УДК 519.6

Вовк Н. Є.

МЕТАЛОГІЧНІ ПРЕДИКАТИ В ПРОЛОЗІ

У статті розглянуто деякі можливості розширення мови логічного програмування Пролог, а саме металогічні предикати (метапредикати).

Ключові слова: логічне програмування, Пролог, металогічні предикати, метапредикати.

Металогічні предикати (метапредикати) слугують корисним розширенням для підвищення виразності логічних програм. Ці предикати виходять за рамки логіки першого порядку, оскільки потрібні для аналізу структури доведення, розглядають змінні (а не позначені ними терми) як об'єкти мови і допускають перетворення структур даних на цілі.

Виділяють чотири основні класи металогічних предикатів [2; 3; 4]:

- типові предикати, що визначають, чи є терм змінною;
- предикати для оцінки, розгляду та перебудови структури термів;
- предикати, що дозволяють розглядати змінні як об'єкти перетворення;
- засоби, що дозволяють перетворювати дані на оброблювані цілі.

Типові металогічні предикати

Основним типовим предикатом є предикат *var(Term)*, який дозволяє визначити, чи є терм *Term* змінною з невизначеним значенням [1; 2]. Поведінка цього предиката аналогічна до поведінки класичних типових предикатів (*integer*, *atom* та ін.). Запит *var(Term)* успішний, якщо *Term* змінна, і неуспішний якщо *Term* вже уніфікований, наприклад:

:- var(X) → *yes;*

:- var(a) → *no.*

Предикат *var* є розширенням Прологу, оскільки неможливо побудувати таблицю імен для задання всіх змінних.

Значення предиката *nonvar(Term)* протилежне для значення предиката *var(Term)*.

Типові предикати можна використовувати для надання програмам певної гнучкості, універсальності. Прикладом застосування може бути перевизначення системного предиката *length(X,N)* – обчислення довжини *N* списку *X*, що дозволить не лише обчислення довжини списку, а й згенерувати список заданої довжини [2].

```
length_(A, B) :- nonvar(A), length1(A, B).

length_(A, B) :- var(A), nonvar(B),length2(A, B).

length1([], 0).

length1([_|Xs], N) :- length1(Xs, N1),  N is N1 + 1 .

length2([], 0).

length2([_|Xs], N) :- N > 0, N1 is N – 1,
    length2(Xs, N).
```

Програма 1. Визначення універсального предиката length

Аналіз структури термів, уніфікація

Для аналізу структури термів використовуються метапредикати *functor(Term,Name,Arirty)*, *arg(N,Term,Arg)* та оператор *Term..=List* [1; 2; 4].

functor(Term, Name, Arirty) дозволяє отримати інформацію про структурний терм–ім'я функтора та кількість аргументів або створити нову структуру із заданим ім'ям та заданою кількістю компонент. Предикат *arg(N,Term,Arg)* успішний, якщо *N*-тий аргумент терму *Term* збігається зі значенням *Arg*.

Застосування предикатів аналізу структури термів добре демонструє програма 2 [1] – явне визначення уніфікації термів у Пролозі. Відношення unify(Term1,TEPM2) істинно, якщо терм Term1 уніфікується з термом Term2 [2]. Правила програми unify описують можливі випадки. Перше речення програми стверджує, що дві змінні є уніфікованими. Наступне речення є записом правила уніфікації, яке стверджує: якщо X змінна, то X уніфікується з Y.

Інший випадок, що є предметом розгляду програми 2, полягає в уніфікації двох складових термів, як це описано предикатом *termjunify (X, Y)*. У цьому предикаті перевіряється, чи мають два терми *X* і *Y* однакові головні функтори та арність, а потім перевіряється, чи всі аргументи попарно уніфікуються.

```
unify(X,Y) :- var(X), var(Y), X=Y.
unify(X,Y) :- var(X), nonvar(Y), X=Y.
unify(X,Y) :- var(Y), nonvar(X), Y=X.
unify(X,Y) :- nonvar(X), nonvar(Y), constant(X),
    constant(Y), X=Y.
unify(X,Y) :- nonvar(X), nonvar(Y), compound(X),
    compound(Y), term_unify(X,Y).
term_unify(X,Y) :- functor(X,F,N), functor(Y,F,N),
    unify_args(N,X,Y).
unify_args(N,X,Y) :-N>0, unify_arg(N,X,Y), N1 is
    N-1, unify_args(Nl,X,Y).
unify_args(0,X,Y).
unify_arg(N,X,Y) :- arg(N,X,ArgX), arg(N,Y,ArgY),
    unify(ArgX,ArgY).
```

Програма 2. Алгоритм уніфікації

Оператор *Term..=List* також використовується для аналізу структур термів. Ціль *Term..=List* успішна, якщо голова списку *List* збігається з ім'ям основного функтора терму *Term*, а хвіст є списком аргументів даного терму. Наприклад, ціль *owns(jane,cat)..=[owns,jane,cat]* успішна.

Використання змінних як об'єктів

У зв'язку зі специфікою мови, робота зі змінними в Пролозі пов'язана з деякими складностями, та при аналізі та обробці термів між змінними та значенням можуть виникнути випадкові співставлення.

