МУЛЬТИПАРАДИГМЕННЕ ПРОГРАМУВАННЯ

Лекція 5

ПРОГРАМУВАННЯ НА МОВІ ПРОЛОГ (2)

ЛОГІЧНЕ ПРОГРАМУВАННЯ ПРОДУКЦІЯХ

Списки

Списки - одна з найчастіше вживаних структур у Пролозі. При записі список беруть у квадратні дужки, а елементи списку розділяють комами, наприклад, **[слон, кінь, мавпа, собака]**

Це список із чотирьох атомів - слон, кінь, мавпа, собака.

Елементами списку можуть бути будь-які терми Прологу, тобто атоми, числа, змінні й складні терми, що дозволяє, зокрема, складати списки зі списків.

Порожній список записується як **[ ]**.

[слон, [ ], X, предок(X, том), [a,b,c], f(22)]

Перший елемент непорожнього списку називається *головою*, а інша частина списку зветься *хвіст*. У списку, що складається тільки з одного елемента, головою є цей єдиний елемент, а хвостом - порожній список.

Позначення **[H|T]** використовується для визначення списку з головою **H** і хвостом **T**. Якщо символ **|** розміщений перед останнім термом списку, то це означає, що цей останній терм визначає інший список. Повний список вийде, якщо з'єднати цей підсписок з послідовністю елементів, розташованих до цієї риси.

В наступному прикладі **1** - голова списку, а **[2, 3, 4, 5]** - хвіст. Пролог покаже це за допомогою співставлення списку чисел зі зразком, що складається з голови й хвоста.

?- [1, 2, 3, 4, 5] = [Head | Tail].
Head = 1
Tail = [2, 3, 4, 5]
Yes

Тут **Head** і **Tail** - тільки імена змінних. Ми так само могли б використати **X** і **Y** або які-небудь інші імена змінних. Помітимо, що хвіст списку завжди є списком. Голова, у свою чергу, є елемент списку, що вірно й для всіх інших елементів, розташованих до вертикальної риси. Це дозволяє одержати, скажімо, другий елемент списку.

Приклад

Використаємо анонімні змінні для голови й списку, що стоїть після риси, якщо нам потрібний тільки другий елемент списку:

?- [слон, кінь, осел, собака] = [_, X | _].

X = кінь

Yes

Розглянемо кілька процедур обробки списків. Зверніть увагу, що всі вони використовують рекурсію, у якій термінальне (базове) правило визначене для порожнього списку.

Приклад

Напишемо предикат для обчислення суми всіх елементів списку чисел.

```
сума_списку([],0).
сума_списку([H|T],S):- number(H),
сума_списку(T,S1),
S is S1+H.
```

Приклад

Предикат місце успішний, якщо третій аргумент є список, який
отриманий вставкою першого аргументу в довільне місце списку, що є другим аргументом.

місце(E, L, [E|L]).

місце(E, [H|L], [H|Y]):- місце(E, L,Y).

Подивимося на результати деяких запитів, що використовують цей предикат.

```
?- місце(1,[2,3],X).
X = [1, 2, 3] ;
X = [2, 1, 3] ;
X = [2, 3, 1] ;
No
?- місце(1,L,[2,1,3]).
L = [2, 3] ;
No
?- місце(X,[2,3],[2,1,3]).
X = 1 ;
No
```

Приклад

Предикат **перестановка** видає списки, отримані перестановкою елементів свого першого аргументу.

```
перестановка([],[]).
перестановка([H|L],Z):- перестановка(L,Y),
місце(H,Y,Z).
```

Приклад використання:

```
?- перестановка([a,b,c],X).
X = [a, b, c] ;
X = [b, a, c] ;
X = [b, c, a] ;
X = [a, c, b] ;
X = [c, a, b] ;
X = [c, b, a] ;
No
```

І, нарешті, приведемо правило для друку всіх можливих перестановок списку:

всі_перестановки(L):- перестановка(L,R), write(R), nl, fail.

