPEL | TYPELS) (',' [TYPEL | TYPELS] )*
CLAUSESL → PREDIMP (PREDIMP) *
PREDIMP → FACTLS | RULELS
FACTLS → FACT (FACT) *
RULELS → RULE (RULE) *
FACT → iden (CONSTLS) '.'
CONSTLS → ATOM (',' ATOM) *
ATOM → iden | numb | var
RULE → RULEL ': - ' RULER
RULEL → iden '(' CONSTLS ')'
RULER → STML
STML → STAT (',' STAT) *
STAT → iden '=' (EXPR | PREDCALL)
PREDCALL → RULEL
PROG → (DOMAINSL | PREDICATESL | DATABASEL | CLAUSESL) GOAL eof
STML → STAT (';' STAT ) *
EXPR → [ '+' | '–' ] TERM ( ( [ '+' | '–' ] ) TERM)*
TERM → FACT ( ( '*' | '/' | '%' ) FACT )*

Конструкція PROG забезпечує обробку тексту файла програми і відповідає функції prog():

```
void prog()
{fnd *p;
 lex;
 while(lex!=EOF)
 {switch(lex)
       {case DOMAINSL:          ddomains();break;
        case DATABASEL:         ddatabase(); break;
        case PREDICATESL:       dpredicates(); break;
        case CLAUSESL:          dclauses(); break;
        case GOALL:             dgoal(); break;
        default:  printf("\n %d syntax error (prog)",nst);
       }
  if(ef){cout<< "\n Predicates sequence error";break;}
 }
 p=fgoal();
 if(p){adrnm=p->start;cpnm=p->cpt;} }
```

Розгляд розділу "domains" Prolog-програми здійснюється функцією ddomains():

```
void ddomains()
{get();
 char *nm=(char*)lval;
 newob(nm,"domains",0,0);
```

```
bool noalt=true;
do
{if(noalt){exam(IDEN); exam('='); }
 switch(lex)
   {case INTL:          exam(INTL);break;
    case STRINGL:       exam(STRINGL);break;
    case REALL:         exam(REALL);break;
    case CHARL:         exam(CHARL);break;
    case SYMBOLL:       exam(SYMBOLL);break;
   }
  lextestmode=true;
  if(exam(';'))noalt=false;
  else {noalt=true;exam('*');}
       lextestmode=false;
 }while(lex!=PREDICATESL && lex!=DATABASEL);
}
```

Синтаксичний аналіз використовує функцію exam(), яка перевіряє, чи має наступна лексема код lx. Якщо відповідність коду не знайдена, виводиться повідомлення про синтаксичну помилку:

```
bool lextestmode=false;
bool exam(int lx)
{if(lx!=lex)
 {if(!lextestmode)printf("\n %d syntax error (exam)
                          %c",nst,lex);
                  return false;}
 get();
 return true;
}
```

Обробка розділу "database" ведеться за допомогою функції ddatabase():

```
void ddatabase()
{do   {get();
       char *nm=(char*)lval;
       newob(nm,"database",0,-1);
       if(lex=='(') {pred_args();
                     exam(')');   }
      }while(lex!=PREDICATESL);
}
```

Обробка предикатів мови Prolog має забезпечуватися викликом dpredicates(), у відповідності до конструкції DPREDICATESL:

```
void dpredicates()
{do   {get();
       char *nm=(char*)lval;
       int cp=0,st=0;
       if(lex=='(') {pred_args(); exam(')');
                     defin(nm,cp,st);
                    }
      }while(lex!=CLAUSESL);
}
```

Обробка розділу database ведеться у аналогічний спосіб.