Така проблема з'являється при побудові програми 3, що визначає відношення *substitute.* Якщо розглянути ціль *substitute(a,b,X,Y),* що задає підстановку константи *a* замість *b* в змінну *X.* Результатом буде *Y.* У такому випадку є дві інтерпретації відношення s*ubstitute*. Логічно обґрунтованим рішенням є співставлення змінній *X* константи *a,* а змінній *Y* – константи *b.* Цей розв'язок програма справді знаходить при уніфікації цілі з основним *substitute (Old,New,Old,New).*

```
substitute(Old,New,Old,New). substitute(Old,New,
    Term,Term):-
atom(Term), Term\=Old.
substitute(Old,New,Term,Term1):-
compound(Term),
functor(Term,F,N),
functor(Term1, F,N), substitute(N,Old,New,Term,
    Term1).
substitute(N,Old,New,Term,Term1):-
N>0,
arg(N,Term,Arg), substitute(Old,New,Arg,Arg1),
    arg(N,Term1,Arg1),
N1 is N-1,
substitute(N1,Old,New,Term,Term1).
substitute(0,Old,New,Term,Term1).
```

Програма 3. Програма підстановки термів

Однак на практиці перевагу має інша інтерпретація. Вважатимемо терми *Y* та *a* різними, таким чином змінній *Y* слід співставити терм *X*. Таке можливо при виконанні другого правила програми 3, а саме:

substitute(Old,New,Term,Term):- atom(Term), Term\=Old.

Однак цей варіант не буде успішним, оскільки змінна не є константою.

Можна уникнути першого рішення, застосовуючи металогічний тест, що гарантує підстановку в основний терм. Тоді уніфікація мусить бути задана явно. Основний факт перетворюється на правило:

substitute(Old,New,Term,New):-ground(Term), Term=Old.

Розгляд змінних окремо від констант вимагає спеціального правила, що також використовує металогічний тест:

substitute(Old,New,Var,Var):-var(Var).

Доповнення програми 3 згаданими вище правилами, а також додавання інших металогічних тестів дозволяє застосовувати програму для неосновних термів.

Програма «створює» програму

Характерною особливістю Прологу є еквівалентність програм і даних: одне і друге можуть бути представлені логічними термами для того, щоб використовувати цю еквівалентність, необхідно, щоб програми можна було розглядати як дані, а дані можна було перетворювати на програми [4]. Засобом перетворення терму на ціль є предикат *c*all(X). Він передає терм X як ціль для розв'язку задачі.

Ще одним предикатом, що дозволяє розглядати програму як дані, є *clause(Head,Body)***. Предикат** *clause(Head,Body)* успішний, якщо в базі даних (програмі) існує відповідне правило

Head:-Body. Проілюструвати його роботу можна на прикладі програми 4 – навчального інтерпретатора мови Пролог [3]. Предикат *solve* отримує як аргумент цільове твердження та обробляє його згідно із семантикою Прологу:

```
solve(true):-!.

solve((A,B)):-!,solve(A),solve(B).

solve(A):-clause(A,B),solve(B).
```

Програма 4. Навчальний інтерпретатор мови Пролог

Припустимо, що існує певна програма

```
p(X,Y):-q(X),r(Y).

q(X):-s(X).

r(X):-t(X).

s(a).

t(b).

t(c).
```

Тоді предикат *solve* буде поводитися як інтерпретатор Прологу:

```
?- solve(p(a,b)). → yes

?- solve(p(X,Y)). →

X = a   Y = b ?

X = a   Y = c

yes

?- solve(p(f,q)). → no
```

Висновки

Звичайно, у цій роботі розглянуто далеко не всі метапредикати Прологу, але вже можна зробити деякі підсумки: Пролог – майже унікальна мова, яку можна розширювати, використовуючи її власні метапредикати; програма на Пролозі може породжувати нові цілі та виконувати їх; програма може аналізувати та модифікувати сама себе; програма може аналізувати структуру своїх змінних та використовувати їх як частину самої програми.

Список літератури

1. Братко И. Программирование на языке Пролог для искусственного интеллекта / И. Братко ; [пер. с англ.]. – М. : Мир, 1990. – 560 с.
2. Стерлинг Л. Искусство программирования на языке Пролог / Л. Стерлинг, Э. Шапиро ; [пер. с англ.]. – М. : Мир, 1990. – 235 с.
3. Люгер Джордж Ф. Искусственный интеллект: стратегии и методы решения сложных проблем / Люгер Джордж Ф. ; [пер. с англ.]. – 4-е изд. – М. : Вильямс, 2003. – 864 с.
4. Covington Michael A. Prolog programming in depth / A. M. Covington, D. Nute, A. Vellino. – Prentice Hall. Upper Saddle River. New Jersey, 1997. – 516 p.

N. Vovk

METALOGICAL PREDICATES IN PROLOG

This paper examines some possibilities of expanding logical programming language Prolog, namely metalogical predicates (metapredicates).

Keywords: logical programming, Prolog, metalogical predicates, metapredicates.

Матеріал надійшов 05.05.2014