Перша підціль предиката обчислює чергову перестановку, друкує її й переходить до останньої підцілі - **fail**. Ця підціль завжди неуспішна, що змушує Пролог повернутися до початку правила й продовжити пошук рішення. Робота процедури завершиться, коли всі перестановки будуть вичерпані:

```
?- всі_перестановки(['маркіза', 'ваші прекрасні очі', | 'обіцяють мені смерть від любові']).
[маркіза, ваші прекрасні очі, обіцяють мені смерть від любові]
[ваші прекрасні очі, маркіза, обіцяють мені смерть від любові]
[ваші прекрасні очі, обіцяють мені смерть від любові, маркіза]
[маркіза, обіцяють мені смерть від любові, ваші прекрасні очі]
```

[обіцяють мені смерть від любові, маркіза, ваші прекрасні очі]
[обіцяють мені смерть від любові, ваші прекрасні очі, маркіза]
No

Приклад

У давньояпонському календарі був прийнятий 60-річний цикл, що складається з п'яти 12-річних підциклів. Підцикли позначалися назвами кольорів: зелений, червоний, жовтий, білий і чорний. Усередині кожного підцикла роки носили назви тварин: пацюк, корова, тигр, заєць, дракон, змія, кінь, вівця, мавпа, курка, собака й свиня. Наприклад, 1984 рік - рік початку чергового циклу - називався Роком Зеленого Пацюка.

Складемо програму, що по заданому номеру року нашої ери **n** друкує його назву в давньояпоському календарі. Розглянемо два випадки: (1) значення **n** не менше, ніж 1984; (2) значення **n** - будь-яке натуральне число.
Скористаємося вбудованим предикатом **nth0(індекс, список, елемент)**, що буде успішним, якщо елемент перебуває на місці з номером індекс, рахуючи від **0**. Для випадку (1) використаємо предикат **nam**, для випадку (2) предикат – **nm**.

```
color(N,X):- N1 is ((N-1984) mod 60)//12,
nth0(N1, ['зелений', 'червоний', 'жовтий',
'білий', 'чорний'],X).
animal(N,X):- N1 is (N-1984) mod 12,
nth0(N1,
['пацюк', 'корова', 'тигр', 'заєць', 'дракон', 'змія',
'кінь', 'вівця', 'мавпа', 'курка', 'собака', 'свиня'],
X).
nam(N,[X,Y]):- number(N), color(N,X), animal(N,Y).
nm(N,X):- N>1983, nam(N,X).
nm(N,X):- N<1984, N1 is N+60, nm(N1,X).
```

ПРОДУКЦІЙНЕ ПРОГРАМУВАННЯ

Мови логічного програмування, засновані на правилах (rule-based), є найпоширенішими й більше 80% ЕС використають саме їх.

Визначення. Продуційна модель або модель, заснована на правилах, дозволяє представити знання у вигляді пропозицій типу "Якщо (умова), то (дія)".

Під "умовою" (*антецедентом*) розуміється деяка пропозиція-зразок, по якому здійснюється пошук у базі знань, а під "дією" (*консеквентом*) - дії, виконувані при успішному результаті пошуку (вони можуть бути проміжними, що виступають далі як умови, і термінальну або цільову, завершальну роботу системи).

Найчастіше висновок на такій базі знань буває прямій (від даних до пошуку мети) або зворотний (від мети для її підтвердження — до даних). Дані — це вихідні факти, що зберігаються в базі фактів, на підставі яких запускається машина висновку або інтерпретатор правил, що перебирає правила із продуційної бази знань.

Продукційна, модель так часто застосовується в промислових експертних системах, оскільки залучає розробників своєю наочністю, високої модульності, легкістю внесення доповнень і змін і простотою механізму логічного висновку.
Є велика кількість програмних засобів, що реалізують продуційний підхід (наприклад, мови високого рівня CLIPS й OPS 5; "оболонки" або "порожні" ЕС - EXSYS Professional і Kappa, інструментальні системи KEE, ARTS, PIES), а

також промислових ЕС на його основі (наприклад, ЕС, створених засобами G2).

