МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Кафедра обчислювальної техніки

КОНСПЕКТ ЛЕКЦІЙ

з програмного модулю
"Логічне програмування"

Розробник: асистент Алещенко Олексій Вадимович
(посада, П.І.Б.)

Затверджено на засіданні кафедри
Протокол № 11 від 24 травня 2017 р.

Завідувач кафедри ОТ
_____ Стіренко С.Г.
(підпис) (прізвище, ініціали)

Київ – 2017

Вступ

Логічне програмування — парадигма програмування, а також розділ дискретної математики, що вивчає методи і можливості цієї парадигми, засновані на виведенні нових фактів з даних фактів згідно із заданими логічними правилами [1]. Логічне програмування засноване на теорії математичної логіки. Найвідомішою мовою логічного програмування є Prolog, що є за своєю суттю універсальною машиною виводу, що працює в припущенні замкнутості системи фактів.

Першою мовою логічного програмування була мова Planner, в якій була закладена можливість автоматичного виведення результату з даних і заданих правил перебору варіантів (сукупність яких називалася планом). Planner використовувався для того, щоб знизити вимоги до обчислювальних ресурсів (за допомогою методу backtracking) і забезпечити можливість виведення фактів, без активного використання стека. Потім була розроблена мова Prolog, яка не вимагала плану перебору варіантів і була, в цьому сенсі, спрощенням мови Planner.

Від мови Planner також відбулися логічні мови програмування QA-4, Popler, Conniver, і QLISP. Мови програмування Mercury, Visual Prolog, Oz і Fril будувалися вже від мови Prolog. На базі мови Planner було розроблене також декілька альтернативних мов логічного програмування, не заснованих на методі backtracking, наприклад, Ether (див. огляд Шапіро [1989]).

Принципи організації та експлуатації мови Prolog

Представлення речень, фактів і запитів в мові Prolog основане на процедурній інтерпретації логічних речень типа фраз Хорна, які представляються в вигляді: "Логічний висновок A буде істинним, якщо умови B[1] і B[2] і ... і B[n] є істинними", запропонованої в 70-і роки XX-го сторіччя Ковальським [2]. Це речення може розглядатися як процедура рекурсивної мови програмування, при інтерпретації якої спочатку

перевіряється доцільність контролю умов за характером цільового твердження, а потім перевіряються умови, що відповідає реалізації зворотної цепи логічного доведення. Як фактичний стандарт синтаксису сучасної мови Prolog, в більшості сучасних Prolog-систем використана Единбурзька версія мови з узагальненою формою представлення речень Хорна.

A :- B[1], B[2],..., B[n].

де n>0, A – заголовок, що визначає форму логічного висновку, яка задається складеним термом або структурою, а послідовність B[i] – список умов, які задаються складеними термами та атомами і складають тіло речення.

Заголовки та елементи тіла розглядаються як логічні функції (предикати), результати яких приймають одне з двох можливих значень true ("істина" або "успіх доведення") та fail ("невдача доведення"), що являють собою базові стандартні предикати-атоми. Найпростіший елемент речення мови Prolog, має варіативний логічний сенс, записується в формі структури або складеного терму, що включає покажчик (ім’я функціонального зв’язку), який називають функтором, та список аргументів, включений в круглі дужки, наприклад, author(smith,X,Y).

Речення мови Prolog складають інформаційну базу (ІБ) або базу знань (БЗ) ІС, яка записується у вбудовану базу даних (БД) Prolog-системи. Речення з однойменними функторами заголовка і однаковою колкістю аргументів (арністю) являють собою диз’юнкцію правил доведення того самого висновку та об’єднуються в Prolog-процедури з одним покажчиком функції. Для правильної роботи програми необхідно забезпечити позиційну відповідність синтаксису і семантики аргументів. Саме контроль такої відповідності являє основу коректної побудови моделі області знань (ОЗ) при абстрагуванні.

Речення мови Prolog, які включають в тілі єдину, стандартну і завжди істинну ціль true називають фактами. Скорочена форма записи факту включає тільки головну частину, яка завершується точкою. Факти

відповідають стверджувальним реченням або твердженням природної мови. Універсальні факти включають серед аргументів імена змінних. Традиційна інтерпретація фактів логічних програм на природній мові має вигляд "В ОЗ, що розглядається існує властивість (зв'язок), поіменована функтором, яка зв’язує об’єкти ОЗ або їх характеристики, задані аргументами". Скінченна множина фактів являє собою найпростішу програму на мові Prolog. Наприклад, відношення авторства програм може бути задано фактами процедури author:

```
author(smith, sqrt, mlib).
author(mouse, _, nwlib).
```

Ці факти можна інтерпретувати так: smith є автором підпрограми sqrt з бібліотеки mlib, а mouse – всіх підпрограм з nwlib. Звідси слідує, що правильність програм можна забезпечити строгим додержанням позиційної відповідності аргументів структури.

Роздільники елементів лівої частини речення визначають послідовність перевірки умов, заданих предикатами. Знак "," задає кон’юнкцію умов (операцію "І"), що визначає їх послідовну перевірку. Перевірка кожної з умов в Prolog-системі розглядається як доведення або доведення окремої цілі. Узагальнення речень Хорна включенням диз’юнкції умов (операції "АБО"), яка задається знаком ";" з врахуванням більш високого пріоритету кон’юнкцій призводить до того, що істинність висновку вважається доведеною тоді, коли досягнуто всі цілі принаймні в одній з груп кон’юнкцій, що входять до загальної диз’юнкції речення. Узагальнені речення Хорна є основою представлення знань в Prolog-системах.

1-е правило логічного доведення в мові Prolog визначає послідовну обробку цілей узагальнених речень Хорна, виконується інтерпретатором або ядром бібліотеки мови Prolog, яку називають скануванням.

2-е правило логічного доведення за Prolog-програмою або правило співставлення: "Тільки можливість співставлення або уніфікації цільового

предикатного виклику з заголовком відповідного речення Prolog-процедури призводить до можливості успішного досягнення цілі речення процедури і необхідності обробки цього речення".

3-е правило: послідовний перегляд речень процедури у випадку невдачі з попереднім реченням та вихід з результатом fail при закінченні фраз Prolog-процедури.

4-е правило: зворотний перегляд цілей тіла речення (backtracking) при одержанні fail для виклику чергової цілі, а при досягненні заголовку перехід на наступне речення Prolog-процедури.