У своєму складі dpredicates() містить послідовні виклики функцій: get() – введення чергової лексеми, pred_args () – обробки параметрів функції:

```
bool pred_args()
{do
 {get();
 switch(lex)
      {case IDEN:        exam(IDEN);break;
       case INTL:        exam(INTL);break;
       case STRINGL:     exam(STRINGL);break;
       case REALL:       exam(REALL);break;
       case CHARL:       exam(CHARL);break;
       case SYMBOLL:     exam(SYMBOLL);break;
  }
 }while(lex==',');
return true;
}
```

Наступний розділ Prolog-програми – "clauses" обробляється функцією dclauses():

```
void dclauses()
{bool bclauses=false;
 do
      {get();lstr;
       if(!bclauses)exam(IDEN);
       switch(lex)
                {case '.': break;
                 case IFL: pred_exec(lstr);exam('.');break;
                 case '(': pred_pars(lstr);exam(')');
                   switch(lex)
                   {case '.': exam('.');break;
                    case IFL: pred_exec(lstr);
                              exam('.');break;
                   }break;
                }
       bclauses=true;
      }while(lex!=GOALL);
  }
```

Виконання предиката забезпечує pred_exec():

```
void pred_exec(char * lstr1)
{int st; int cp=0;bool g=false;
 if(!strcmp(lstr1,"goal"))g=true;
 odc *p;
 fnd *pl;
 do    {get();
        switch(lex)
        {case WRITEL: fact();gen(OPR,2);break;
         case IDEN:    p=findob(lstr);
                  if(!p) newob((char*)lval,lstr1,
                           (out?3:4),(out?cgv++:++clv));
                  p=findob(lstr);
                  exam(IDEN);
                  switch(lex)
                  {case '=': exam('=');expr();
                             if(p->what==1)
                       printf("\n %s isn't variable",p->name);
                             gen(STI,p->val); break;
                  case '>': stat();break;
                  case '<': stat();break;
                  case '(': pred_pars(lstr);
                  cp=(lex==')' ? 0 : fctl());
                        exam(')');
                  pl=eval(lstr1,cp);
                  gen(LIT,cp);cp=gen(CAL,pl->start);
                  if(!pl->isd)pl->start=cp; break;
       case ',': break;
       default : ef=true;break;
       }}
        if(ef)break;
       }
```

```
while(lex==',');
if(!g)gen(OPR,9);
st=gen(INI,clv); gen(OPR,10); }
```

Обробка параметрів предиката відбувається окремою функцією pred_pars():

```
void pred_pars(char *lstr1)
{odc *p; int cp=0;
 do   {get();
       switch(lex)
       {case '[': {test_list(lstr1); exam(']'); break; }
        case IDEN: if(!findob(lstr))
                     newob((char*)lval,lstr1,3,++clv);
                   exam(IDEN);
                    if(lex=='('){pred_pars(lstr1);
                    exam(')');break;}
                   break;
        case NUMB: exam(NUMB);break;
       }
      }
 while(lex==',');
 for(p=ptol;p<pto;p++)p->val-=cp+3;
}
```

Обробка списків відіграє значну роль у функціонуванні Prolog-програми, тому для їх обробки призначається окрема функція:

```
void test_list(char *lstr)
{      get();
       switch(lex)
       {case ']':  break;
        case IDEN: get();
                    switch(lex)
                   {case ']': break;
                    case '|': pred_pars(lstr);break;
                    case ',': test_list(lstr);break;
                   }
                    break;
       case NUMB: get();
                    switch(lex)
                    {case ']': break;
                     case '|': pred_pars(lstr);break;
                     case ',': test_list(lstr);break;
                    }
                    break;
       }
}
```

Розділ goal у мові Prolog відіграє роль своєрідної функції main() (у розумінні Сі-програми). Тому, фактично у ньому відбувається виклик "предиката" з іменем "goal":

```
void dgoal()
{defin("goal",0,0);  pred_exec("goal");  exam('.'); }
```

