УДК 519.68

Д. Зербіно, Ю. Цимбал
Національний університет “Львівська політехніка”,
кафедра автоматизованих систем управління

ОСНОВНІ КОНЦЕПЦІЇ МОВИ ПРОГРАМУВАННЯ “OPEN PROLOG”

Розглянуто недоліки існуючої мови PROLOG для розв’язання логічних задач. Для прикладу взята класична задача на комбінаторний перебір варіантів. Показано, що розв’язок буде елементарним, якщо ввести оператори типу “припустимо, що існує ...” та “припустимо, відсутнє ...”. Авторами створено програмне ядро для реалізації нової версії мови PROLOG.

The most distinct disadvantages of the PROLOG language for solving the logical problems are presented in the article. A classical problem on combinatorial depth-first search was given as an example. It is shown that the solution will be elementary in case of putting into the new type of operations “something is supposed to exist” and “something is supposed to be absent”. The kernel of the program for the realization of the new language was created by the authors.

Вступ

З розвитком комп’ютерних технологій постійно зростає необхідність появи нових та вдосконалення існуючих мов програмування високого рівня абстракції. Що ми розуміємо під «якістю» мови програмування? На наш погляд, це зручність, точність і швидкість формулювання завдань для комп’ютера. Розглянемо кожну властивість окремо. Зручність – це можливість наближення логіки програмної системи до логіки програміста. Зручність забезпечується діалоговими інтерфейсами різних типів, зокрема, графічними та на рівні лексики та синтаксису самої мови. Програміст оперує абстрактними, тобто нечіткими категоріями. Що неконкретніше завдання – тим легше його сформулювати і тим складніше його виконати. На перший погляд, це суперечить другій задекларованій нами властивості мови – точності. Сформулювати спочатку точне завдання доволі важко, тому для виконання завдання його надалі необхідно уточнити. Якщо завдання не уточнюється, то це означає, що програміст має на увазі певну стандартну ситуацію, яка обумовлена в даному конкретному випадку. Тут ми переходимо до третьої властивості – швидкості формулювання обчислювального завдання. Засіб спілкування повинен містити базу стандартних ситуацій, на яку може спиратися програміст під час спілкування з комп’ютером.

Найбільш інтелектуальними мовами програмування сьогодні вважається PROLOG [1,2] або подібні до неї. Такі мови можуть робити припущення, тобто в тому випадку, якщо програміста не задовольняють результати, обчислення можна відмінити, а програму повернути у стан, в якому існували альтернативні розгалуження процесу.

Пропозиції щодо вдосконалення існуючої мови програмування PROLOG

Насамперед необхідно сформулювати наявні недоліки та обмеження такої мови програмування високого рівня абстракції, як PROLOG. Розглянемо приклад: нехай потрібно швидко розв’язати таку навчальну логічну задачу.

Туристам, серед яких є діти і собака, необхідно переїхати на поромі річку. На поромі можуть їхати не більше двох осіб. Поромом дозволено керувати лише дорослим. Окрім того, для групи людей, що знаходяться на поромі або на березі, існують такі обмеження: Маша не може бути разом з Сашком або з Сергієм без присутності Павла. Павло також не може бути разом з Дарією або Діаною без присутності Маші. Бультер’єр не може бути разом з вказаними людьми без присутності Хазяїна. Дорослими вважаються Хазяїн, Маша, Павло.

Задача розв’язується досить просто перебором за допомогою рекурсії. Проблема полягає лише в тому, щоб цей перебір швидко і зручно задати. Логічні задачі найкраще розв’язуються мовою PROLOG. Спробуємо сформулювати цією мовою обчислювальне завдання.

В основу мови покладено поняття логічного предикату. В найпростішому вигляді – це звичайна таблиця, до якої можна додавати дані або з якої їх вилучати. Перевезення осіб на поромі можна представити як переміщення даних з однієї таблицю до іншої. У тому випадку, коли послідовність переміщень зайшла в глухий кут, можна відмінити операцію і повернутися на довільну кількість кроків назад. Перебір продовжується доти, доки не знайдеться розв'язок – послідовність переміщень даних від початкової до кінцевої таблиці.

Основний недолік сучасних PROLOG-систем полягає в неможливості відміни операцій над даними в таблицях. Концепція припущень під час програмування націлена на роботу зі змінними, а не з даними, які знаходяться в таблицях. Цей недолік пропонується усунути у новій версії мови PROLOG під робочою назвою Open PROLOG.

Формально робота з припущеннями в таблицях може бути представлена як оператори "***Suppose_deleting (предикат))***" – припустити відсутність заданих даних в предикаті та "***Suppose_appending (предикат)***" – припустити наявність заданих даних в предикаті. Просто – без приставки "*Suppose*", ключові слова "*delete*" та "*append*" означатимуть відповідно знищення або введення даних без можливості відмінити ці операції.

У сучасній мові PROLOG відсутні глобальні змінні, що значно ускладнює програмування звичайних задач. При цьому навіть елементарні арифметичні операції виконувати досить важко. Отже, пропонується ввести глобальні змінні для спрощення реалізації операцій типу "підрахувати кількість розв'язків".

У класичній версії відсутній такий вбудований предикат, як цикл, який видавав би послідовні значення в будь-якому проміжку від мінімального значення до максимального. Введення такого вбудованого предикату, як "LOOP" у версії Open PROLOG значно спрощує програмування, а також саму програму.

Істотним недоліком самої концепції предикату є те, що його значення вважаються невпорядкованими, і тому немає операторів, які б впливали на цей порядок. У новій версії пропонується ліквідувати цей недолік і ввести прості і зручні "**стандартні**" предикати "*MAX*", "*MIN*" і "*SORT*", які дадуть змогу знайти максимальне і мінімальне значення або впорядкувати предикати за значеннями певного аргументу для подальшого розгляду (наприклад, за ступенем важливості).