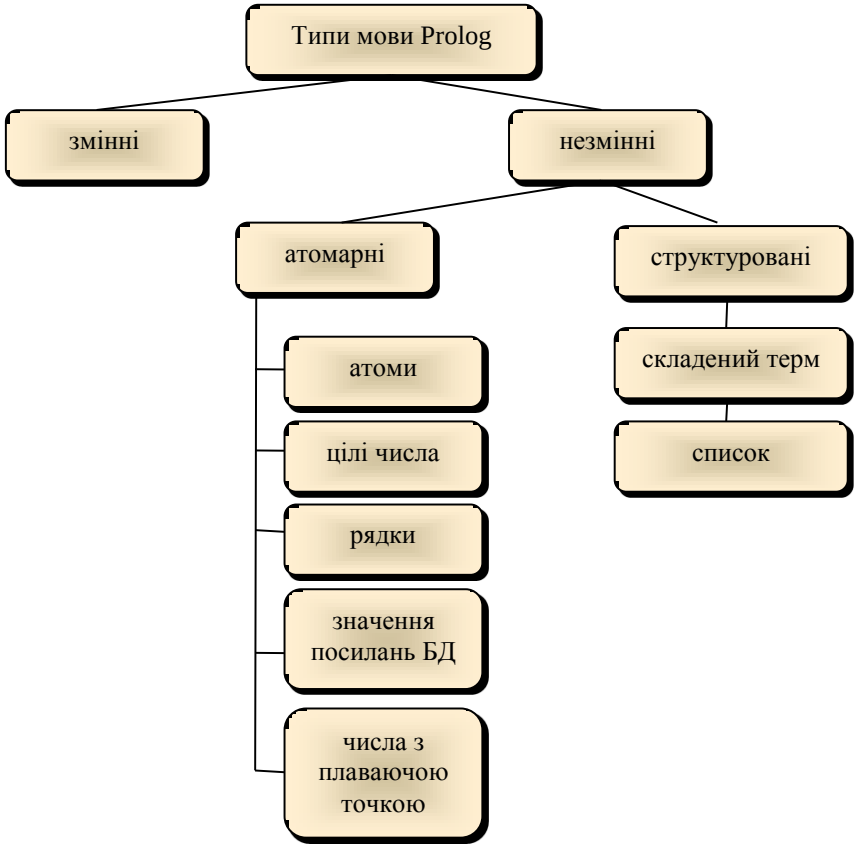
В основу діалогової оболонки мови Prolog покладені засоби введення запитань та запитів на розв’язання задач або одержання інформації з Prolog-системи, включаючи системні предикати і предикати користувача, занесені в БЗ. Просте запитання включає одну ціль, визначену умовою задачі, і призначене для з’ясування зв’язків між об’єктами системи, що визначається поодинокими викликами.

Запитання, сформульоване таким чином, відповідає запитанню на природної мові виду: "Чи є справедливим при поточному стані БЗ Prolog-системи висновок про істинність зв’язку або відношення заданого виду С". Складне запитання можна представити подібно тілу речення Хорна як послідовність кон’юнктивних або диз’юнктивних цілей (умов), в якому підсумкова відповідь відповідає успіху кон’юнкції або диз’юнкції окремих цілей. Запитання записується в режимі діалогу після знака "?-" і завершується точкою. Прості запитання включають поодиноку ціль, а складні – список цілей, що розділені комами. Відповідь на запитання дається або в формі видачі варіантів значень підстановок змінних, указаних в запитанні, або "yes"/"no" для екзистенційних запитань, в яких відсутні імена змінних в аргументах. Це відповідає реченням природної мови "Ціль G виводиться з програми P, яка розміщена в БЗ Prolog-системи" або "Предикату G відповідає істинний факт F з програми P".

Спростити визначення складного запитання можна шляхом включення в БЗ Prolog-системи правило, яке об’єднує на загальному рівні цілі складного запитання. При побудові запитань імена змінних повинні задавати об’єкти, для яких потрібні варіанти рішень. Аргументи в предикатах необхідно задавати у відповідності з вимогою жорсткої позиційної відповідності семантики аргументів в ІБ та в запитаннях.

Узагальнення запитань і фактів зводиться до того, що запитання задає первинну ціль в ланцюжку міркувань, а відповідність факту запитанню виявляється на ділянці, яка завершує логічні міркування.

Всі елементи речень мови Prolog представляються у вигляді складових частин, що включають типи з наступної ієрархії.



Змінні

Змінні можуть уніфікуватися з даними будь-якого іншого типа і з іншими змінними. Змінні, уніфіковані тільки з іншими змінними, називають незв’язаними. Коли визначається значення однієї змінної (вона зв’язується зі значенням іншого об’єкта і надалі співпадає з цим об’єктом), то

визначаються значення і всіх уніфікованих з нею змінних. Область дії змінної обмежена оброблюваним реченням. Передача значень змінних іншим реченням забезпечується уніфікацією при звертанні до цільової процедури. Назви або імена змінних задаються послідовністю алфавітно-цифрових символів, що починається з великої букви або підкреслення "_" або може бути поодиноким підкреслюванням, представляючи змінну без імені або анонімну змінну, що використовується, коли вам не цікаве значення, яке повертається предикатом або передається як фактичний параметр.

Атоми

Атоми – це текстові дані, які задають іменовані або фіксовані об’єкти програми. Вони можуть включати в собі букви, цифри та інші символи. Якщо вони починаються з великої букви або з цифри, або коли вони включають в собі роздільники та інші символи, то вони повинні бути замкнені в апострофи. Апостроф у представленні атома дублюється. Приклади атомів:
tennis '23' 'Ukraine' 'Kyiv' '-->' 'Oxana' 'S'

Атоми завжди уніфікуються тільки з атомами, які співпадають за записом, виключаючи зовнішні апострофи і можуть використовуватися як функтори в складених термах. Гранична довжина атомів залежить від версії та реалізації мови Prolog і в AMZI!Prolog обмежена 255 знаками. В процесі виконання атоми відображуються так само, як і у вхідному тексті на мові Prolog.

Цілі числа

Цілі числа – це позитивні або від’ємні цілі числа в числовому проміжку від (-2,147,483,648) до (+2,147,483,647), представлені 32 двійковими розрядами. Приклади цілих чисел:
3 299 234567 -889 0

Цілі числа завжди уніфікуються з рівними цілими числами. Символи коду ASCII у внутрішньому поданні перетворюються в цілі числа і для

числового представлення знаків ASCII використовується попередній знак зворотного апострофа "`".

Наприклад, наступні 2 елементи рівні: `a і 97.

Числа з плаваючою точкою

Десяткові числа з плаваючою точкою надають можливості для обчислень над числами в широкому діапазоні значень. Ці числа мають точність до 15 десяткових знаків після точки і представляються в традиційному форматі з обмеженим представленням десяткового порядку інтервалом від -308 до 308. Приклади значень дробових чисел:

0.5 55.3 -83.0E21 2134.2 122.345e25

Наступні терми не є числами з плаваючою точкою:

5 -17.23E-1000

Числа з плаваючою точкою уніфікуються з рівними за значенням числами цього же типа, однак два близьких числа можуть не уніфікуватись через неспівпадіння молодших розрядів.

Рядки

Рядки – текстові константи, які використовуються для ефективних маніпуляцій з текстом. Вони записуються як знаки коду ASCII, включені в знаки "$". Наприклад, рядок:

$Він програмує
задачу прийняття рішення $

включає символ переведення рядка. Порожні рядки записуються так: $$. Для включення знака долара в рядок його треба указати двічі. Наприклад:

$ Студент заплатив $$5 за флешки.$

Рядки уніфікуються з рядками, що мають ідентичні тексти. Рядки ніколи не уніфікуються з атомами, навіть якщо у них ідентичні тексти. Розміри рядків залежать від версії та реалізації мови Prolog і в AMZI!Prolog рядки обмежуються 4 Кбайтами.

Структури або складені терми

Пролог передбачає тільки один складний тип даних, який називають структурою або складеним термом. Будь-який об’єкт задачі, який формалізується, як і будь-яке відношення між об’єктами, подається термом того чи іншого вигляду. Наприклад, дату можна представити наступним термом.

date(friday, 21, february, 1997)

Це приклад структурованого терму, який звичайно називають просто структурою. Структура складається з покажчика функції (функтора), який являє собою ім’я відношення, і послідовності компонентів, що являють собою об’єкти або характеристики відношення. Число компонентів (аргументів) структури в мові Prolog часто називають арністю (а деякі перекладачі) розмірністю структури. В даному прикладі: date – функтом (покажчик функції) структури, friday, 8, february, 1997 – компоненти структури; арність структури – чотири. Синтаксис мови Prolog потребує наступного: аргументи включаються в дужки і розділяються комою ","; між функтором і відкриваючою дужкою, майже в усіх реалізаціях не припускається пропуск.

Структури – це типи даних загального призначення, які можуть використовуватися для групування або формування зв’язку між об’єктами. Структура складається з покажчика об’єкта (функтора) і його аргументів. Наступний запис є прикладом структури мови Prolog.