Синтаксична конструкція STML забезпечує обробку списку операторів, хоча детальну обробку забезпечує найбільш складна конструкція мови – STAT. Цим схемам оброки відповідають дві функції: stml () та stat (). Власне, перша з них здійснює послідовний (в разі потреби) виклик другої – із безпосереднім аналізом поточного ідентифікатора – iden, числової константи numb, символів '[', '(' та '='.

```
void stml()
{stat();   while(lex==';') {get();  stat();}}

void stat()
{odc *p;  int t1,t2;  get();
 switch(lex)
{case IDEN: p=findob((char*)lval); get();
       if(lex==',' ||lex == ')')break;
       expr();
       if(p->what==1)printf("\n %s isn't variable",p->name);
       gen(STI,p->val);
       break;
 case NUMB: gen(LIT,lval); get();
       gen(STI,cnl++);
       if(lex==')')break;
       if(lex!=',') {expr();gen(STI,cnl++); } break;
 case '[' : test_list("");get();break;
 case '(' : term(); gen(5,1); break;
 case '=' : get();expr();break;
 default:   printf("\n %d syntax error (stat)",nst);
}}
```

Як видно зі схеми конструкції STAT, її складовими є вирази EXPR або списки операторів STML. EXPR є комбінацією операцій додавання, віднімання, множення, ділення виразів та констант і складається з комбінацій конструкцій TERM, та опосередковано – FACT і FCTL. Вказаним вище конструкціям відповідають функції expr(), term(), fact(), fctl():

```
void expr()
{int neg=(lex=='-');
  int plusmin=0;
 if(lex=='+' || lex=='-'){plusmin=1;get();}
 term();
 if(neg)gen(OPR,8);
 while(lex=='+' || lex =='-')
 {neg=(lex=='-'?4:3);
  get();term();
  gen(OPR,neg);
 }
 if(lex=='.' && plusmin)
 {neg=(lex=='-'?4:3);
  gen(OPR,neg);
 }}

void term()
{int op;
  fact();
 while(lex=='*' || lex =='/' || lex=='%')
 {op=(lex=='*' ? 5 : (lex=='/' ? 6:7));
  get();fact();gen(OPR,op);}}
int fctl ()
{int cf=1;  expr();
 while(lex = = ',' ) {get();expr();cf++;}
 return cf;
}
void fact()
{char *nm;
 int cp;
 odc *p;fnd *pl;
 switch(lex)
 {case IDEN:   nm=(char*)lval; get();
```

```
                    if(lex=='(')
                    {get();  if(lex==')')fctl();
                     exam(')');
                     pl=eval(nm,cp);
                     gen(LIT,cp);cp=gen(CAL,pl->start);
                     if(!pl->isd)pl->start=cp;
                    }
                    else {p=findob(nm);
 if(!error_syntax && p) gen(p->what==1 ? LIT :
                              (p->what==2 ? LDE:LDI),p->val);
                    }
                    break;
  case WRITEL: get();fact(); break;
  case '(':  get();expr();exam(')'); break;
  case NUMB: gen(LIT,lval); get(); break;
  case '.':  break;
  case '/':  break;
  case '*':  break;
  default:          printf("\n %d syntax error (fact)",nst);
 }
}

void fact()
{char *nm;
 int cp;
 odc *p;fnd *pl;
 switch(lex)
 {case IDEN: nm=(char*)lval;get();
        if(lex = ='(' )
        {get();cp=(lex= =' ) ' ? 0 : fctl());
          exam(')');pl=eval(nm,cp);
          gen(LIT,cp);cp=gen(CAL,pl->start);
          if(!pl->isd)pl->start=cp;
        }
       else { p=findob(nm);  gen(p->what==1 ? LIT :
                  (p->what==2 ? LDE:LDI),p->val);
            }
        break;
  case '(':  get();expr();exam(')');break;
  case NUMB: gen(LIT,lval);get();break;
  default:          printf("\n %d syntax error (fact)",nst);
}}
```

Слід одразу зазначити, що у текстах наведених функцій реалізації синтаксичного аналізатора включені додаткові функції контролю таблиць, пов'язані вже з інтерпретацією програми.