Класична версія мови PROLOG передбачає розділ "GOAL" – мета програми, але насправді в ньому пишеться алгоритм розв'язання задачі у вигляді послідовності підцілей, яких треба досягти. Цей розділ пропонується розділити на дві частини: "власне мета", наприклад, перевезти туристів на правий бік річки, та "алгоритм", який описує послідовність предикатів.

Під час розв'язання складних системних задач у аналітика виникає необхідність давати певну інтерпретацію своєму пошуку (наприклад, графічну). Таку інтерпретацію не дозволяє робити класична версія PROLOG. Цей недолік можна виправити, якщо зробити мову відкритою для під'єднання інших модулів іншими мовами програмування.

Особливості реалізації мови Open PROLOG

Для реалізації концепції повернення програми у попередній стан необхідно використовувати стек стану програми. У цей стек записуються змінні, а також вміст звичайного стеку в момент, коли програма натрапляє на альтернативу. Так реалізується концепція повернення програми на декілька кроків назад до вибору альтернативного ходу розв'язання задачі. Для реалізації операторів "***Suppose_deleting***" та "***Suppose_appending***" в кожен предикат додається байт ознак, який вказує, чи видалене у цей момент це значення. Нульове значення старшого біту означає, що число видалене, але щоб повернути його назад, достатньо цей біт встановити в одиницю. Окрім цього, адреса та попереднє значення цього байту вводиться в стек стану програми. Отже, при поверненні програми в точку альтернативи відновиться і стан даних. З іншими бітами байту ознак можна зв'язати ту чи іншу інтерпретацію дій програміста, оскільки інтерпретація повинна мати не лише назву, але і певні дані чи дії.

Методика програмування

Методика програмування мавою Open PROLOG майже не відрізняється від методики програмування класичною мовою PROLOG. Кожен предикат виражає певне логічне поняття. Перша відмінність полягає у тому, що для назв предикатів можна використовувати не лише англійські літери, а довільні, окрім спецсимволів. Кожен предикат поступово уточнює інформацію, яка до нього надходить, тобто, змінні, які були вільними до виконання предикату, стають зв'язаними після його виконання.

Зауважимо ще одну відмінність. При врахуванні порядку фактів у предикаті майже відпадає необхідність користуватися списками. Наприклад, додавання елементу до списку можна розглядати як операцію "*Suppose_appending*", а списком вважати набір фактів у предикаті. Отже, якщо додавання елементу до списку привело до хибного результату (*Fail*), то операція додавання буде звичайним чином скасована.

Необхідно також зауважити, що алгоритмічно зручніше працювати з послідовністю фактів, ніж зі списком. Тому для компактнішого вигляду пропонується представляти набір фактів з одного елементу як послідовність у квадратних дужках, як показано в наступному прикладі.

Приклад програми

Для демонстрації можливостей Open PROLOG наведемо приклад програми, що розв'язує задачу з туристами. Формулювання обчислювального завдання фактично зводиться до опису задачі:

Clauses
Дорослий [Хазяїн, Маша, Павло].
Лівий_берег [Маша, Сашко, Сергій, Павло, Дарія, Діана, Бультер'єр, Хазяїн].
Правий_берег [ ].

Перевезення :– *Suppose_deleting* (Лівий_берег (X)),
Suppose_deleting (Лівий_берег (Y)),
Дорослий (X), Лівий_берег [ ].
Перевезення :–
Suppose_deleting (Лівий_берег (X)), *Suppose_deleting* (Лівий_берег (Y)),
Дорослий (X), *not* (Заборона (Лівий_берег)),
not (Заборона_на_поромі), *Suppose_appending* (Правий_берег (X)),
Suppose_appending (Правий_берег (Y)),
Suppose_deleting (Правий_берег (Z)),
Suppose_appending (Лівий _берег (Z)),
not (Заборона (Правий_берег)).
Заборона_на_поромі (Маша, Сашко).
Заборона_на_поромі (Маша, Сергій).
Заборона_на_поромі (Павло, Дарія).
Заборона_на_поромі (Павло, Діана).
Заборона_на_поромі (Бультер'єр, X) :– X = not (Хазяїн).

Заборона (*P*) :– *P* (Маша), *P* (Сашко), *not* (*P* (Павло)).
Заборона (*P*) :– *P* (Маша), *P* (Сергій), *not* (*P* (Павло)).
Заборона (*P*) :– *P* (Павло), *P* (Дарія), *not* (*P* (Маша)).
Заборона (*P*) :– *P* (Павло), *P* (Діана), *not* (*P* (Маша)).
Заборона (*P*) :– *P* (Бультер'єр), *not* (*P* (Хазяїн)).

Algorithm
LOOP(J,1,999), Перевезення, fail.
Goal
Лівий_берег [ ].

Для наближення концепції предикату і списку дозволяються порожні предикати, тобто такі, які не містять жодного факту або правила. На відміну від старої версії спрощена система оголошень предикатів та введення макрокоманд, які представлені у прикладі, замість змінної *P* можна підставляти назву предикату.

Висновок

Отже, нами була висунута концепція покращеної мови PROLOG та реалізоване програмне ядро для її перевірки. Наскільки буде зручною нова парадигма припущень "*suppose*" для фактів, може визначити лише час, а також користувачі.

1. Братко И. Программирование на языке Пролог для искусственного интеллекта: Пер. с англ. – М.: Мир, 1990. – 560 с. 2. Стерлинг Л., Шапиро Э. Искусство программирования на языке ПРОЛОГ: Пер. с англ. – М.: Мир, 1990. – 333 с.