'програма'(plants, publisher(green_house))

Структури можна уніфікувати з іншими структурами. Вони успішно уніфікуються, якщо функтори структур співпадають і всі відповідні аргументи уніфікуються. Наприклад, наступні структури уніфікуються.

'програма'(Author, Title)

'програма'(james,$ Запис в B-дерево$)

Наступні структури не уніфікуються, тому що атом charles не уніфікується з атомом chuck.

name(jones,charles) і name(jones,chuck)

Структури подаються за канонами префіксного запису, тобто функтор записується першим, у супроводженні його аргументів. Важливо розуміти, що імена, які обираються для функтора і компонентів структури довільні, і їх бажано обирати у відповідності зі змістом задачі, причому бажано, щоб вони не пересікались з іменами понять (стандартних функцій) мови. Коли структура використовується для представлення відношення, необхідно установити, як ця структура повинна інтерпретуватися, і узгодити в межах програми інтерпретацію всіх структур, що мають той самий функтор і таку ж розмірність. В даному прикладі інтерпретація структури наступна: "Структура являє собою дату. Чотири її компоненти представляють день тижня, який відповідає цій даті, число, місяць і рік."

Програмісту рекомендують використовувати такі імена для функтора і компонентів, які підказували б людині, що читає програму, як інтерпретується дана структура. В наведеному прикладі перший і третій компоненти структури є атомами. З цих позицій атом можна розглядати як спеціальний вид терму без аргументів. Функтор має той же синтаксис, що і атом.

Переклад на російську мову англійських слів і виразів, який часто використовується як мнемонічні імена об’єктів і відношень, потребує додавання до кодів букв латинського алфавіту кодів кирилиці – маленьких і великих букв алфавітів більшості слов’янських мов. В існуючих реалізаціях мови Prolog на комп’ютерах з можливістю роботи як з латинським алфавітом, так і з кирилицею (великими і маленькими буквами того чи іншого алфавіту), на жаль, не можна будувати і використовувати кириличні атоми без апострофів. Наприклад, якщо jazz ensemble "повноцінний" атом, то джаз ансамбль (в реалізації!) не є таким.

Представлення зв’язків та відношень в мові Prolog

Наведемо компоненти Пролог-програм і продемонструємо використання цих компонентів в маленькій, але повноцінній програмі, яка є базою даних про людей. Покажемо, як видобувається інформація з цієї БД. Подібна інформація може бути представлена багатьма різними способами: тут ілюструється, як вдало обране подання полегшує інтерпретацію видобутої інформації.

Прості відношення в Prolog-програмах представляються фактами, в яких зв'язок іменується функтором. При цьому з точки зору синтаксису мови Prolog послідовність визначення аргументів в процедурі немає значення, подібно аргументам функцій і процедур в інших мовах програмування. Тому користувач має право для скорочення різноманіття форм обрати стандартні обмеження на впорядкування списку аргументів, а також виділити для приймача результатів наперед заданий аргумент, найчастіше останній. Для іменування відношень зручно задавати як функтор ім’я зв’язку, що дозволяє представляти двомісні оператори в інфіксному запису без дужок, близької за структурою до речень природної мови.

При проектуванні ІБ ІС бажано запобігти дублювання інформації, але при практичному застосуванні в ІС зручно виходити зі сприйняття покажчика функції (функтора) як імені зв’язку або покажчика підлеглості в ієрархічній структурі. Вибір оптимальних структур фактичної інформації залежить від розв’язуваних задач, що визначаються загальною областю досліджень (ОД) і підобластями. Підобласті групуються за правилами, що визначають розв’язання своїх задач і серед фактичної інформації повинні включати достатньо даних для розв’язання задач. Тому фактичну інформацію, що об’єднує декілька процедур доцільно накопичити в єдиному файлі фактів ОД.

Як уже згадувалось для представлення відношень можна користуватися наборами фактів, які представляють відношення реляційної алгебри "один-до-одного" або "один-до-багатьох", поіменованими функторами структур, що

включають фіксовану кількість полів з ключовими даними та даними, що видобуваються з таблиці. Способи роботи з таблицями відомі з курсу "Інформаційні системи і бази даних".

Для полегшення заповнення БД Prolog-системи користувачу зручно працювати зі спеціальними шаблонами, що визначають синтаксис и семантику аргументів предикатів. В тексті еталонної програми ці шаблони наведені в формі текстових мета позначень.

Предикати аналізу типів

Стандартні (системні) предикати або процедури мови Prolog описуються в форматі метапозначень, які показують правила його використання. Символ, що передує кожному аргументу, умовно показує спосіб його обробки при використанні предиката.

Знак плюс "+" перед аргументом показує, що він використовується як вхідний аргумент, для якого перед виконанням предиката повинно задаватися визначене значення або посилання на змінну. Знак мінус "-" перед аргументом показує, що він використовується як вихідний аргумент і при виході приймає значення, одержане в результаті обробки. Однак в деяких випадках повернуте значення може не змінитися. Знак питання "?" перед аргументом показує, що аргумент можна використати як вхідний, так і вихідний, залежно від контексту, в якому він використовується. Опис предиката наводить результат обчислення для обох типів аргументів.

Стандартні предикати, застосовуються для перевірки програмістом значень даних користувача для уникнення одержання неправильних результатів.

var(-X) – true, якщо X є змінною;

nonvar(+X) – true, якщо X не є змінною;

atom(+X) – true, якщо X є атомом;

integer(+X) – true, якщо X є цілою константою;

atomic(+X) – true, якщо X - або число, або атом;

float(+X) – true, якщо X - число з плаваючою точкою;

number(+X) - true, якщо X - числові дані (цілі або з плаваючою точкою).

string(+X) - true, коли X є рядком символів.

ref(+X) – true, коли X є номером посилання БД.

Предикати числових відношень

Арифметичні відношення викликають арифметичне обчислення значень аргументів E1 і E2, представлених числовими арифметичними виразами.

E1>E2 – true, якщо значення E1 більше, ніж E2.

E1<E2 – true, якщо значення E1 менше, ніж E2.

E1>=E2 – true, якщо значення E1 більше або дорівнює E2.

E1=<E2 – true, якщо значення E1 менше або дорівнює E2.

E1=:=E2 – true, якщо значення E1 дорівнює E2.

E1=\=E2 – true, якщо значення E1 не дорівнює E2.

Використання Eclipse IDE

Amzi!IDE – розширення проекту IDE Eclipse (www.eclipse.org) з відкритими текстами вхідних кодів. Це – застосування Java, яке забезпечує дружній інтерфейс GUI Amzi! Prolog (та багатьох інших мов).

Eclipse підтримує багатомовне оточуюче середовище розробки. Їх називають 'Перспективами'. Можна відкрити перспективу командою Вікно | команда Open Perspective, або натискаючи кнопки в вертикальній смузі на левом краю. Кнопка Paw відображує перспективу мови Prolog. Кнопка Bug відображує перспективу Налагодження.

Правила і агрегати в Prolog-системах

Мову Prolog побудовано таким чином, щоб гранично спростити представлення рекурсивних зв’язків і забезпечити їх декларативне подання. Тому в структурах мови Prolog відсутні агрегати з перенумерованими елементами (масиви) і всі дії з побудови або аналізу агрегатів реалізують за допомогою спеціальних структур, які називають списками.