## 6. ПРОГРАМНА РЕАЛІЗАЦІЯ ФУНКЦІЙ ІНТЕРПРЕТАТОРА

Результатом роботи блоку синтаксичного аналізу є проміжний код, що надалі обробляється функціями інтерпретатора програми.

Набір функцій інтерпретації формує Prolog-процесор (аналогічний описаному у [7, 8] для мови програмування SPL), який складається: з пам'яті програми, що моделюється масивом TCD та змінними adrnm, cpnm, cgv; стеку st зі значеннями змінних та констант програми. Стек st характеризується покажчиками: вершини стека – t, початку області локальних даних функції – sp (запису активації). Покажчик p – вказує на поточну команду, обрану у STD.

Попереднім, до розробки транслятора мови Prolog, кроком була програмна реалізація інтерпретатора мови програмування SPL, на основі матеріалів, описаних у [7]. Тому процес відлагодження Prolog-транслятора полягав у постійному порівнянні початкових даних, що готуються на етапах лексичного та синтаксичного аналізу обох програм. При цьому, для повторного використання коду, текст блоку інтерпретації практично не змінювався, хоча з точки зору практичної реалізації та розуміння роботи він не є ідеальним, особливо з точки зору мови програмування Prolog.

## 7. ВИСНОВКИ

Таким чином, у запропонованій статті з практичної точки зору розглядається побудова інтерпретатора мови програмування Prolog. Наводиться програмна реалізація блоків лексичного та синтаксичного аналізу. Розв'язання розробки транслятора вимагає великого об'єму копіткої роботи із практичної реалізації розроблених теоретичних моделей, тому значну частину матеріалу займає код функцій інтерпретатора.

Звичайно, що подібний опис є лише першим кроком на шляху творення повноцінного транслятора. Перші тестування виявили його здатність коректно виконувати нескладні Prolog-програми. Наступними кроками є відпрацювання схем автоматичного прийняття рішень для задачі функціонування робота спочатку у статичному просторі, потім – додавання засобів адаптації до змін у робочому просторі.

Наукова цінність роботи полягає у розробці та практичній реалізації граматик мови Prolog. Розроблені граматики можуть використатися для моделювання Prolog за допомогою інших мов програмування.

Практична значимість результатів роботи полягає у можливості їх використання для розробки нових і самостійної реалізації власних трансляторів вже існуючих мов програмування.

Кінцевою метою розробки транслятора на даному етапі є його доведення до працездатного стану та його інтеграція у Visual C++ проект, що забезпечуватиме моделювання системи підтримки та прийняття рішень інтелектуального робота.

## 8. СПИСОК ЛІТЕРАТУРИ

[1] Малпасс Дж. *Реляционный язык Пролог и его применение*. – М.: Наука, 1990. – 464 с.

[2] Люгер Дж.Ф. *Искусственный интеллект: стратегии и методы решения сложных*

*проблем*. – М.: Изд. дом «Вильямс», 2003. – 894 с.

[3] Гаврилова Т.А., Хорошевский В.Ф. *Базы знаний интеллектуальных систем*. – СПб.: Питер, 2001. – 384 с.

[4] Ахо А., Сети Р., Ульман Дж. - *Компиляторы: принципы, технологи, инструменты*. – М.: Изд. дом «Вильямс», 2003. – 768 с.

[5] Гордеев А.В., Молчанов А.Ю. *Системное программное обеспечение*. – СПб.: Питер, 2002. – 736 с.

[6] Карпов В.Э. *Классическая теория компиляторов*. – М.: Московский государственный ин-ститут электроники и математики, 2003. – 96 с.

[7] Проценко В.С., Чаленко П.Й., Ставровський А.Б. *Техніка програмування мовою Сі*. – К.: Либідь, 1993. – 224 с.

[8] Бєлов Ю.А., Проценко В.С., Чаленко П.Й. *Інструментальні засоби програмування*. – К.: Либідь, 1993. – 248 с.