Висновок на знаннях

Найбільше поширення одержала продуційна модель подання знань. При її використанні база знань складається з набору правил, а програма, що управляє перебором правил, називається *машиною висновку*.

> *Машина висновку* (інтерпретатор правил) — це програма, що імітує логічний висновок експерта, що користується даною продуційною базою знань для інтерпретації даних, що надійшли в систему.

Зазвичай вона виконує дві функції:

- перегляд існуючих даних (фактів) з робочої пам'яті (бази даних) і правил з бази знань і додавання (у міру можливості) у робочу пам'ять нових фактів;
- визначення порядку перегляду й застосування правил. Цей механізм управляє процесом консультації, зберігаючи для користувача інформацію про отримані висновки, і запитує в нього інформацію, коли для спрацьовування чергового правила в робочій пам'яті виявляється недостатньо даних.

У переважній більшості систем, заснованих на знаннях, механізм висновку являє собою невелику по об'єму програму й включає два компоненти - один реалізує безпосередньо висновок, інший - управляє цим процесом.

Дія *компонента висновку* засновано на застосуванні правила, називаного *modus ponens*: "Якщо відомо, що правдиве твердження А, і існує правило виду "ЯКЩО А, ТО В", тоді твердження В також істина".

Таким чином, правила спрацьовують, коли перебувають факти, що задовольняють їхній лівої частини: якщо щиро посилку, то повинне бути істинно й висновок.

*Компонент висновк*у повинен функціонувати навіть при недоліку інформації. Отримане рішення може й не бути точним, однак система не повинна зупинятися через те, що відсутня яка-небудь частина вхідної інформації.

Керуючий компонент визначає порядок застосування правив і виконує чотири функції:

1.*Зіставлення*— зразок правила зіставляється з наявними фактами.

2.*Вибір* — якщо в конкретній ситуації можуть бути застосовані відразу кілька правил, то з них вибирається одне, найбільш підходяще за заданим критерієм (дозвіл конфлікту).

3.*Спрацьовування* — якщо зразок правила при зіставленні збігся з будь-якими фактами з робочої пам'яті, те правило спрацьовує.

4. *Дія* — робоча пам'ять піддається зміні шляхом додавання в неї висновку правила, що спрацювало.

Якщо в правій частині правила втримується вказівка на яку-небудь дію, то воно виконується (як, наприклад, у системах забезпечення безпеки інформації).

Інтерпретатор продукцій працює циклічно. У кожному циклі він переглядає всі правила, щоб виявити тих, посилки яких збігаються з відомими на даний момент фактами з робочої пам'яті. Після вибору правило спрацьовує, його висновок заноситься в робочу пам'ять, і потім цикл повторюється спочатку.

В одному циклі може спрацювати тільки одне правило. Якщо кілька правил успішно зіставлені з фактами, то інтерпретатор робить вибір за певним критерієм єдиного правила, що спрацьовує в даному циклі. Цикл роботи інтерпретатора схематично представлений на мал. 5.1.