Представлення та особливості обробки списків в мові Prolog

Списки є базовим засобом агрегатування однорідних даних мови Prolog. Список – це послідовність елементів, які розміщуються в заданій послідовності. Будь-який об'єкт мови Prolog, наприклад, змінні, структури або інші списки може бути елементом списку. В базовій формі запису списку, яка легко читається, елементи відділяються один від одного комою, а їх послідовність включається в квадратні дужки. Наприклад, список атомів a, b і c представляється в цій формі як

[a, b, c]

Внутрішнє представлення списку мови Prolog реалізовано у вигляді двомісної структури з функтором '.' і двома аргументами голови і хвоста. Головою в списку є перший елемент списку. Хвіст являє собою список, що включає всі інші елементи списку. Порожнім називається список, який не включає елементів і представляється замкненими квадратними дужками ([]). Таким чином, список с одним елементом a в базовій формі можна записати як [a], а у вигляді структури – '.'(a,[]). Список, що складається з атомів a, b і c може бути записаний як

'.'(a, '.'(b, '.'(c,[])))

В базовій формі для розділення голови і хвоста списку використовується вертикальна риска і в такій формі список [a, b, c, d] представляється як

[a|[b, c, d]]

Списки допускають різноманітні представлення комбінованих форму у вигляді послідовності елементів з залишковими списками. Так список, заданий в формі [a, b|[c]], є еквівалентним списку [a|[b, c]], а таким чином – і спискам [a, b, c], [a, b, c|[]] та [a|[b|[c]]].

Якщо залишковий список або хвіст задано незв'язаною змінною, наприклад [a, b, c|X], то весь список називається частково визначеним або недовизначеним і змінна X, задана в хвості списку, може використовуватися

для приєднання нового списку як хвоста, який також може включати хвостову незв’язану змінну, до голови списку будь-якої конфігурації.

Списки цифр записуються в особливій формі, як список цілих чисел, що відповідають кодам ASCII послідовності текстових знаків. Списки літер включаються в подвійні лапки. Наступні три списки є еквівалентними:

"abc" [97,98,99] '.'(97,'.'(98,'.'(99,[])))

З еквівалентності списку і спеціальної структури випливає, що списки уніфікуються з іншими списками, якщо уніфікуються їх частини. Насамперед перевіряється чи є обидва уніфікованих об’єкта списками, а потім – попарно перевіряється можливість уніфікації голів і хвостів. В граничних окремих випадках порожній список уніфікується тільки з порожнім списком, а одноелементний список – зі списком з порожнім хвостом.

Складання програм обробки списків

Для завершення рекурсивних процедур мови Prolog з позитивним результатом true, який надає можливості формування і передачі результатів у вхідних змінних, необхідно використовувати спеціальні правила або факти, що забезпечують вихід з рекурсивної процедури. Для оновлення значень змінних через їх локальну природу в мовіProlog в рекурсивних процедурах в число аргументів повинні включатися змінні посередники або накопичувачі, в яких при обробці зберігаються проміжні результати. Програми сортування використовують базові предикати порівняння, властивості уніфікації списків для їх формування і декомпозиції, а також спеціальні предикати об’єднання двох списків, наприклад, предикат, заданий процедурою :

```
append([],L,L).
append([H|T],L,[H|T1]):-append(T,L,T1).
```

Ця процедура приєднує елементи списку, заданого в другому аргументі, в кінець списку, заданого в першому аргументі і повертає об’єднаний список в третьому аргументі, який при рекурсивних звертаннях розглядається як

посередник. Тому що формування списку починається з приформування головного елемента до хвоста списку, який за своєю суттю є накопичувачем, і не потребує інших накопичувачів. З іншого боку ця процедура може розглядатися як запит на виведення всіх варіантів голови і хвоста списку для заданого об'єднаного списку.

Управління програмами рекурсивної обробки

Щоб програма досягла заданої мети з однозначними результатами, вона потребує спеціальних засобів управління. Звичайні мови програмування використовують подібні оператори для побудови циклів і розгалужень.

В мові Prolog предикати управління змінюють спосіб, яким програма буде виконуватися за допомогою управління операторами вашої програми і цілями в кожному операторі. Prolog-система намагається закінчити розв'язання підцілі перед тим, як вона переходить на іншу ціль. Це називають depth-tivstsearch (пошук першої глибини). Для управління програмою використовуються предикати з наступного списку.

! - *відсічення* (*cut*) є завжди успішною ціллю, але при зворотному перегляді вона приводить до виходу з поточної процедури з результатом fail без аналізу альтернативних варіантів пошуку рішення, передбачених в процедурі. Відсічення не повинні використовуватися в межах управляючих предикатів ifthen, ifthenelse або структури управління case.

case(+[A_1 -> B_1, A_2 -> B_2,...|C]) обробляє першу ціль з B_i, якщо досягнута A_i. При невдачі всіх B_i обробляється ціль C, якщо вона опущена – case повертає true.

ifthen(+P,+Q) обробляє ціль Q, якщо досягнута P, в іншому випадку повертає true без перевірки цілі Q.

ifthenelse(+P,+Q,+R) обробляє ціль Q, якщо досягнута P, а в іншому випадку – ціль R.

[!P!] - швидкий пропуск цілей *P* при зворотному перегляді, коли цілі, включені в дужки пропускаються.

call(+P), де терм *P* інтерпретується як ціль, і результат call(P) співпадає з результатом P.

not(+P) або \+P або not P, де терм P являє собою ціль. Якщо P досягає успіху, not P не досягає успіху, а якщо P не досягає успіху, not P досягає успіху.

Описи Prolog-процедур шаблонів

В еталонній програмі наведені приклади Prolog-процедур для розв'язання задач перевірки належності об'єкта мови Prolog до контрольованого списку (member); злиття двох списків, впорядкованих за зростанням числових значень (merge); визначення довжини списку (lenlst); а також процедура швидкого сортування (qsort). В прикладі використані допоміжні процедури приєднання елементів списку в кінець базового списку (append) і процедури розбиття списку на частини, упорядковані за потрібним відношенням порядку.

В програмах, зв'язаних з визначенням арифметичним відношенням порядку, використовуються операції арифметичних відношень, наведені в роботі 1. Однак якщо потрібно використати відношення порядку для будь-яких, в тому числі нечислових об'єктів мови, то перед знаком відношення порядку об'єктів треба записати літеру «@»

O1@>O2 – true, якщо об'єкт O2 передує об'єкту O1.

O1@<O2 – true, якщо об'єкт O1передує об'єкту O2.

O1@>=O2 – true, якщо об'єкт O1не передує об'єкту O2.

O1@=<O2 – true, якщо об'єкт O2 не передує об'єкту O1.

Для порівняння різноманітних об'єктів в Prolog-системах визначена шкала відношення порядку з наступною послідовністю зростання ваги об'єктів:

- змінні в порядку зростання внутрішніх номерів;

- числові об’єкти в порядку зростання значень;
- рядки в алфавітному (лексикографічному) порядку (в кодах ASCII);
- атоми в алфавітному (лексикографічному) порядку (в кодах ASCII);
- номера посилань на БД, в довільному порядку внутрішнього подання;
- складні терми або структури впорядковуються спочатку за арністю, потім за іменем головного функтора і потім рекурсивно за вкладеними аргументами; списки розглядаються як бінарні структури (з арністю 2).

Предикати порівняння об’єктів не обчислюють вирази і не уніфікують значення, а таким чином не змінюють величину їх аргументів. Вони порівнюють символьні образи або внутрішні коди термів.

Визначення предикатів порівняння об’єктів представлені нижче, де O1 і O2 – це позначення об’єктів, які є аргументами предикатів.

O1==O2 – визначає, чи є O1 і O2 еквівалентними.