***Цимбал Олександр Михайлович**, канд. техн. наук, докторант, Харківський національний університет радіоелектроніки. Наукові інтереси – мови програмування, системи штучного інтелекту. Адреса: Харків, пр. Леніна 14, ХНУРЕ, каф. ТАВР., тел. (057) 7021486.*

# THE IMPLEMENTATION OF PROLOG INTERPRETER: LEXICAL AND SYNTAX ANALYSES

**Alexander Tsimbal**

Kharkiv national university of radio electronics,
Kharkiv, Lenin Ave. 14, mcdulcimer@ukr.net

**Abstract:** *The article considers the development of Prolog programming language interpreter with lexical and syntax analyzer, table control and interpreter blocks. There is proposed the schemes of syntax analyzes and the elements of translator's program code. The software is developed in Visual C++ 2005 (Beta) environment.*

**Keywords:** *Programming language, interpreter, Prolog, grammar, lexical analysis, syntax analysis.*

Prolog (Programming in Logic) implements the data processing by program construction in form of predicates. The main option of Prolog is practical implementation of logical knowledge model [1].

The development of artificial intelligence methods and the need of their language support set the task of Prolog modernization. The development of new Prolog translator presents an actual problem, aimed to solve the robotics tasks.

Prolog applications are especially suitable for knowledge representation tasks, namely for production and framework models [2,3]. Also, the recursive list processing procedures give the great possibilities for informational systems organization, for problem solving in various fields including robotics [3, 4]. The interesting way of Prolog application is described in [6], where the reversal task is set – the principles of other language implementation are described in Prolog.

The review of possibilities to construct new Prolog implementation requires the using of standard methods, described in [4]. But as to practical view point it's useful to apply the program constructions described in [7, 8]. After the insignificant transformations these materials can be easily used for the translator development.

Robotics requires the decision of several problem classes: from the one hand – the tasks of calculations in cinematic models, which can be decided by universal programming language application, for instance C. Also C-language (or Assembler) is good during the control signals forming and transmission to robot control system.

From the other hand – on decision support level, the calculation is supplementary aspect, but logical programming is the key issue. Therefore, the common task of robotic systems simulation can be presented as the set of heterogeneous tasks, decisions of which will require the application of various program methods. There is proposed to implement the intellectual robot decision support system on base of Prolog. To supply it's functionality as system part it's required to develop the interpreter of Prolog programming language.

The article objective is a description of main issues of Prolog translator development. There were set the tasks of Prolog grammar description, the lexical and syntax analyzers implementation. The development tool is C-language.

Prolog language interpreter has syntax similar to Turbo-Prolog, developed by Borland Inc. There will be used the principles of [7], applied during simple programming language translator (SPL) creation.

The alphabet of Prolog consists of Latina characters, decimal values, the service characters: {'(', ')', ',', ';', '=', '+', '-', '*', '/', '%' ,'[',']', '.', '|', '>', '<'} and of space. The alphabet characters construct the lexemes – the minimal language units.

The Prolog identity is a sequence of letters and digitals, beginning of character. Unsigned value is a sequence of digitals. From the syntax view point all the identities are the same and correspond lexeme IDEN. Unsigned digitals present the lexeme NUMB.

Except IDEN and NUMB Prolog has twelve special keyword - {"domains", "database", "predicates", "clauses", "goal", "integer", "real", "string", "char", "symbol", ":-", "write"}.

In the current implementation Prolog contains the scalar data of integer, real, symbol and string type. Also the sequence of certain type values can form the list of elements. The elements of list are divided by commas inside the square brackets: [12, 31, -5, 6] or [‘a’, ‘d’, ‘E’]. The identities of Prolog variables are written by capital letter, the constants – by small. All the variables are local inside predicates.

The expression of Prolog consists of constants, variables and predicate names (with round brackets for parameter setting or without them). The

operation signs are "+", "-", "*", "/", "%". Expressions have view: v, c, ($e_1$), predicate($e_1$, $e_2$,…,$e_n$ ), $e_1 + e_2$, $e_1 - e_2$, $e_1 * e_2$, $e_1 / e_2$, $e_1 \% e_2$, $+ e_1$, $-e_1$, here v – variable, c – constant, predicate – the predicate name, $e_1$, $e_2$,…,$e_n$ – expressions.