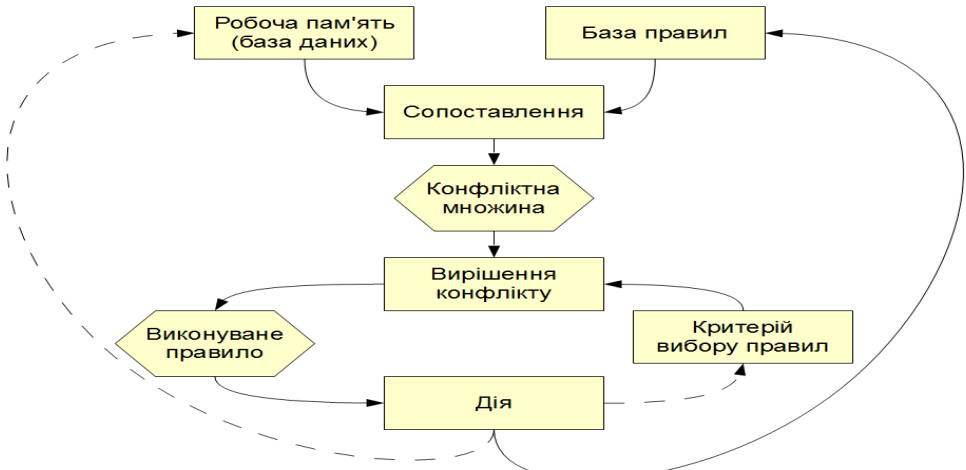

Рис. 5.1. Цикл роботи інтерпретатора

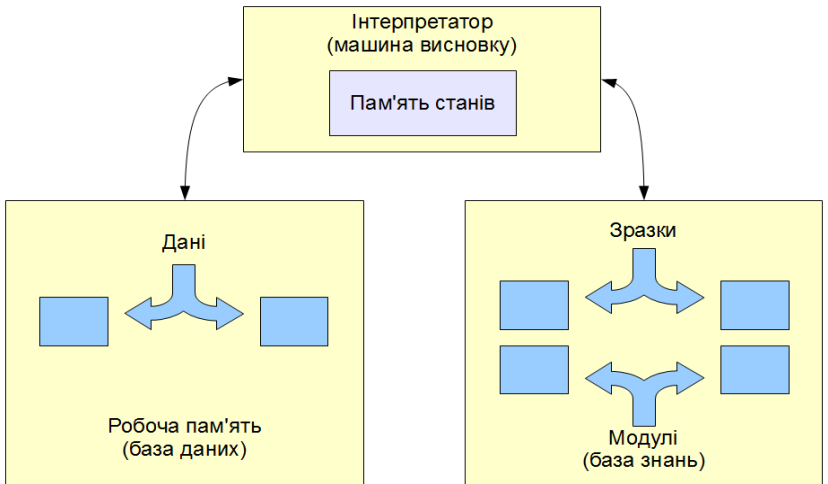

Рис. 5.2. Схема функціювання інтерпретатора

Інформація з робочої пам'яті послідовно зіставляється з посилками правил для виявлення успішного зіставлення. Сукупність відібраних правил становить так називану *конфліктну множину*. Для дозволу конфлікту інтерпретатор має критерій, за допомогою якого він вибирає єдине правило, після чого воно спрацьовує. Це виражається в занесенні фактів, що утворять висновок правила, у робочу пам'ять або в зміні критерію вибору конфліктуючих правил.

Якщо ж на закінчення правила входить назва якої-небудь дії, то воно виконується.

Робота машини висновку залежить тільки від стану робочої пам'яті й від складу бази знань. На практиці звичайно враховується історія роботи, тобто поводження механізму висновку в попередніх циклах. Інформація про поводження механізму висновку запам'ятовується в пам'яті станів (мал. 5.2). Звичайно пам'ять станів містить протокол системи.

У системах із *прямим висновком* по відомих фактах відшукується висновок, що із цих фактів треба . Якщо такий висновок вдається знайти, то воно заноситься в робочу пам'ять. Прямий висновок часто називають висновком, керованим даними, або висновком, керованим антецедентами.

Існують системи, у яких висновок ґрунтується на сполученні згаданих вище методів - зворотного й обмеженого прямого. Такий комбінований метод одержав назву циклічного.

Нехай є фрагмент бази знань із двох правил:

- П1: Якщо "відпочинок - улітку" й "людина - активна", то "їхати в гори".

- П2: Якщо "любить сонце", те "відпочинок улітку".