O1\==O2 – визначає, чи є O1 і O2 нееквівалентними.

compare(-Comp,+O1,+O2) – порівняння елементів O1 і O2 з видачею знака відношення у вигляді = , якщо O1 дорівнює O2; < , якщо O1 менше O2; > , якщо O1 більше O2.

sort(+L1,-L2) – упорядковує список L1 у відповідності зі стандартним відношенням порядку об’єктів, і після виключення дублікатів видає результат в L2. Наприклад:

?-sort([q, a, f, a, q, f],L).

L=[a, f, q]

keysort(+L1,-L2) – впорядковує список термів виду Key-Value (Ключ-Значення), зі збереженням дублікатів і повертає його в L2. Наприклад:

?-keysort([a-3,b-2,a-1,c-5],L).

L=[a-3,a-1,b-2,c-5]

eq(?X,?Y) – визначає, чи є X і Y такими об’єктами, що займають одну область в сховищі БЗ.

Ціль eq успішно досягається лише в тому випадку, якщо аргументи є елементами, які займають те саме місце сховища або якщо обидва аргументи представлені рівними атомами. Наприклад: пара цілей X=male(a), eq(X,X) і ціль male(a)==male(a) позвертає true, а eq(male(a), male(a)) – fail. Незв'язані змінні в аргументах приведуть до успіху eq, якщо вони уніфіковані з однією змінною. Така перевірка суворої рівності корисна, коли необхідно знайти границі структури невизначених даних.

Практично в усіх процедурах, побудованих за рекурсивним принципом, використані спеціалізовані універсальні факти, що забезпечують успішний вихід з процедури. Для скорочення часу виконання програм за рахунок скорочення процесу зворотного перегляду після перевірки взаємовиключних умов необхідно використовувати предикати відсічення.

Порядок виконання рекурсивних програм обробки списків

Порядок виконання рекурсивних викликів процедур визначається правилом рекурсивного виклику. В наведеному нижче узагальненому рекурсивному правилі предикатний виклик f0(X,...) виконує обробку елементів списку, починаючи з головного, а предикатний виклик f1(X,...) – починаючи з кінцевого елемента списку.

fr([X|Xs],...):-f0(X,...),fr(Xs,...),f1(X,...).

Таким чином, щоб змінити послідовність функціональної обробки елементів, включаючи введення-виведення даних, достатньо перенести виклик відповідних функцій ліворуч (перед) або праворуч (після) від рекурсивного виклику. Умова виходу з рекурсії найчастіше задається фактом з порожнім списком.

Робота в режимі налагодження

Написані програми, як правило, потребують налагодження. Налагодження здійснюється шляхом відстеження виконання програми за допомогою вбудованого налагоджувача, який видає свої повідомлення в

спеціальному вікні. При налагодженні можна спостерігати і повідомлення програми, і повідомлення налагоджувача.

SWI-Prolog пропонує 2 налагоджувачі. Перший – традиційний консольний трасувальник Prolog tracer, а другий – віконний налагоджувач, який дозволяє працювати на рівні вхідних кодів.

Розглянемо детальніше кожний з налагоджувачів.

Консольний трасувальник Prolog tracer

Ви можете трасувати ваші обчислення, виконавши команду:

?-trace.

Ця команда переведе Prolog в режим трасування, який показує кожний шаг обчислень. Вихід з режиму трасування виконується за допомогою команди **?-notrace**.

Приклад:

```
student_of(S,T):-follows(S,C),teaches(T,C).
follows(paul,computer_science).
follows(paul,expert_systems).
follows(maria,ai_techniques).
teaches(adrian,expert_systems).
teaches(peter,ai_techniques).
teaches(peter,computer_science).

?- trace.
Yes
[trace]  ?- student_of(S,peter).
   Call: (7) student_of(_G311, peter) ? creep
   Call: (8) follows(_G311, _L210) ? creep
   Exit: (8) follows(paul, computer_science) ? creep
   Call: (8) teaches(peter, computer_science) ? creep
   Exit: (8) teaches(peter, computer_science) ? creep
```

Exit: (7) student_of(paul, peter) ? creep

S = paul ;

Після кожного рядка виведення можна задати команду. Нижче наведено список припустимих команд налагоджувача:

+:	spy	-:	no spy
/c|e|r|f|u|a goal:	find	.:	repeat find
a:	abort	A:	alternatives
b:	break	c(ret, space):	creep
[depth] d:	depth	e:	exit
f:	fail	[ndepth] g:	goals (backtrace)
h (?):	help	i:	ignore
l:	leap	L:	listing
n:	no debug	p:	print
r:	retry	s:	skip
u:	up	w:	write
m:	exception details	C:	toggle show context

За допомогою клавіші **RETURN –** ми легко просуваємося за нашим обчисленням. **Call –** означає проходження вниз за вузлом SLD-дерева (В мові Prolog для кожного запиту з правил і фактів створюється так зване SLD-

дерево); відображується тільки перший літерал рішення. **EXIT** - означає проходження вгору за вузлом дерева. Число зліва показує глибину вузла в SLD-дереві, починаючи відлік з 7. Так, в прикладі сутність **teaches(peter, computer_ science)** викликається на 8 рівні (тобто на рівні 2 в SLD-дереві).

Після находження першого рішення виконується повернення до останньої точки вибору подальших дій, введенням крапку з комою. Наприклад:

Redo: (8) follows(_G311, _L210) ? creep

Exit: (8) follows(paul, expert_systems) ? creep

Call: (8) teaches(peter, expert_systems) ? creep

Fail: (8) teaches(peter, expert_systems) ? creep

Redo: (8) follows(_G311, _L210) ? creep

Exit: (8) follows(maria, ai_techniques) ? creep

Call: (8) teaches(peter, ai_techniques) ? creep

Exit: (8) teaches(peter, ai_techniques) ? creep

Exit: (7) student_of(maria, peter) ? creep

S = maria;

Redo означає пошук альтернативного рішення. Друге рішення для **follows(S,C)** веде до неуспішної (failure) гілки, тому що **teaches(peter, expert_systems)** не має рішень. Таким чином, знов виконується redo, після якого знаходиться друге рішення.

teaches(peter, ai_techniques) – не останній факт з **teaches** в нашій програмі. Тем не менше після повернення видно, що більше рішень нема.

Redo: (8) teaches(peter, ai_techniques) ? creep

Fail: (7) student_of(_G311, peter) ? creep

No

Числові розрахунки і аналітичні перетворення в Prolog-системах

Для визначення можливості використання традиційної інфіксної форми представлення математичних виразів в більшості Prolog-систем існує спеціальний засіб для визначення операторів (операцій), які допускають інфіксну форму у вигляді директиви визначення операторів :-op/3. Воно дозволяє перетворити в інфіксну форму будь-які двомісні і одномісні відношення, включаючи арифметичні операції.

Предикати, що виконують арифметичні обчислення

До цієї групи відносяться оператори арифметичних відношень, описані в роботі 1, а також: X is E1 – обчислює E1 і уніфікує значення результату обчислення з X.

Арифметичні дії задаються такими арифметичними виразами, як X+1, з такими обчислювальними предикатами, як предикат обчислення is, або один з предикатів арифметичного порівняння.

Арифметичний оператор являє собою структуру, в якій функтор задає виконувану арифметичну операцію. Аргументи повинні бути числовими величинами, які задають операнди функтора. Наприклад, розглянутий елементарний арифметичний вираз можна представити у вигляді функтора '+' з двома аргументами:

'+'(X,1).

Більш складні вирази зручно представляти в інфіксній формі з врахуванням пріоритетів операцій і використанням дужок.

Арифметичні оператори (операції), оператори (операції) відношення і оператор (операції) обчислення виразу is, які можна представити в інфіксній формі еквівалентні двоміснім структурам, в яких знаки арифметичних операцій включені в апострофи. Хоча вбудований синтаксичний аналізатор мови Prolog буде перетворювати арифметичні вирази до префіксного запису, як в вищенаведеному прикладі, Пролог дозволяє використовувати інфікс-нотацію.