Prolog has in-build operators of assignment (comparison) v=c; and of print - write(v).

Prolog program has logical and / or computational contents. It has the defined structure and consists of the following sections:

| | |
|---|---|
| domains | The definition of type names, complex types and lists; |
| database | The definition of facts of Prolog database; |
| predicates | The declaration of predicate prototypes; |
| clauses | The implementation of predicates; |
| goal | The call of predicates or the program running. |

During the lexical analyzes, there is defined the lexemes of program, including the keyword "domains", "database", "predicates", "clauses", "goal", "integer", "real", "string", "char", "symbol", ":-", "write".

The syntax of Prolog (as to [4]) is described by extended grammars with using of characters "[", "]" in regular expressions.

If r is a regular expression which defined the set R, then [r] is a regular expression for set $R \cup \{\varepsilon\}$, here $\varepsilon$ - is empty chain. The terminal characters of extended grammar are lexemes written by small letters. Non-terminals are letter sequences defining the syntax constructions of program.

There is defined the following non-terminals:
TYPEL – the type; TYPELS – the type of list, DOMAINSL – "domains" section, PREDICATESL – "predicates" section, PARAMSL – parameters list, CLAUSEL – "clauses" section, PREDIMP – predicate implementation, FACTLS – the list of facts, RULELS – the list of rules, FACT – the fact description, CONSTLS – the list of atoms, ATOM – the atom, RULE – the rule, RULEL – the left part of the rule (consequent), RULER – the right part of the rule (antecedent), STML – the operators list, STAT – the operator, PREDCALL – the predicate call, GOAL – "goal" section, PROG - program, EXPR – expression, TERM - sum, FACT - multiplication.

The general syntax of Prolog language is proposed in the following view:

TYPEL → (intl | reall | stringl | symbol)
TYPELS – TYPE '*'
DOMAINSL → (iden=TYPEL | iden TYPELS) *
DATABASEL → (iden '(' PARAMSL ')') *
PREDICATESL → (iden '(' PARAMSL ')') *
PARAMSL → (TYPEL | TYPELS) (',' [TYPEL | TYPELS] )*
CLAUSESL → PREDIMP (PREDIMP) *
PREDIMP → FACTLS | RULELS
FACTLS → FACT (FACT) *
RULELS → RULE (RULE) *
FACT → iden (CONSTLS) '.'
CONSTLS → ATOM (',' ATOM) *
ATOM → iden | numb | var
RULE → RULEL ': - ' RULER
RULEL → iden '(' CONSTLS ')'
RULER → STML
STML → STAT (',' STAT) *
STAT → iden '=' (EXPR | PREDCALL)
PREDCALL → RULEL
PROG → (DOMAINSL | PREDICATESL | DATABASEL | CLAUSESL) GOAL eof
STML → STAT (';' STAT ) *
EXPR → [ '+' | '–' ] TERM ( ( [ '+' | '–' ] ) TERM)*
TERM → FACT ( ( '*' | '/' | '%' ) FACT )*

Every construction is presented by C-language function implementation.

The intermediate code, which is a result of syntax analyzes stage, should be processed by interpreter functions of translator. This functions set forms the Prolog processor (analogical to the described in [7, 8] for SPL).

Therefore, the proposed article practically considers the development of Prolog programming language interpreter.

The mentioned description is the first stage on way to full-scale translator. The next steps will include the special settings to implement the robot problem solver, firstly for static and after – for the dynamically changed world.

The scientific value of article is in development and practical implementation of Prolog language grammar. The developed grammars can be used for Prolog simulation on other software base.

The practical value of article consists in their applications for new the translator's development.

The future objective of translator development is it's integration to more general project of intellectual robot decision support system.