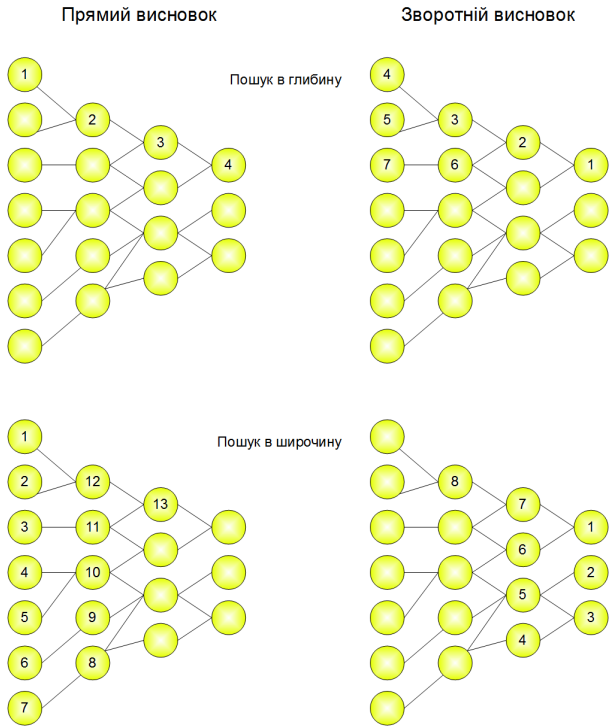

Рис. 5.3. Стратегії висновку

Припустимо, у систему надійшли факти - "людина активна" й "любить сонце".

ПРЯМИЙ ВИСНОВОК— виходячи з фактичних даних, одержати рекомендацію.

- 1-й прохід.

- *Крок 1.* Пробуємо /7/, не працює (не вистачає даних "відпочинок - улітку").
- *Крок 2.* Пробуємо /72, працює, у базу надходить факт "відпочинок - улітку".

- 2-й прохід.

- *Крок 3.* Пробуємо *Я*/, працює, активізується мета "їхати в гори",
  що і виступає як рада, що дає ЕС.

ЗВОРОТНИЙ ВИСНОВОК— підтвердити обрану мету за допомогою наявних правил і даних.

- 1-й прохід.

- *Крок 1.* Ціль — "їхати в гори": пробуємо *П1* — даних "відпочинок — улітку"
  ні, вони стають новою метою й шукається правило, де вона в лівої частини.
- *Крок 2.* Ціль "відпочинок — улітку": правило *П2* підтверджує мета й активізує її.

- 2-й прохід.

- Крок 3. Пробуємо П1, підтверджується шукана мета.

Методи пошуку в глибину й завширшки

У системах, база знань яких нараховує сотні правил, бажаним є використання стратегії керування висновком, що дозволяє мінімізувати час пошуку рішення й тим самим підвищити ефективність висновку. До числа таких стратегій ставляться: пошук у глибину, пошук завширшки, розбивку на підзадачі й альфа-бета-алгоритм.

При пошуку в глибину в якості чергової підцілі вибирається та, котра відповідає наступний, більше детальному рівню опису задачі. Наприклад, що діагностує система, зробивши на основі відомих симптомів припущення про наявність певного захворювання, буде продовжувати запитувати уточнювальні ознаки й симптоми цієї хвороби доти, поки повністю не спростує висунуту гіпотезу.

При пошуку завширшки, навпроти, система спочатку проаналізує всі симптоми, що перебувають на одному рівні простору станів, навіть якщо вони ставляться до різних захворювань, і лише потім перейде до симптомів наступного рівня детальності.

Розбивка на підзадачі має на увазі виділення підзадач, рішення яких розглядається як досягнення проміжних цілей на шляху до кінцевої мети. Прикладом, що підтверджує ефективність розбивки на підзадачі, є пошук несправностей у комп'ютері - спочатку виявляється підсистема, що відмовила (живлення, пам'ять і т.д.), що значно звужує простір пошуку.

Якщо вдається правильно зрозуміти сутність задачі й оптимально розбити її на систему ієрархічно зв'язаних целей-підцілей, то можна домогтися того, що шлях до її рішення в просторі пошуку буде мінімальний.

Альфа-бета-алгоритм дозволяє зменшити простір станів шляхом видалення галузей, не перспективних для успішного пошуку. Тому проглядаються тільки ті вершини, у які можна потрапити в результаті наступного кроку, після чого безперспективні напрямки виключаються. Альфа-бета-алгоритм знайшов широке застосування в основному в системах, орієнтованих на різні ігри, наприклад, у шахових програмах.

В наступній лекції ми розглянемо систему продукційного програмування CLIPS.