Визначення оператора переводять арифметичні вирази з інфіксної нотації до префіксної нотації під час введення, і з префіксної нотації до інфіксної нотації під час виведення. Таким чином, можна записати вищезгаданий приклад, як:

X+1.

Оператори спрощують введення і виведення цих арифметичних виразів в систему. Семантична інтерпретація обчислювальних операторів запускається оператором is або операторами (операціями) арифметичних відношень, які ініціюють числову обробку в Prolog-системах.

Обчислювальні можливості мови Prolog основані на стандартних правилах представлення відношень за допомогою покажчиків функцій (функторів) та аргументів.

Система swi-prolog і деякі інші реалізації мові Prolog мають широкі можливості довизначення арифметичних операцій за рахунок використання вбудованої мови C і компільованої арифметики, однак можливості його використання обмежені використаною конфігурацією діалогової оболонки. В прикладних програмах існують арифметичні вирази, в яких тип всіх змінних визначений і не змінюється. Такі вирази можна обробити зі збільшеною швидкістю, що забезпечується компільованою арифметикою. Крім того існує можливість модульного включення окремих процедур і предикатів мови Prolog, складених на інших мовах програмування.

В систему swi-prolog включено ряд предикатів, для яких необхідно визначити числові значення вхідних змінних:

inc(+X,-Y) – нарощує X на 1 і повертає значення в Y.

dec(+X,-Y) – зменшує X і повертає значення в Y.

randomize(+Speed) – перебудовує генератор випадкових чисел.

Аргумент Speed – це ціле число, яке задається в програмі.

Арифметичні операції виконують числові перетворення. Предикат для виконання простої арифметики легко записується в мові Prolog. Предикат

evaluate, наведений нижче, одержує арифметичний вираз і відображує його разом з результатом.

```
evaluate(X):-
Ans is X, case([Ans=err->
write('Cannot evaluate the expression:')
    /write(X=Ans)]).
```

Семантичну інтерпретацію неарифметичних операторів вбудованими засобами мови Prolog реалізувати неможливо.

Арифметичні операторі і функції

Арифметичні (обчислювані) оператори, описані нижче, можуть появляться в арифметичних виразах. Операнди X і Y представляються числовими даними, обчислюваними виразами або одним з чисел, заданих наступними атомами: random – породжує значення довільного числа в інтервалі 0.. 1; pi – породжує значення 15 десяткових розрядів числа π.

Оператори арифметичних виразів:

X+Y – додавання;

X*Y – множення;

X-Y – віднімання;

X^Y – піднесення до ступеню;

X/Y – ділення з дійсним результатом.

X//Y – ділення цілих чисел з цілим результатом.

-X – унарний мінус (міняє знак X).

Порозрядні оператори для цілих чисел:

X/\Y – кон'юнкція (AND); X\/Y – диз'юнкція (OR);

\(X) – інверсія (NOT);

X<<Y – зсув X ліворуч на Y позицій;

X>>Y – зсув X праворуч на Y позицій;

[X] – обчислюється як X.

X mod Y – повертає залишок від ділення X на Y.

Функції:

abs(X) – абсолютне значення X; exp(X) – експонента;

acos(X) – арккосинус X; ln(X) – логарифм натуральний;

asin(X) – арксинус X; log(X) – логарифм десятковий;

atan(X) – арктангенс X; sin(X) – синус X;

cos(X) – косинус X; tan(X) – тангенс X;

sqrt(X) – квадратний корінь з X.

round(X,N) – X округляється до N десяткових цифр, де N – ціле число між 0 і 15.

Арифметичний вираз обчислюється тільки в тому випадку, коли всі аргументи мають числові значення. Дробові числа можна явно перетворювати в цілі числа системним предикатом integer(X), зворотне перетворення виконує предикат float(X). Результатом виконання арифметичних операторів з нечисловими аргументами є атом err. Будь-які арифметичні порівняння з err приводять до результату fail.

Принципи представлення вхідних даних і результатів розв'язання числових рівнянь і нерівностей

Вхідні дані для обчислень в Prolog-системі можуть бути представлені у вигляді виразів, що включають константи, функції та змінні. Для поліноміальних рівнянь це можуть бути списки пар (<ненульовий коефіцієнт >, <покажчик ступеня >), які називають нормальною форма [1], списки коефіцієнтів полінома, починаючи з коефіцієнта при нульовому ступеню аргументу або представлення поліномів в формі стандартних операторів використовуваної версії мови Prolog.

В термі norm_form([(1,2),(2,1),(3,0)]) аргумент, представлений у вигляді списку, відповідає нормальній формі полінома 1*X^2+2*X^1+3*X^0, представленого в традиційній алгебраїчній формі. В термі lst_form([3,2,1])

список, заданий в аргументі відповідає аргументам полінома (3*X+2)*X+1, представленого за схемою Горнера.

З прикладів видно, що в Prolog-системах більш читабельна традиційна форма представлення поліномів, хоча і має деяку надлишковість, вона ж більш зручна і для організації обчислень і перетворень.

Розв'язання рівнянь і нерівностей в найбільш загальному вигляді представляють в формі області допустимих значень. Традиції представлення відкритих і закритих інтервалів значень за допомогою круглих і квадратних дужок не дозволяють представляти їх в рамках синтаксису мови Prolog. Тому зручно використовувати спеціальну форму представлення таких інтервалів. Наприклад, для визначення входження граничної точки до інтервалу можна користуватися унарним знаком рівності, для визначення нескінченних значень – атом ' ', а для визначення періодично повторюваних інтервалів значень спеціальну структуру з визначенням границь, параметра кратності і виразів для границь повторюваних інтервалів.

Рекурсивні обчислення і перетворювання в Prolog-системі

Подібно рекурсивній обробці списків для організації обчислень за ітеративними і рекурсивними алгоритмам при їх побудові використовуються допоміжні аргументи, які називають накопичувачами. В представленій нижче еталонній програмі представлена нестандартна процедура sqrt/2 для обчислення квадратного кореня з застосуванням ітеративної формули Ньютона. Базова процедура має два аргументи: в першому розміщується аргумент функції, а в другому – результат обчислення функції. В допоміжній внутрішній процедурі sq/3 використовується змінна-накопичувач Y0.

Для перетворень списків в структури, які можна застосовувати як предикатні виклики, можна використовувати системний предикат прямого і зворотного перетворення структури S на список L: ?S = .. ?L. Напрямок перетворення визначається від аргументу, що має визначене значення відповідного типу до аргументу, що є незв'язаною змінною. Функтор

структури S стає головним елементом списку L, а аргументи структури S – наступними елементами списку, або навпаки зі збереження цієї ж відповідності. Більше того, при використанні в структурах функторів арифметичних операцій з відповідною кількістю аргументів можна одержати прості вирази для використання в предикатах арифметичних відношень та в предикаті is. Для обробки складних виразів доцільно використовувати вкладені списки та їх перетворення на структури.

Список результату будується зі структури використанням функтора як голови списку, а переліку аргументів – як хвоста. Якщо обидва аргументи зв’язані зі значеннями, предикат перевіряє чи будуть вони еквівалентними. Наприклад, наступний запит перетворює структуру в список:

```
?-tree(oak, seed(acorn))=..X.
X=[tree, oak, seed(acorn)].
```

Щоб перетворити список в структуру можна набрати:

```
?-X=..[tree, oak, seed(acorn)].
X=tree(oak, seed(acorn)).
.
```

Для аналітичних перетворень можна використовувати спеціальні процедури в поєднанні з уніфікацією. У зв’язку з тим, що списки в мові реалізовано через двомісні операції, а більшість операцій також є двомісними, то їх рекурсивні перетворення також виглядають аналогічно. Так для перетворення полінома формі списку коефіцієнтів типа lst_form можна скористатися простою рекурсивною процедурою вигляду:

```
lst_to_gor([K|Ks],X,E*X+K) :- lst_to_gor(Ks,X,E).
lst_to_gor([K|[]],_,K).
```

При звертанні до цієї процедури при уніфікації другого аргументу з атомом x в третьому аргументі можна одержати вираз, зручний для відображення Prolog-системою, а при уніфікації другого аргументу з іменем змінної в третьому – повертається вираз, який після уніфікації цієї змінної з

числовим значенням можна використовувати для обчислень в операторе is. Цей оператор буде нормально виконуватися тільки тоді, коли числами визначені всі коефіцієнти. Тому при перетвореннях з нормальної форми треба пропускати, опущені нульові коефіцієнти.

Для однозначного перетворення з форми списку до форми аналітичного виразу достатньо використати виклик предиката lst_to_gor з першим аргументом, що являє собою список коефіцієнтів, другим, що включає атом для відображення полінома або незв'язану змінну для можливих розрахунків полінома і третім, що є незв'язаною змінною-приймачем виразу полінома. Для зворотного перетворення з форми аналітичного виразу до форми списку також можна використати виклик предиката lst_to_gor, в якому на місці першого аргументу задається змінна-приймач списку, в другому аргумент полінома, а в третьому вираз полінома за схемою Горнера. Однак для однозначного відновлення полінома, що уникнути альтернативних варіантів зі складними коефіцієнтами, необхідно в кінці першого правила процедури lst_to_gor додати предикат відсічення:

lst_to_gor([K|Ks],X,E*X+K):-lst_to_gor(Ks,X,E),!.

Після формування аналітичних виразів для відображення може бути доцільним створити предикати «розморожування» атомарних позначень невідомих змінних шляхом їх перетворення на імена незв'язаних змінних мови Prolog. Для перетворення атомів у виразах за схемою Горнера на незв'язані змінні та надалі в числові дані доцільно використовувати спеціальну Prolog-процедуру:

unfreeze(G*Xa+K,Xa,X,G*X+K):- unfreeze(Ks,Xa,X,E),!.

unfreeze(K,_,_,K).

Тоді виклик unfreeze(Gd,Xa,X,Gc)сформує з відображуваного полінома за схемою Горнера з невідомою x обчислювальний поліном за схемою Горнера з невідомою змінною X, при уніфікації якої з числовим значенням предикатом is можна одержати числове значення полінома.

Подібним чином можна виконувати будь-які ізоморфні перетворення з одноманітними об’єктами. Для більш складних адекватних або еквівалентних перетворень механізм створення рекурсивної Prolog-процедури складається з побудови процедури, подібної до lst_to_gor, в якій кінцевий факт, що визначає вихід з рекурсії замінюється звертанням до таблиці елементарних перетворень, в яку можна включити базові факти або правило, що відповідає особливим законам перетворень. Таким чином, можна реалізувати табличне інтегрування і диференціювання, пряме і зворотне перетворення Лапласа, а також цілеспрямовані перетворення рівнянь і нерівностей з метою одержання їх рішень в формульному вигляді.

Інтерактивні оболонки для надійної експлуатації інформаційної бази цільової предметної області

Відсутність в Prolog-системах вбудованих засобів синтаксичного і семантичного контролю введених даних викликає необхідність розробки спеціальних засобів запитів з використанням вбудованих можливостей системи з введення-виведення.

Основні предикати введення-виведення термів виконують обмін атомами, змінними і твердженнями мови Prolog. Терми повинні відповідати стандартному формату, і коли вводяться користувачем, і коли видаються програмою. Введення терму закінчується точкою і натисненням клавіші введення, які не інтерпретуються як частина терму. Атоми, представлення яких включає пропуски, букви кирилиці або починаються з великої латинської букви, включаються в апострофи.

Предикат write(+Term) використовують, щоб записати терм в стандартному пристрої виведення. Будь-які незв’язані змінні представляються як знак підкреслення, за яким слідує шістнадцяткове число. Предикат write розпізнає синтаксис визначень оператора і використовує ці визначення при відображенні терму.

Предикат writeg(+Term) подібний предикату write. Однак предикат writeg розміщує ім’я атома або функціональний елемент в лапки, коли необхідно, щоб цей терм був в формі, прийнятній для предиката read. Предикат writeg також відображує рядки між обмежувачами в вигляді знаків долара ($).

Предикат display(+Term) використовують, щоб записати терми в префіксній формі для стандартного пристрою виведення без аналізу визначень операторів, як це роблять write і writeg. Таким чином, всі терми представляються в префіксному запису.

Предикат get0(-Char) зчитує символ зі стандартного пристрою введення. Він може бути використаний для зчитування будь-якого символу або для пошуку спеціального символу. Він повертає значення символу, якщо аргументу не приписано значення. Якщо ж аргументу приписано значення, то буде зчитуватися наступний символ.

get(-Char) як і get0, зчитує символ зі стандартного пристрою введення, але ігнорує символи, які не відображуються. Предикат get0_noecho(-Char) зчитує символ зі стандартного пристрою введення без його відображення на стандартному пристрої виведення.

Предикат skip(+Char) зчитує і пропускає знаки, поки не буде знайдено спеціальний знак.

Окремі знаки від предикатом put(+Char). Наприклад, можна почути звуковий сигнал термінала за запитом:

?-put(17).

Предикат nl переводить рядок на стандартному пристрої виведення.

Предикат tab(+Num) записує задане число пропусків (до 255) в стандартний пристрій виведення.

Предикат tmove(+Row, +Column) переміщує курсор в задану позицію поточного вікна.

.

Організація діалогів та циклів за зворотним переглядом

Найпростіший діалог можна організувати предикатом користувача dial(+Msg, +Ans), який вводить терм або структуру у відповідь на задане повідомлення.

dial(M, A):- write(M), write('='), read(A), nl.

Циклічний діалог або навіть перегляд БЗ з обробкою внутрішніх можна організувати за допомогою зворотного перегляду позалогічними предикатами, в тому числі, предикатами введення-виведення, предикатами доступу до БЗ, а також предикатами виділеними дужками [!...!].

dialrf(M, A):- pict(M), write('='), read(A), nl.

Такі предикати обробляються лише в основному напрямку обробки.

Аналіз синтаксису і семантики аргументів предикатів

Синтаксис і семантика аргументів грають ключову роль при побудові інтелектуальних систем (ІС). До синтаксичних об'єктів відносять допустимі формати представлення інформації, а до семантичних – правила їх інтерпретації в досліджуваній ІС. Таким чином, при просуванні вгору за рівнями інтелектуальності задач відбувається зміщення семантичних правил в область синтаксису. Зворотний рух можна розглядати як семантичну деталізацію синтаксичних правил високого рівня.

На рівні запитів в рамках базової системи SWI-Prolog до синтаксису можна віднести типи термів, що вводяться, та інших елементів вбудованої БД в еталонному прикладі наведена процедура семантичної підказки form_goal. Подібним чином можна побудувати процедуру синтаксичної перевірки правильності запитів.

Базові елементи реляційної алгебри

Подібному аналізу можуть підлягати не тільки фактична інформація з БД, але і правила, побудовані програмістом при складанні програм, які заносяться в БЗ для розв'язання задач.

Реляційна алгебра включає 5 базових операцій [5], які в Prolog-системі задаються наступними правилами.

1. Об'єднання:

r_union_s(X1,X2,...,Xn) :- r(X1,X2,...,Xn).

r_union_s(X1,X2,...,Xn) :- s(X1,X2,...,Xn).

2. Симетрична різниця:

r_diff_s(X1,X2,...,Xn) :- r(X1,X2,...,Xn), not s(X1,X2,...,Xn).

r_diff_s(X1,X2,...,Xn) :- r(X1,X2,...,Xn), not s(X1,X2,...,Xn).

3. Декартів добуток:

r_x_s(X1,X2,...,Xmn) :- r(X1,X2,...,Xm), s(Xm1,Xm2,...,Xmn)

4. Проекція:

r1n(X1,Xn) :- r(X1,X2,...,Xn).

5. Вибірка:

rc(X1,X2,...,Xn) :- r(X1,X2,...,Xn), condition(X1,X2,...,Xn).

З широкого ряду похідних операцій частіше за все використовується:

6. Пересічення:

r_meet_s(X1,X2,...,Xn) :- r(X1,X2,...,Xn), s(X1,X2,...,Xn).

Використання подібних правил в інтелектуальній оболонці Prolog-системи для редагування може послужить основою для побудови CASE-системи (computer aided system engineering) при створенні ІС на базі мови Prolog.

Предикати перетворення об'єктів

Дуже часто необхідні перетворення даних з однієї форми в іншу. Базовий предикат, що маніпулює структурами і списками =.. був розглянутий попередній роботі. Цей предикат зручно використовувати, коли до структури

доповнюються нові аргументи. Їх можна додати до структури предикатом add_args, визначеним нижче. Предикат перетворювання використовується двічі: для прямого і зворотного перетворювання.

```
add_args(Struct,ArgList,NewStruct):- Struct=..List,
   append(List,ArgList,NewList),NewStruct=..NewList.
```

Предикат add_args можна використовувати для доповнення третього аргументу до структури book, як показано нижче.

```
?-add_args(book(pooh,milne),[aa],X).
  X=book(pooh, milne, aa).
```

Системний предикат аналізу функтора functor(?Struct, ?Name, ?Arity) аналізує структуру і повертає назву структури та арність. Наприклад, предикат functor розбирає структуру book, як показано нижче:

```
?-functor((book(pooh, milne, aa)),X,A).
X=book,
A=3.
```

Якщо аргумент Struct невизначений, предикат functor створює структуру, що має специфіковані Name і Arity (максимум 255). Аргументи структур можуть бути незв’язаними змінними.

Предикат arg(+N,+Term,-Value) повертає значення N-го, починаючи від 1 аргументу структури Term. Наприклад, повернути перший аргумент структури book можна наступним запитом:

```
?-arg(I, book(poetry, milne),X).
X=poetry.
```

Системний предикат arg також використовується для зв’язування змінних в структурі. Предикат Arg0(+N,+Term,-Value) ідентичний предикату arg, але на відміну від нього, аргументи нумеруються, починаючи з 0.

Системний предикат argrep(+Term,+N,+Arg,-NewStruct) замінює аргумент в структурі Term новим аргументом Arg і повертає нову структуру

в четвертому аргументі. Наприклад, для заміщення в структурі student(alan,soph,95) і аргументу 95 на 88 виконується запит:

?-argrep(student(alan,soph,95),3,88,X).

X=student(alan,soph,88)

Відзначимо, що argrep не змінить терм в БД, він тільки повертає нову структуру в NewStruct.

Системний предикат name(?Atom, ?List) перетворює атом або ціле число в список символів або список в атом, що залежить від того аргументу, який підлягає обробці. Коли Atom – це атом, List перетворюється в список кодів ASCII, який створює символ в назві атома. Коли атом ціле число, List формується як список кодів ASCII для відображення цілого числа. Коли List заданий як список в коді ASCII, створюється назва атома. Якщо обидва аргументи визначені предикат name перевірить рівність їх символьних представлень. Наприклад:

. ?-name(quiche,L).

L=[113,117,105,99,104,101]

Yes

?-name(X[114]).

X=r

Предикат length(+List,-N) визначає довжину списку. Аргумент List повинен мати значення в вигляді списку заданої довжини. N – задається змінною, в якій length повертає довжину списку. Наприклад, за наявності списку символів цей предикат виконує наступні дії:

?-length("tennis", N).

N=6.

Оператори для аналізу бази знань

Окрема група предикатів забезпечує обробку речень в БЗ, які називають твердженнями. Ці предикати розуміють синтаксис операторів і дозволяють

використовувати метод від стенографії для відповідних операторів в БЗ. При додаванні твердження в БЗ їх назва і арність не повинна співпадати з ключами для тих термів, які зберігаються в тому самому середовищі.

Предикат current_predicate(?Predicate) уніфікує Predicate через зворотний перегляд з іменем або зі структурою name/arity предиката, занесеного до БЗ.

Предикат clause(+Head,-Body) уніфікує через зворотний перегляд Head з заголовком речення та Body з тілом речення. Аргумент Head повинен бути визначеним. Фрази фактів повертають атоми true як їх тіла. Предикат clause при зворотному пошуку, повертає все оператори процедури, які визначаються за допомогою Head і Body.

Предикат asserta(+Clause) додає твердження, задане в аргументі, в початок відповідної процедури. Якщо специфіковане твердження не має тіла, asserta визначає тіло як true. Цей предикат допомагає контролювати порядок, в якому твердження розміщуються в БЗ. Наприклад, при введенні другого речення предиката append, а не перший, можна використати asserta для додавання першого речення БД в його відповідне місце:

?-asserta((append([],X,X)))

Дублювання круглих дужок, що використовується в asserta, викликано пріоритетом оператора ':-'.

Предикат assertz(+Clause) додає твердження в кінець процедури і за синтаксисом співпадає з asserta. Предикат assert(+Clause) співпадає з assertz.

Предикат retract(+Clause) виключає з процедури БЗ перше речення, заголовок якої специфікується аргументом.

Предикат abolish(+Clause) виключає з БЗ всі речення процедури, що специфікується назвою і арністю.

Вікно може займати цілий екран або частину екрана. Будь-яке Вікно може займати цілий екран або частину екрана. Будь-яке вікно при створенні з’являється на задньому плані, і переводиться на передній, коли стає поточним. Тільки одне вікно може бути поточним і все управління екрана, і

операції введення-виведення використовують це вікно. В поточному вікні використовують наступні предикати: write, writeg, display, put, nl і tab, а також предикати управління курсором. Наприклад, система Arity/Prolog супроводжує до 32 заданих вікон, з яких 5 вікон використовує інтерпретатор, по одному вікну – кожний стандартний і вписаний блок діалогу. Форма вікна варіюється наступні можливості: подвійний або простий бордюр або без бордюру, включення тексту у верхній лівій межі вікна, використання атрибутів границі і заднього плану вікна.

Предикат створення нового вікна має наступний синтаксис.

define_window(+Name,?Label,?(ULR,ULC),?(LRR,LRC),

?(Window_attr,Border_attr))

Опис еталонних програм

Наведена процедура lab4, використовуючи вбудовані предикати для організації діалогу, виконує аналіз термів, що вводяться і визначає належність їх типів. Предикат a_term виконує перевірку типів аргументів, що аналізується. Предикат form_goal формують цілі за шаблонами семантичної підказки pict. Ці предикати можна використати для організації аналізу синтаксису і семантики.

Тому при висхідному розборі можна приймати до уваги тільки самі оператори (операції), що дозволяє одержати спрощені варіанти алгоритмів.

Джерела:

1. "Логічне програмування" - https://uk.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D1%96%D1%87%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F.
2. Пустоваров В.І. Методичні вказівки до виконання лабораторних робіт для студентів напрямку 6.050102 "Комп'ютерна інженерія" - К.: КПІ, 2014.- 136 с.